ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики і інформатики

«До захисту допущено»
Завідувачка кафедри ПМІ

Ольга ДМИТРІЄВА
(підпис) (ім'я та прізвище)

«17» червня 2022 р.

Випускна кваліфікаційна робота
бакалавра
(освітній ступень)

на тему
«[на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля»»
зі спецчастиною
«Розробка програмних модулів, інтерфейсу користувача та бази даних»

Виконав: студент 4 курсу, групи ІПЗ-18

Спеціальності 121 Інженерія програмного забезпечення

Бервін М.С. _____
(прізвище та ініціали) (підпис)

Керівник доц. каф. ПМІ, к.т.н., доц. Ірина НАЗАРОВА _____
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент доц. каф. КІ, к.т.н., доц. Віталій ШАМАЄВ _____
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Нормоконтроль:

_____ Ірина НАЗАРОВА
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Луцьк – 2022 р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерних наук і технологій
Кафедра прикладної математики та інформатики
Освітньо-кваліфікаційний рівень (освітній ступінь) бакалавр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувачка кафедри ПМІ

_/Ольга ДМИТРІЄВА
"6".05.2022 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Михайлу Бервіну

1. Тема проєкту (роботи) Автоматизована система обліку документів на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля».
Спецчастина Розробка програмних модулів, інтерфейсу користувача та бази даних.
Керівник проєкту (роботи) Ірина НАЗАРОВА, к.т.н, доц., доц._каф. ПМІ
(ім'я, прізвище, науковий ступінь, вчене звання, посада)
затверджені наказом від "06" травня 2022 року №180
2. Строк подання студентом проєкту (роботи) 17.06.2022
3. Вихідні дані до проєкту (роботи) результати науково-дослідної роботи, матеріали переддипломної практики.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1) аналіз предметної області і постановка завдання;
   2) проєктування бази даних;
   3) проєктування системи;
   4) розробка інтерфейсу користувача;
   5) розробка алгоритму заповнення шаблонів документів;
   6) тестування додатку.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
   1) блок-схема розробленого алгоритму заповнення шаблонів документів;
   2) діаграма варіантів використання;
   3) екранні форми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Підпис, дата	Підпис, дата

7. Дата видачі завдання 6 квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Об'єктний аналіз предметної області і постановка завдання	10.04.2022	
2.	Проєктування бази даних	13.04.2022	
3.	Проєктування системи	16.04.2022	
4.	Розробка інтерфейсу користувача	28.04.2022	
5.	Розробка алгоритму заповнення шаблонів документів	15.04.2022	
6.	Тестування додатку	26.05.2022	
7.	Оформлення пояснювальної записки	01.06.2022	
8.	Підготовка слайдів презентації	05.06.2022	
9.	Підготовка демонстраційної версії розробленого програмного продукту	12.06.2022	
10.	Написання доповіді	17.06.2022	

Студент _____ Михайло БЕРВІН_
(підпис) (ім'я, прізвище)

Керівник роботи _____ Ірина НАЗАРОВА_
(підпис) (ім'я, прізвище)

АНОТАЦІЯ

Михайло Бервін. Розробка автоматизованої системи обліку документів на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля» / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення ДВНЗ ДонНТУ, Луцьк, 2022.

Об’єктом дослідження є процеси проектування, розробки тестування систем обліку документів.

Предметом дослідження є методи та алгоритми розробки систем обліку документів.

Метою роботи є розробка автоматизованої системи обліку документів на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля».

Практичне значення розробки автоматизованої системи обліку документів на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля» полягає в створені програмного додатку, основними завданнями якого є автоматизація заповнення типових документів.

Методи дослідження, що застосовані при розробці автоматизованої системи обліку документів: метод порівнянь, метод проєктування програмного забезпечення, метод проєктування баз даних та емпіричний метод. Реалізація автоматизованої системи обліку документів здійснювалася завдяки основним положенням, теорії алгоритмів, теорії об’єктно-оріентованого програмування та модульного програмування

Ключові слова: автоматизована система, документи, облік документів, SQLite, Qt, C++, COM, OLE, відокремлений підрозділ, «Шахта Україна», підприємство, «Селидіввугілля».

ANNOTATION

Mykhailo Bervin. Development software system for automating accounting of document at the department "Shahta Ukraine" of the state enterprise "Selidivvuhilya" / Final qualification work for obtaining the educational qualification level "bachelor" in specialty 121 Software Engineering of DonNTU, Lutsk, 2022.

The object of research is the processes of design, development and testing software system for automating accounting of document.

The subject of research is the methods and algorithms of automated document accounting system development.

The aim of the work is to develop the software system for automating accounting of document at the department "Shahta Ukraine" of the state enterprise "Selidivvuhilya".

The practical significance of developing the software system for automating accounting of document at the department "Shahta Ukraine" of the state enterprise "Selidivvuhilya" to automate the completion of standard documents.

Research methods used in the development of the automated document accounting system: the method of comparisons, the method of software design, the method of database design and the empirical method. Implementation of the automated document accounting system was carried out due to the basic provisions of the theory of algorithms, the theory of object-oriented programming and modular programming

Keywords: automated system, documents, document accounting, SQLite, Qt, C ++, COM, OLE, department, "Shahta Ukraine", state enterprise, "Selidivvuhilya".

ЗМІСТ

ВСТУП

Бухгалтери багато часу витрачають на індивідуальне заповнення однотипних документів. Відпускні, лікарняні, та інші документи заповнюються доволі часто та індивідуально під кожного працівника. При цьому, це може займати доволі багато часу та не виключає можливі помилки від однотипності праці.

Тож автоматизована система обліку документів повисіть якість та швидкість праці.

Враховуючи вищевказане обґрунтування темою кваліфікаційної роботи обрано розробку автоматизованої системи обліку документів на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля».

Об’єкт дослідження – процес проектування та розробки систем обліку документів.

Мета роботи – розробка автоматизованої системи обліку документів та бази даних.

1 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Вибір теми роботи та її актуальність

Бухгалтери багато часу витрачають на індивідуальне заповнення однотипних документів. Відпускні, лікарняні та інші документи заповнюються доволі часто та індивідуально під кожного працівника. При цьому, це може займати доволі багато часу та не виключає можливі помилки від однотипності праці. Тому автоматизована система обліку документів в рамках будь-якого підприємства підвищить якість та швидкість праці.

В якості теми кваліфікаційної роботи було обрано розробку автоматизованої системи обліку документів на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля».

1.2 Мета роботи

Метою роботи є розробка автоматизованої системи обліку документів та бази даних для автоматизованої системи обліку документів на відокремленому підрозділі «Шахта Україна» державного підприємства «Селидіввугілля».

1.3 Практичне завдання роботи

Практичне значення розробки автоматизованої системи обліку документів полягає у створені програмного додатку, основними завданнями якого є заповнення шаблонів документів, гнучке налаштування шаблонів документів для обраного підрозділу.

1.4 Методи дослідження

Методи дослідження, що застосовані при розробці додатку: метод порівнянь, метод проєктування програмного забезпечення, метод проєктування баз даних та емпіричний метод. Реалізація додатку

здійснювалася завдяки основним положенням теорії алгоритмів, теорії об’єктно-оріентованого програмування та модульного програмування

1.5 Обґрунтування методів дослідження

Емпіричний метод [1] дослідження був реалізований для отримання інформації, щодо структуризації системи та її проєктування. Метод порівнянь [2] був застосований для аналізу конкурентного середовища. Метод проєктування програмного забезпечення було використано для процесу розробки. За допомогою методу проєктування баз даних були створенні бази даних.

1.6 Аналіз конкурентів у предметній області

На підприємстві існує внутрішня система обліку документів. Вона була, також, розроблена як дипломна робота у 1997 році. Додаток працює тільки на операційній системі Windows95, має застарілі форми документів без можливості їх змін, що не дозволяє його використовувати з документами, які зазнали змін з 1997 року. Також, був загублений сирцевий код, що унеможливлює редагування додатку, додавання та оновлення форм документів.

Також існує доволі багато автоматизованих систем обліку документів.

Одним із найпопулярніших будуть рішення BAS [3]. Вони включають облік документів, як електронний, так і фізичний, має у складі моніторингові системи та звітності на їх основі. Можуть бути встановленими на сервері підприємства. Но, для доступу до сервісів необхідно вступи до «Спілки автоматизаторів бізнесу», дати пройти одному зі співробітників дводенний курс, котрий коштує від 700 гривень та обговорити інтегрування.

На рисунку 1.1 зображено демо-версію однієї з систем BAS – «BAS Бухгалтерія».

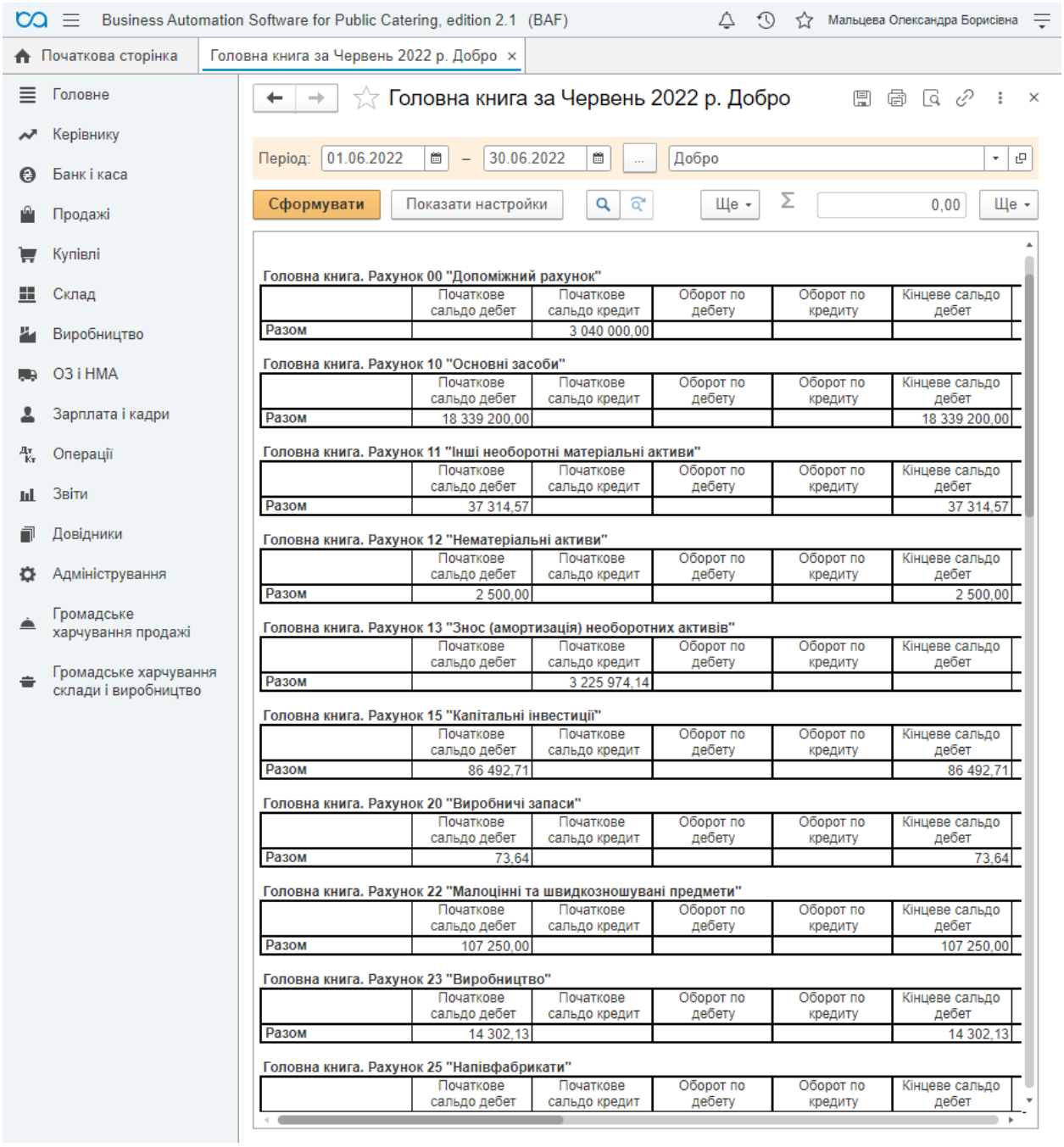


Рисунок 1.1 – Демо-версія «BAS Бухгалтерія»

Також, одним з найпопулярніших є сервіс «Вчасно» [4]. Він може підписувати документи, надсилати документи, робити коментарі, перевіряти документи антивірусом, зберігає документи у хмарному архіві, розподіляє індивідуальні права між співробітниками. Сам сервіс пропонує роботу через сайт, так й інтегрування з обліковою системою підприємства. Також, він працює як з персональних комп’ютерів, так й з мобільних пристроїв. Однак,

сервіс не є безкоштовним, а його вартість коштує більше 1600 гривень у місяць, або, у разі інтеграції, одноразову сплату 3000 гривень та подальшу плату 2,40 гривень за документ. Також, при підключені сервісу, підприємство повинне перейти на практично повністю електронний документообіг.

Крім них, сервіси компанії «Актив-Софт» [5] є доволі популярним:

– «FREDO:ДокМен» – сервіс для обміну торгівельними угодами;

– «FREDO:Звіт» – сервіс для формування та подачі регламентованої звітності;

– «СОТА» – веб-сервіс для формування та подачі податкової та бухгалтерскої звітності.

Також, є сервіси інтеграції с платформою BAS.

Загалом, сервіси у більшій мірі направлені на електронний документообіг, проте мають можливості для роздрукування. Також, сервіси є подрібненими та вузько направленими. Один з сервісів буде коштувати від 150 до 800 гривень на місяць.

1.7 Технічні особливості реалізації роботи

Реалізація додатку передбачає необхідність зберігання інформації про документи та їх поля, які заповнюються.

Функціонал додатку передбачає упорядкований збір інформації про документи, а саме назву, розташування шаблону та інформації заповнювання полів: ім’я, тип змінної та індекс фрагменту тексту, який заповнюється. Для впорядкованого збору інформації про документи та їх заповнювання необхідно застосувати технології COM (Component Object Model) та OLE (Object Linking and Embedding).

Також, для того, щоб зчитувати інформацію про працівників та автоматично заповнювати даними документи, функціонал додатку передбачає підключення до бази даних працівників підприємства.

Враховуючи ці вимоги, необхідно спроєктувати базу даних з таблицями документів та заповнюючих полів.

Шаблони документів також необхідно зберігати. Для цього слід виділити невелику кількість пам’яті на сервері, де буде розташовано додаток. Так, для зберігання 1 сторінки повністю текстового файлу формату .docx у середньому необхідно 20 КБ пам’яті.

1.7.1 Component Object Model

Microsoft Component Object Model (COM) — це незалежна від платформи, розподілена, об’єктно-орієнтована система для створення двійкових програмних компонентів, які можуть взаємодіяти [6].

Окрім визначення основного стандарту двійкових об’єктів, COM визначає деякі базові інтерфейси, які надають функції, спільні для всіх технологій на основі COM, і надають невелику кількість функцій, котрі потребуються всім компонентам. COM також визначає спільну роботу об’єктів в розподіленому середовищі та має додаткові функції безпеки, які допомагають забезпечити цілісність системи та компонентів.

1.7.2 Object Linking and Embedding

Object Linking and Embedding (OLE) дозволяють користувачам працюючим в одному додатку, маніпулювати даними, які записані в різних форматах та отриманих із декількох джерел [7].

Наприклад, користувач розробленого додатку може замінити частину тексту в документі, який створено в іншому додатку. Активація процесу заміщення приводить до того, що інший додаток загружає свій користувальницький інтерфейс або ту частину, яка має інструменти, необхідні для редагування об’єкта. Так, користувач може маніпулювати даними з зовнішніх джерел в контексті одного документа.

OLE опирається на основу, яка складається з COM, структурованого сховища та уніфікованої передачі даних.

1.8 Висновки за розділом

У цьому розділі було обґрунтовано вибір теми кваліфікаційної роботи, визначено її мету.

Також було проведено аналіз конкурентів, за результатами якого було виявлено, що нині працююча система обліку документів не справляється з більшістю задач, які перед нею постають, а рішення на ринку в основному потребують від підприємства змін у моделі обігу документів. Отже, нині працююча система потребує або заміни, або оновлення, що ускладнене відсутністю сирцевого коду, а впровадження одного з сучасних рішень призведе до глобальних змін у логіці обліку документів, щомісячними витратами на перекваліфікування кадрів та обслуговування ліцензій.

У розділі описані технічні особливості реалізації роботи та теоретичне обґрунтування для вирішення поставлених задач.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Архітектура програмної системи

Додаток матиме декілька функцій:

– додавання, видалення та редагування імені шаблону документа в системі;

– додавання, видалення та редагування полів для заміщення в шаблоні документа;

– заповнення полів шаблону документа;

– збереження, відкриття та роздрукування заповнених документів.

Додаток створює внутрішню базу даних та таблиці шаблонів документів та їх полів, якщо її не існувало до запуску.

Під час виконання функцій заповнення полів шаблону документів, збереження, відкриття та роздрукування заповнених документів, відбувається використання технологій COM та OLE, для роботи функцій пакету програм Microsoft Office.

Поля шаблону документа можуть заповнюватися як у ручному режимі, так й автоматично підставлятися та розраховуватися на основі даних, взятих з зовнішньої бази даних.

На рисунку 2.1 показана діаграма варіантів використання.

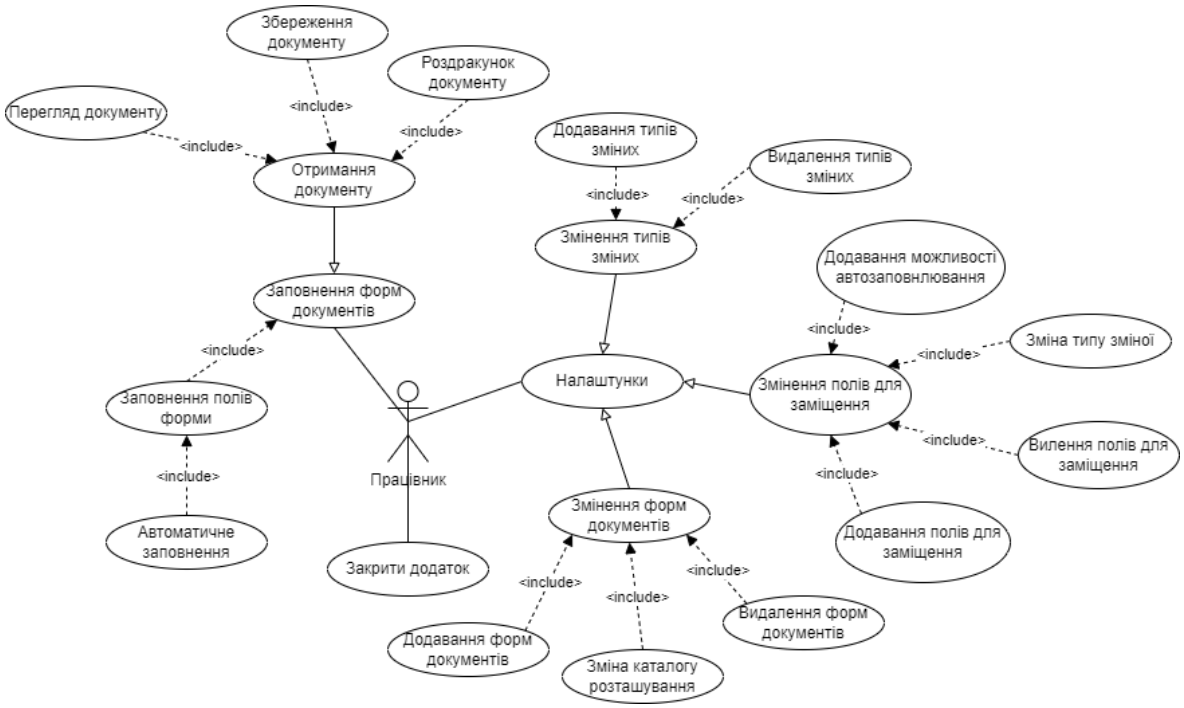


Рисунок 2.1 – Діаграма варіантів використання

Для запуску роботи додатку необхідно його розмістити на сервері або локальному комп’ютері. Щоб користувачі могли взаємодіяти з додатком їм необхідно лише встановити його. Згідно з цими вимогами створено діаграму розміщення (рисунок 2.3).

Рисунок 2.2 – Побудована діаграма розміщення

2.2 База даних

У стандартний пакет установки Qt входить модуль SQLite3.

SQLite – безсерверна, реляційна база даних. Для з’єднання з такою базою даних немає необхідності встановлювати сервер чи робити якісь налаштування [8].

Для більш зручного перегляду даних в таблицях додатково встановлено DB Browser (SQLite).

2.2.1 DB Browser (SQLite)

DB Browser – якісний графічний інструмент для створення, проєктування й редагування файлів бази даних SQLite [9].

Інтерфейс програми інтуїтивно зрозумілий. Програмний продукт дозволяє без знань мови SQL легко проводити необхідні маніпуляції з базою даних.

2.3 COM та OLE

Для реалізації функцій технологій COM та OLE використано бібліотеки Active Qt [10] та QFile [11].

ActiveQt – слугує для реалізації програмного каркасу ActiveX, який реалізовано на основі COM та OLE, з можливостями створення локальних та мережевих OLE-об’єктів.

QFile – слугує для зчитування та запису файлів. Функціонал бібліотеки використовується для створення копії шаблону документа, як буде заповнено.

За допомогою цієї бібліотеки робиться копія шаблону та його перейменування (рисунок 2.3).

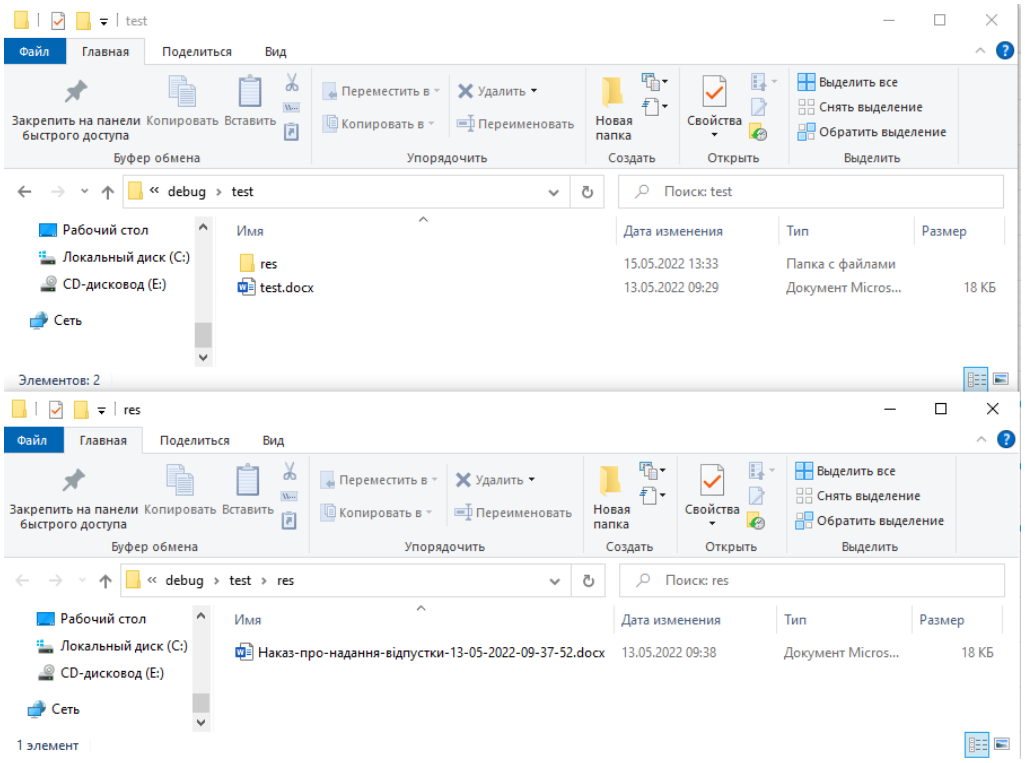


Рисунок 2.3 – Оригінальний та скопійований шаблон

В отриманому файлі, робиться підстановка значень, які введені через інтерфейс додатку, за допомогою функцій бібліотеки ActiveQt (рисуноки 2.4 – 2.6).

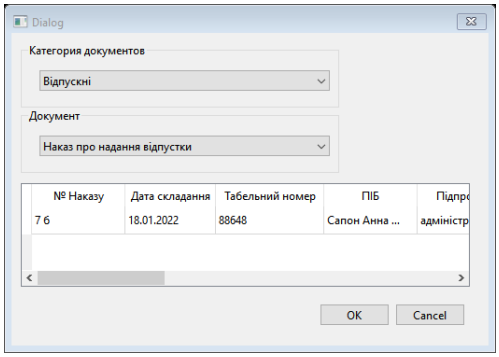


Рисунок 2.4 – Інтерфейс введення значень

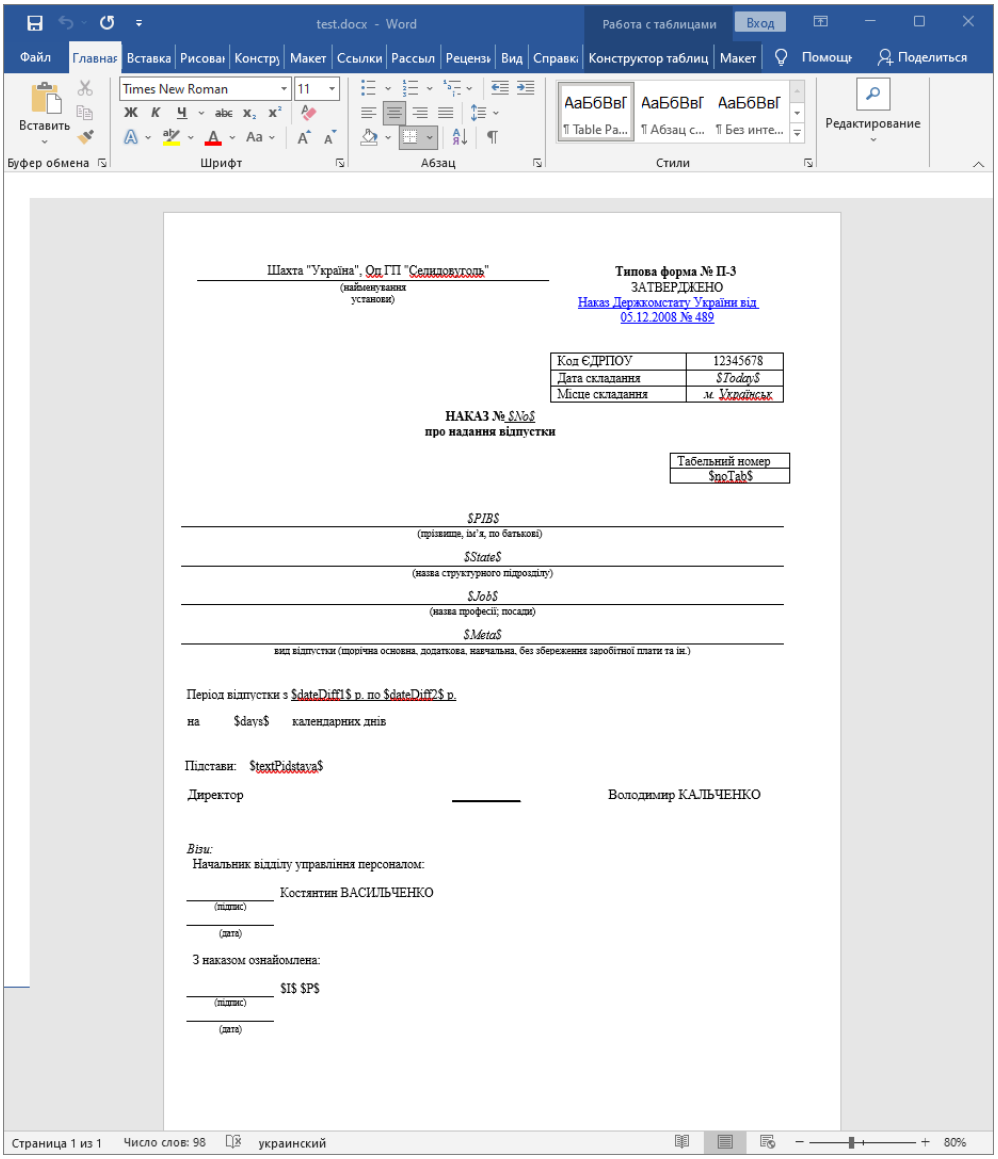
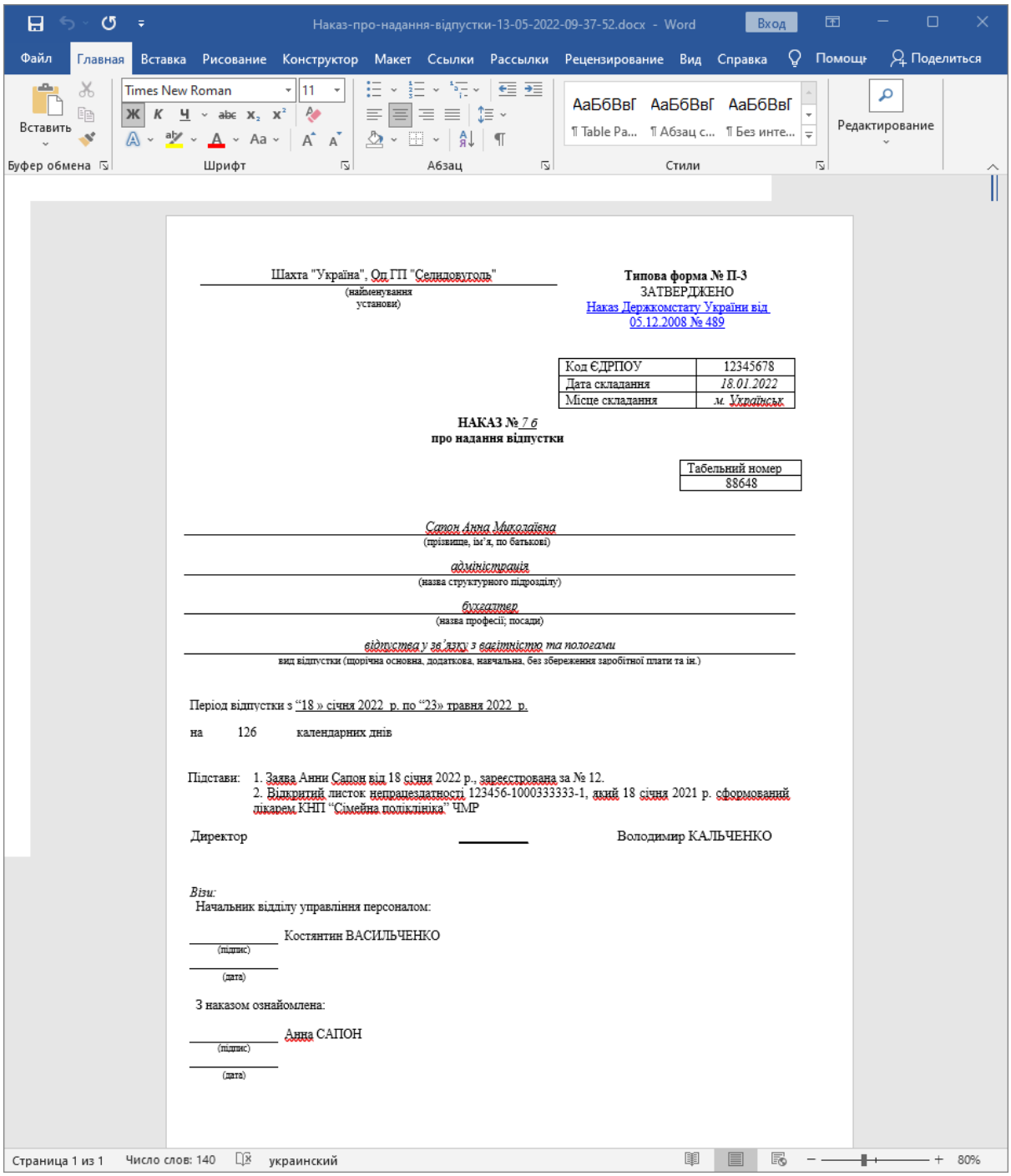


Рисунок 2.5 – Шаблон документу



Наказ-про-надання-відпустки-13-05-2022-09-37-52.docx - Word

Шахта "Україна", Оп ГП "Селидовуголь"
(найменування установи)

Типова форма № П-3
ЗАТВЕРДЖЕНО
Наказ Держкомстату України від 05.12.2008 № 489

Код ЄДРПОУ	12345678
Дата складання	*18.01.2022*
Місце складання	*м. Українськ*

НАКАЗ № *7 б*
про надання відпустки

Табельний номер
88648

Сапон Анна Миколаївна
(прізвище, ім'я, по батькові)

адміністрація
(назва структурного підрозділу)

бухгалтер
(назва професії; посади)

відпустка у зв'язку з вагітністю та пологами
вид відпустки (щорічна основна, додаткова, навчальна, без збереження заробітної плати та ін.)

Період відпустки з "18 » січня 2022 р. по "23» травня 2022 р.

на 126 календарних днів

Підстави: 1. Заява Анни Сапон від 18 січня 2022 р., зареєстрована за № 12.
2. Відкритий листок непрацездатності 123456-1000333333-1, який 18 січня 2021 р. сформований лікарем КНП "Сімейна поліклініка" ЧМР

Директор _____ Володимир КАЛЬЧЕНКО

Візи:
Начальник відділу управління персоналом:

_____ Костянтин ВАСИЛЬЧЕНКО
(підпис)
_____
(дата)

З наказом ознайомлена:

_____ Анна САПОН
(підпис)
_____
(дата)

Страница 1 из 1 Число слов: 140 украинский 80%

Рисунок 2.6 – Вихідний документ

2.4 Мова програмування та середовище розробки

Мовою програмування було обрано C++, для створення функціоналу додатку, та XML, для створення інтерфейсу.

Середовищем для розробки було обрано Qt [12]. Ця IDE має 3 типи ліцензії – Application Development, Device Creation (платні) та Community (безкоштовну).

У пакеті встановлення Qt вже заздалегідь включені мови програмування C++ та XML, модуль для роботи з базою даних SQLite3, менеджер пакетів для завантаження бібліотек та додатково завантажена бібліотека ActiveQt.

Інтерфейс програми представлений на рисунках 2.7-2.8.

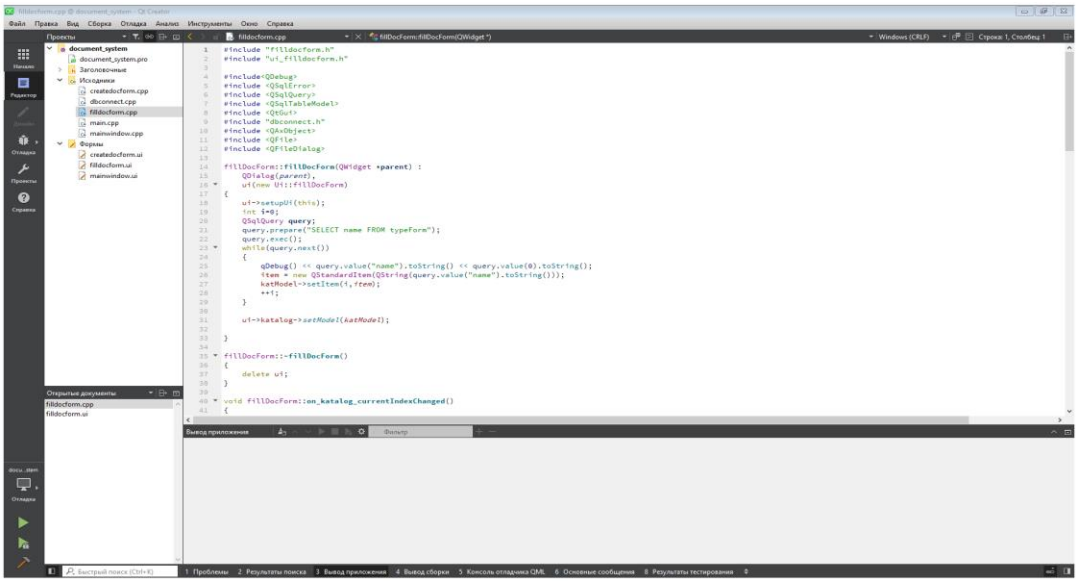


Рисунок 2.7 – Загальний інтерфейс Qt

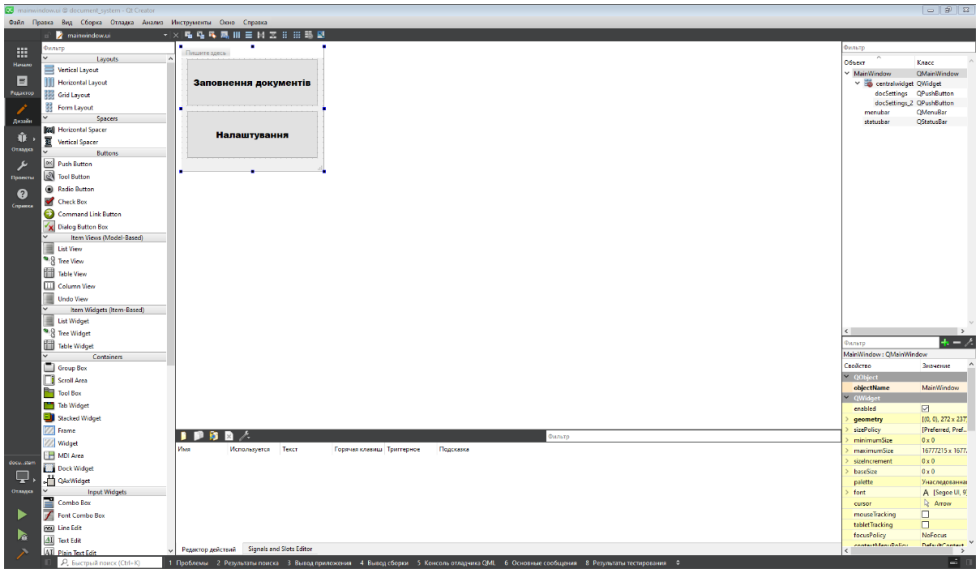


Рисунок 2.8 – Вбудований редактор дизайну интерфейсу в Qt

Qt має ряд переваг перед конкурентами:

– швидка багаторівнева розробка;

– кросплатформність;

– використання на мобільних платформах (Android, IOs, Symbian, Harmattan, Maemo);

– наявність слотів и сигналів;

– зручна міжпроцесна взаємодія;

– переносимість на рівні вихідного коду (Microsoft Windows, Linux, Mac OS X, QNX, Android, IOS);

– гарна документація.

2.5 Мова UML у проєктуванні програмних продуктів

При проєктуванні програмних продуктів застосовується мови UML. UML – стандартний інструмент для візуалізації, специфікації, конструювання та документування компонентів програмних систем [13].

Переваги мови UML:

– об'єктно-орієнтована мова;

– UML дозволяє описати систему з будь-якої точки зору;

– дозволяє описати різні аспекти системи;

– стрімко розвивається та широко застосовується;

– мову, можна використовувати для опису процесів в багатьох галузях життя.

Для створення моделі системи за допомогою мови UML використовується два основних поняття: сутності та відношення.

Існує 4 типи сутностей:

а) структурні:

1) клас;
2) інтерфейс;
3) компонент;

4) варіант використання;

5) кооперація;

6) вузол;

б) поведінкові:

1) взаємодія;

2) стан;

в) сутності, які групують;

г) анотації.

Кожна сутність має власне графічне представлення.

Відношення демонструють різні зв’язки між сутностями. У UML представлені такі типі відносин:

– залежності;

– асоціація;

– узагальнення;

– реалізація.

Для побудови UML-діаграм існує багато інструментальних засобів. Найбільш відомими серед них є: Draw.io, Microsoft Visio, Gliffy, Rational Rose.

Для створення UML-діаграм та блок-схем, які розроблені при проєктуванні додатку, було обрано сервіс Draw.io [14]. Цей інструмент дозволяє створювати різні типи UML-діаграм, блок-схеми алгоритмів та, навіть, бізнес-процесів.

Draw.io володіє такими перевагами:

– безкоштовне розповсюдження;

– робота у web-версії або у додатку на комп’ютері чи смартфоні;

– можливість редагувати стиль елементів;

– вибір готових діаграм;

– широкий набір елементів;

– кросплатформеність;

– доступні популярні формати JPG, PNG, SVG для зберігання діаграм;

– можливість взаємодіяти з сервісами Google Drive, OneDrive, Dropbox, GitHub.

На рисунку 2.9 можна ознайомитися з наборами шаблонів діаграм.

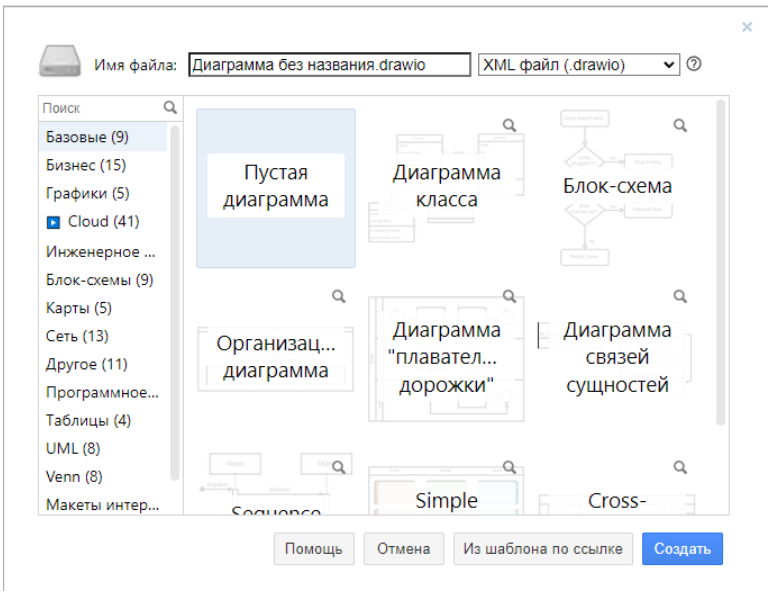


Рисунок 2.9 – Набір шаблонів діаграм у Draw.io

На рисунку 2.10 зображено загальний інтерфейс програми.

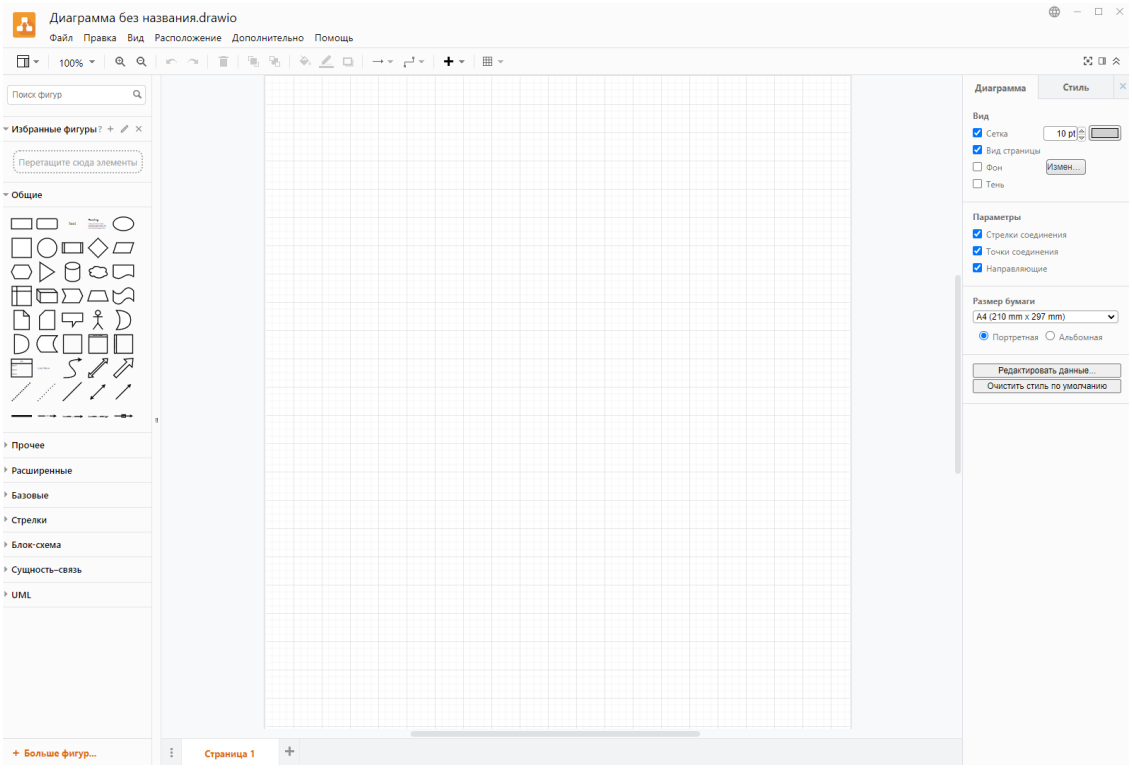


Рисунок 2.10 – Інтерфейс Draw.io

2.6 Висновки за розділом

У другому розділі була представлена архітектура додатку, описані функції. Було розглянуто методи вирішення поставлених задач. Бібліотеки для використання технологій COM та OLE обрано Active Qt, робота з базою даних реалізовуватиметься за допомогою вбудованого модуля SQLite3.

Для написання програмного коду було обрано мову C++ та XML та безкоштовну версію середовища розробки Qt.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ

3.1 Початок роботи з Qt Creator

За допомогу вбудованого майстра створення проєктів та файлів, створимо новий проєкт. Так як, додаток повинен працювати на персональних комп'ютерах, виберемо варіант створення «Додаток Qt Widgets», який вже містить каркас головного вікна, методи його відображення та мову програмування C++, як основну. Далі необхідно вказати назву проєкту, в нашому випадку – document_system, та шлях, де буде знаходитися проект з додатком. Для проєкту використаємо систему зборки qmake і компілятор MinGW. Всі інші пункти залишаємо за замовчуванням.

Покрокове створення представлено на рисунках 3.1–3.5.

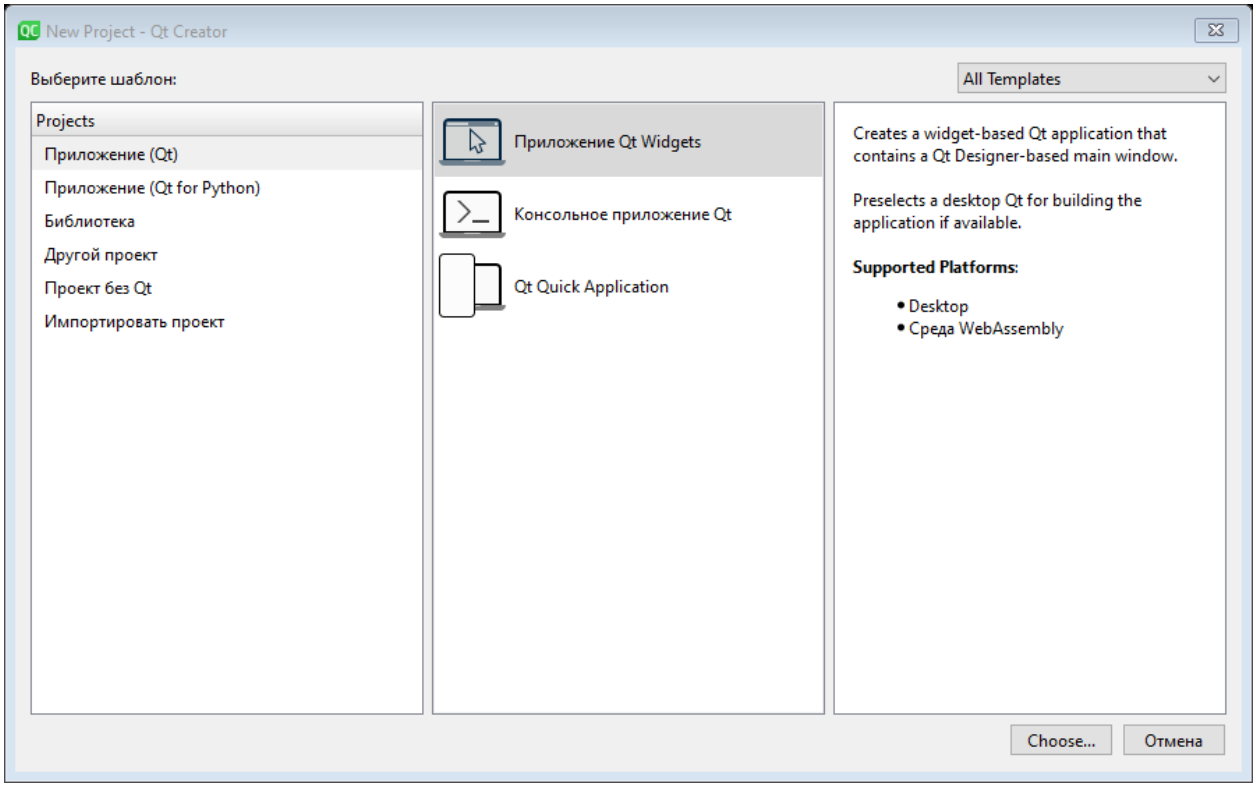


Рисунок 3.1 – Процес створення проекту, крок 1

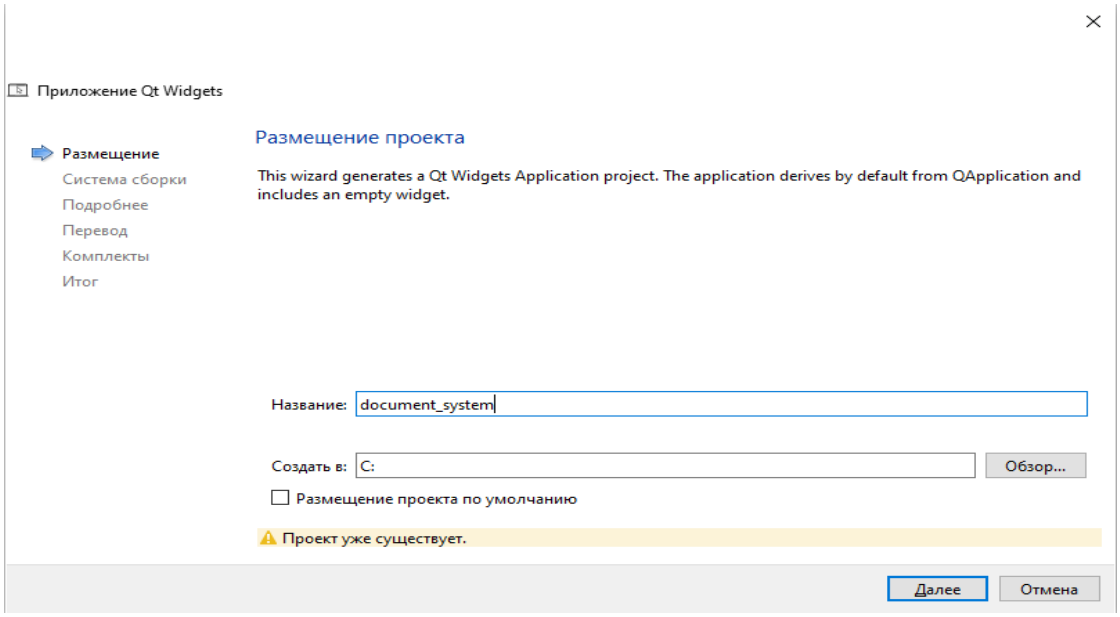


Рисунок 3.2 – Процес створення проекту, крок 2

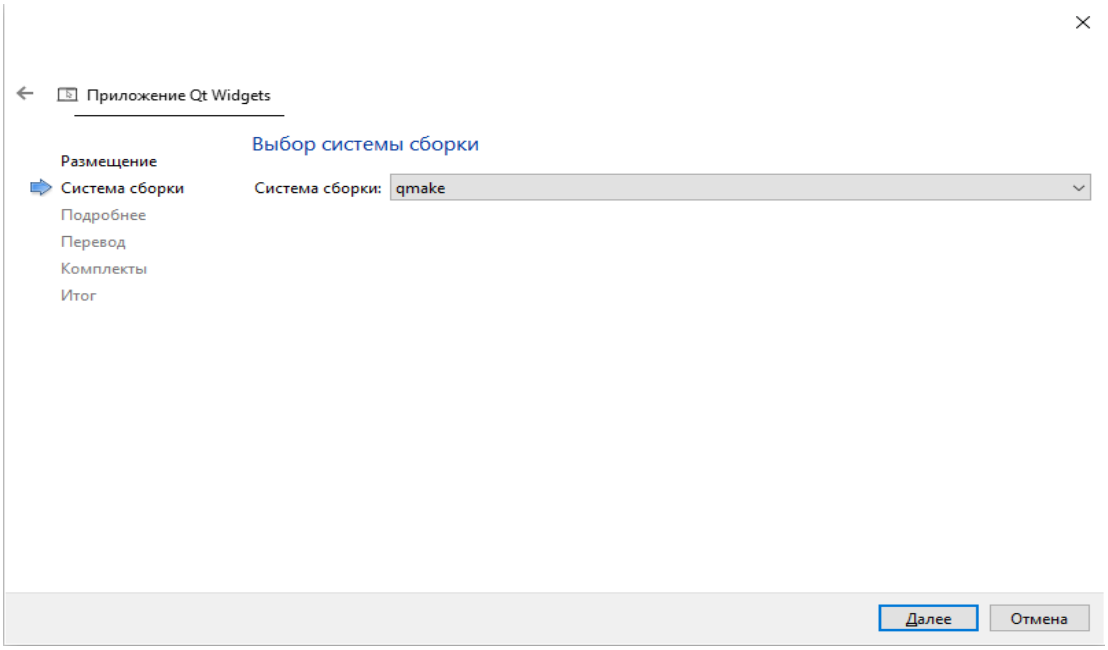


Рисунок 3.3 – Процес створення проекту, крок 3

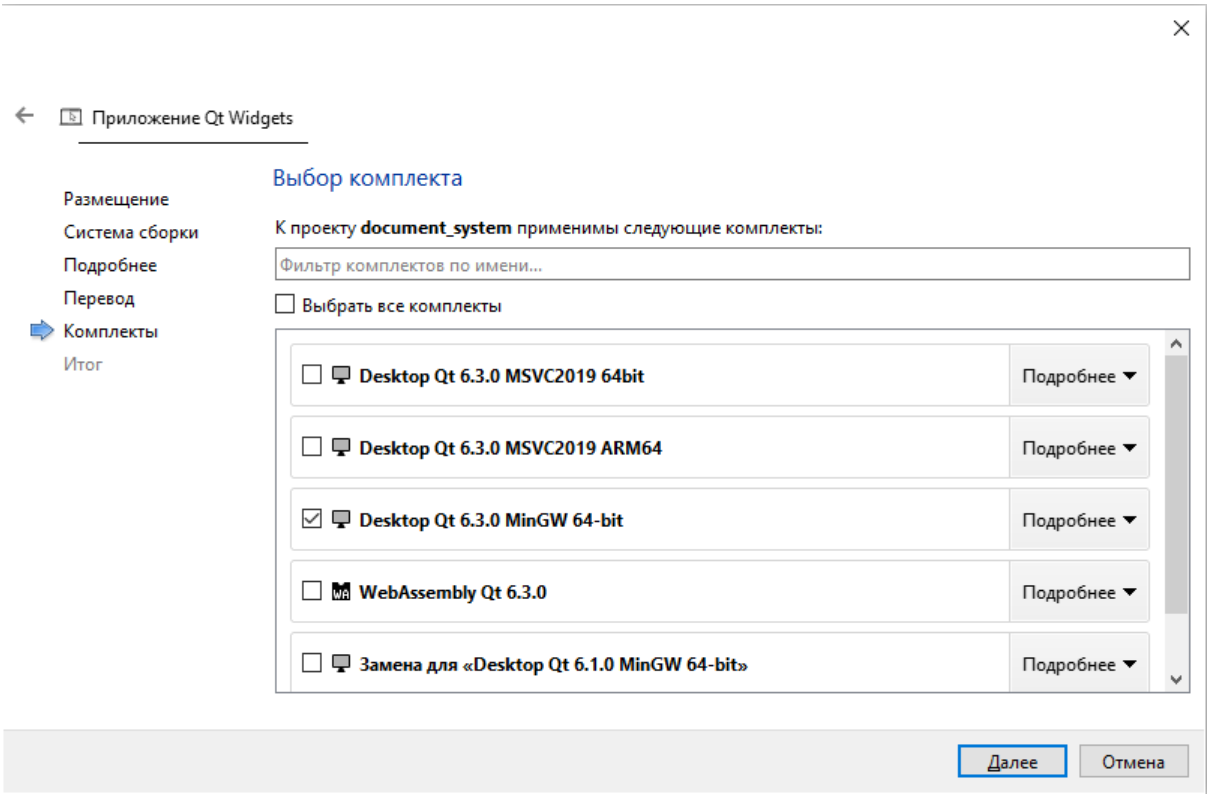


Рисунок 3.4 – Процес створення проекту, крок 4

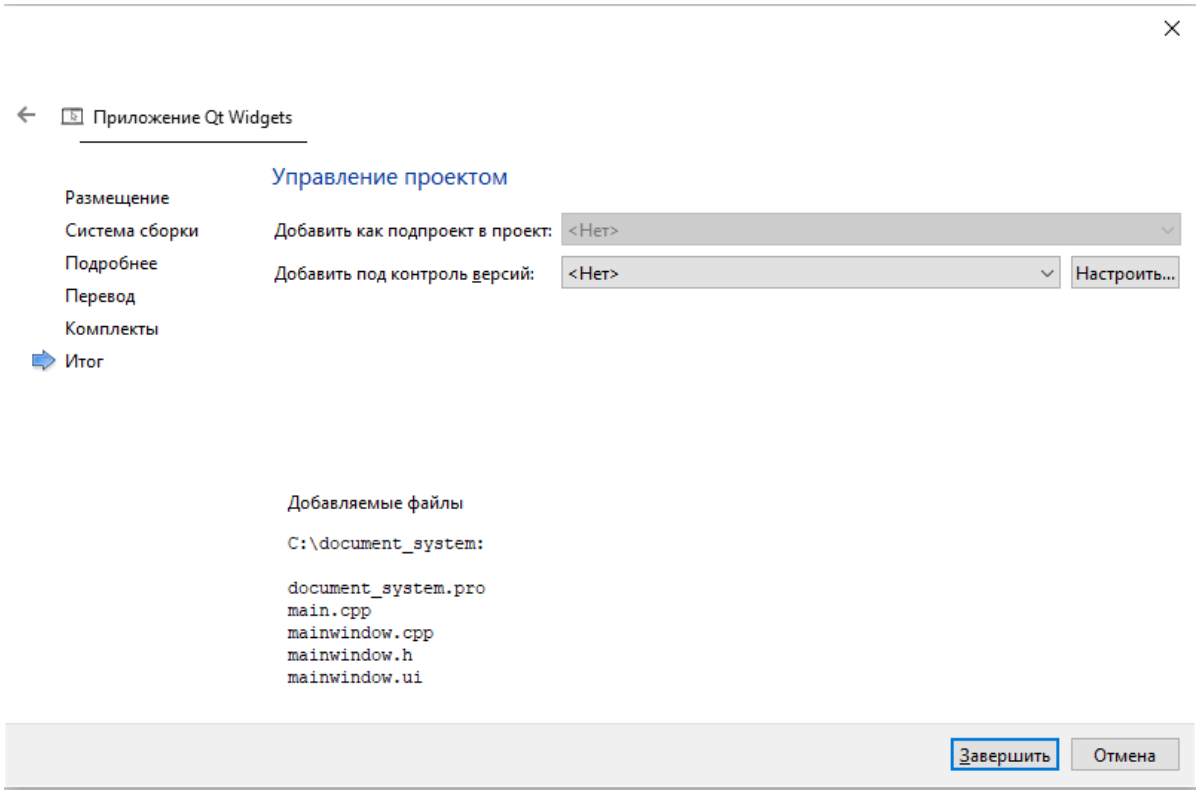


Рисунок 3.5 – Процес створення проекту, крок 5

3.2 Модулі проекту

Проєкт створений та розбитий по модулям, окремі модулі відповідають за окремі частини. На рисунку 3.6 відображено усі модулі, які існують в програмному додатку.

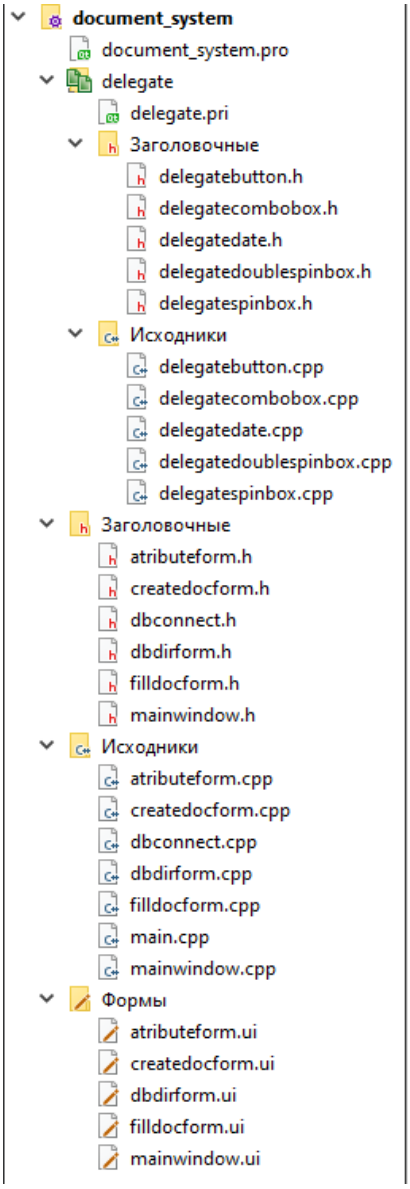


Рисунок 3.6 – модулі програмного додатку

3.2.1 Модуль document_system.pro

Модуль document_system.pro є головним модулем проєкту. В ньому містяться вся необхідна інформація для qmake, щоб побудувати додаток, а саме:

– Qt модулі, які використовуються в проєкті;
– загальні параметри конфігурації;
– список включених файлів (.pri);
– список файлів сирцевого коду (.cpp);
– список файлів заголовків (.h);
– список файлів інтерфейсу користувача (.ui).

Лістинг представлено у додатку В.1.

3.2.2 Модуль main.cpp

Модуль main.cpp є головним модулем додатку. В ньому відбувається створення об’єкту класу QApplication з однойменної бібліотеки Qt, який оброблює та координує роботу вікон додатку, основного циклу подій. Також, він викликає функцію connect() модуля dbconnect для підключення до бази даних та відкриває головного вікна додатку через створення об’єкту класу MainWindow однойменного модуля та виклику функції show(), яка унаслідується від стандартного класу Qt QMainWindow.

Лістинг представлено у додатку В.2.

3.2.3 Модуль dbconnect

Модуль розроблено для створення та взаємодії з базою даних. Він складається з однойменних файлів заголовку та сирцевого коду форматів .h і .cpp відповідно. Для роботи з базою даних необхідно імпортувати модуль QSqlQuery, в якого входить модуль роботи з базами даних SQLite та є вбудованим інструментом середовища Qt Creator.

База даних містить сім таблиць: atribute, form, poles, propAtribute, properties, typeAtribute та typeForm.

Таблиця atribute містить такі поля:

– id з типом даних integer, є первинним ключем;
– name з типом даних varchar;

– type з типом даних integer.

Таблиця form складається з таких полів:

– id з типом даних integer, є первинним ключем;

– nameForm з типом даних varchar;

– typeForm з типом даних integer;

– dir з типом даних varchar.

Таблиця poles містить такі поля:

– id з типом даних integer, є первинним ключем;

– name з типом даних varchar;

– idForm з типом даних integer;

– attribute з типом даних integer;

– properties з типом даних varchar;

– flag з типом даних bool.

Таблиця propatribute містить такі поля:

– id з типом даних integer, є первинним ключем;

– idAtribute з типом даних integer;

– idProperties з типом даних integer;

– value з типом даних varchar;

– caseIndx з типом даних integer.

Таблиця properties містить такі поля:

– id з типом даних integer, є первинним ключем;

– name з типом даних varchar;

– value з типом даних varchar;

– type з типом даних integer;

– caseIndx з типом даних integer.

Таблиця typeAtribute містить такі поля:

– id з типом даних integer, є первинним ключем;

– name з типом даних varchar;

– type з типом даних integer.

Таблиця typeForm містить такі поля:

– id з типом даних integer, є первинним ключем;

– name з типом даних varchar;

На рисунку 3.7 відображена структура таблиць бази даних, за допомогою DB Browser.

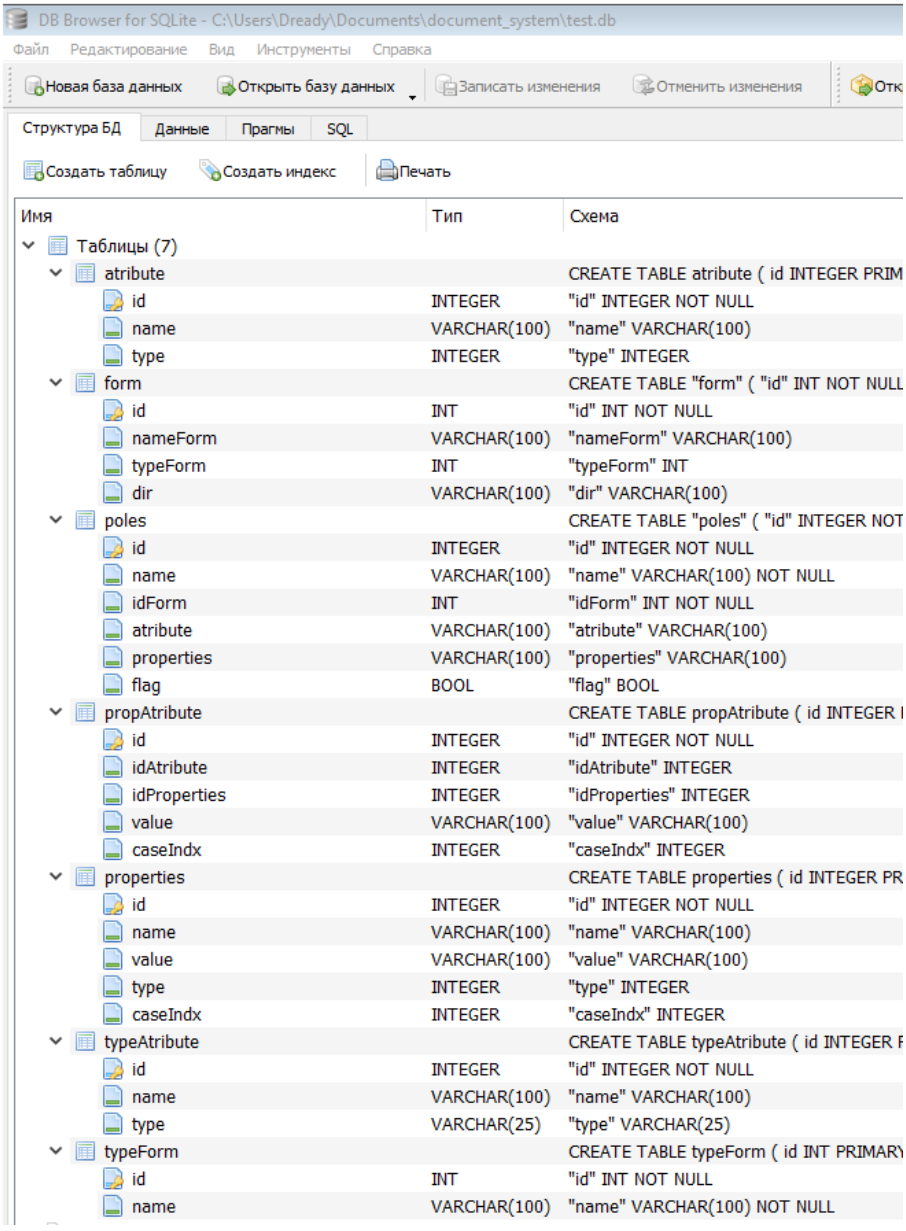


Рисунок 3.7 – Таблиці бази даних

Модуль dbconnect окрім конструктора та деструктора, має функцію connect(), яка, в разі відсутності бази даних, створює її, підключає додаток до бази даних, робить можливим доступ до неї на всіх етапах роботи додатку.

Лістинг представлено у додатку В.4.

3.2.4 Модуль mainwindow

Модуль розроблено для відображення головного меню додатку. Він складається з однойменних файлів заголовку, сирцевого коду та інтерфейсу користувача форматів .h, .cpp і .ui відповідно. Клас MainWindow є спадкоємцем публічного базового класу Qt QMainWindow з відповідного модуля. Також підключені інші модулі вікон, а саме – atributeform, createdocform, filldocform, dbdirform.

Модуль містить одну функцію – FuncPass(), яка створює вікно з полем для введення паролю, перевіряє на правильність паролю та повертає результат перевірки.

У вікні розташовано 2 кнопки, та ще 6 знаходяться у верхньому меню:

– «Заповнення документів» – знаходиться безпосередньо у вікні, при натисканні, подає сигнал у слот on_start_clicked(), в якому відбувається відкриття вікна модуля filldocform;

– «Вихід» – знаходиться безпосередньо у вікні, при натисканні, подає сигнал у слот on_exit_clicked(), в якому відбувається закриття головного вікна, що призводить до виходу з додатку;

– «Користувач»–«Адміністратор» – знаходиться у верхньому меню, при натисканні, подає сигнал у слот on_loginAdmin_triggered(), в якому відбувається звернення до функції FuncPass () та при позитивному результаті, програма входить у профіль адміністратора, змінюючи назву меню з «Користувач» на «Адміністратор» та розблоковуючи меню «Змінити пароль» і «Налаштування»;

– «Адміністратор»–«Користувач» – знаходиться у верхньому меню, при натисканні, подає сигнал у слот on_loginUser_triggered(), в якому відбувається вихід з профілю адміністратора, змінюючи назву меню з «Адміністратор» на «Користувач» та заблоковуючи меню «Змінити пароль» і «Налаштування»;

– «Адміністратор»–«Змінити пароль» – знаходиться у верхньому меню, при натисканні, подає сигнал у слот on_changePass_triggered(), в якому відбувається звернення до функції FuncPass () та при позитивному результаті надається можливість змінити пароль;

– «Налаштування»–«Документи» – знаходиться у верхньому меню, при натисканні, подає сигнал у слот on_documentAction_triggered(), в якому відбувається відкриття вікна модуля createdocform;

– «Налаштування»–«Типи даних» – знаходиться у верхньому меню, при натисканні, подає сигнал у слот on_typesAction_triggered(), в якому відбувається відкриття вікна модуля atributeform.

– «Налаштування»–«Бази даних» – знаходиться у верхньому меню, при натисканні, подає сигнал у слот on_dbAction_triggered(), в якому відбувається відкриття вікна модуля dbdirform.

Лістинг представлено у додатку В.5.

3.2.5 Модуль atributeform

Модуль розроблено для відображення вікна налаштувань типів даних. Він складається з однойменних файлів заголовку, сирцевого коду та інтерфейсу користувача форматів .h, .cpp і .ui відповідно. Клас atributeForm є спадкоємцем публічного класу Qt QDialog з відповідного модуля. Також підключені інші модулі з списку delegate.pri, а саме – delegatecombobox, delegatespinbox.

Цей модуль потрібен для створення типів даних, для подальшого більш легкого та інтуїтивно зрозумілого заповнення документів. Типи даних вказуються під час створення шаблону документа.

У вікні розташовано список імен типів даних та таблиця з властивостями, які є зв’язаними: при виборі типу даних з списку, в таблиці, відображаються його властивості. Також, на формі присутні 4 кнопки:

– «Додати» – подає сигнал у слот on_pushButton_add_clicked(), в якому відбувається створення нового типу даних;

– «Видалити» – подає сигнал у слот on_pushButton_delete_clicked(), в якому відбувається видалення вибраного типу даних в списку;

– «Зберегти» – при натисканні, подає сигнал у слот on_buttonBox_clicked(QAbstractButton *button), в якому відбувається збереження властивостей типу даних;

– «Закрити» – при натисканні, подає сигнал у слот on_buttonBox_clicked (QAbstractButton *button), в якому відбувається закриття вікна.

Лістинг представлено у додатку В.6.

3.2.6 Модуль createdocform

Модуль розроблено для відображення вікна налаштувань шаблонів документів. Він складається з однойменних файлів заголовку, сирцевого коду та інтерфейсу користувача форматів .h, .cpp і .ui відповідно. Клас CreateDocForm є спадкоємцем публічного класу Qt QDialog з відповідного модуля. Також підключен модуль з списку delegate.pri – delegatecombobox.

Більша частина вікна розбита на 4 виділені області:

а) «Категорія документів» – містить об’єкти, котрі стосуються категорій документів:

1) випадаючий список категорій документів;

2) «Новий» – кнопка додавання нової категорії документів;

3) «Редагувати» – кнопка видалення вибраної категорії документів, при натисканні;

4) «Видалити» – кнопка видалення вибраної категорії документів;

б) «Документ» – містить об’єкти, котрі стосуються шаблонів документів, та їх список є ідентичним до списку області «Категорія документів»;

в) «Унікальні параметри» – містить об’єкти, котрі стосуються унікальних параметрів шаблону документа:

1) таблиця параметрів;

2) «Додати» – кнопка додавання поля в таблицю параметрів;

3) «Видалити» – кнопка видалення поля в таблиці параметрів;

г) «Повторювальні параметри» – містить об’єкти, котрі стосуються повторювальних параметрів шаблонів документів, та їх список є ідентичним до списку області «Унікальні параметри».

Також, присутні ще 2 кнопки:

– «Зберегти» – при натисканні, подає сигнал у слот on_buttonBox_clicked(QAbstractButton *button), в якому відбувається збереження параметрів шаблону документа;

– «Закрити» – при натисканні, подає сигнал у слот on_buttonBox_clicked (QAbstractButton *button), в якому відбувається закриття вікна.

Лістинг представлено у додатку В.6.

3.2.7 Модуль filldocform

Модуль розроблено для відображення вікна заповнювання шаблонів документів. Він складається з однойменних файлів заголовку, сирцевого коду та інтерфейсу користувача форматів .h, .cpp і .ui відповідно. Клас fillDocForm є спадкоємцем публічного класу Qt QDialog з відповідного модуля. Також підключені всі модулі з списку delegate.pri, а саме – delegatebutton, delegatecombobox, delegatedate, delegatedoublespinbox, delegatespinbox.

У вікні розташовано два випадаючих списки: категорії документів та їх шаблонів. Також, дві таблиці з унікальними та повторювальними параметрами, та 2 кнопки: «Заповнити» – для заповнення шаблону, «Закрити» – для закриття вікна.

У цьому модулі були використані COM та OLE технології, тому була підключена бібліотека QAxObject модуля ActiveQt, для взаємодії з COM та OLE об’єктами, та QFile, для створення копій файлів.

Для копіювання шаблону, використовуються функція QFile::copy().

Для подальшого заповнення шаблону, вже використовується функції бібліотеки QAxObject:

– new QAxObject("Word.Application") – конструктор, для створення COM об’єкту додатку Word;

– querySubObject("Documents") – створює каталог документів;

– querySubObject("Open(QString)",documentDirNew) – додає до каталогу документ, який розташовується за шляхом documentDirNew;

– querySubObject("ActiveDocument()") – змінює стан відкритого документа на активний;

– querySubObject("Content()") – отримує доступ до змісту активного документа;

– querySubObject("Find") – отримує доступ до об’єкту пошуку в середині змісту документа.

Сам об’єкт Find при виконані може приймати до 20 змінних. Так як, для виконання необхідно вводити змінні за послідовністю їх розташування, тоді використання 15 з них стає необхідним, а саме:

– QString FindText – шуканий текст;

– bool MatchCase – налаштовує чутливість до регістру;

– bool MatchWholeWord – шукати як ціле слово;

– bool MatchWildcards – використання підставних знаків;

– bool MatchSoundsLike – дозволити знаходження схожих слів;

– bool MatchAllWordForms – дозволити знаходити всі форми слова;

– Forward – пошук до кінця документа;

– Wrap – визначає поведінку перенесення, якщо до пошуку був вказано діапазон і операція була провальною;

– Format – пошук за форматуванням;

– ReplaceWith – текст заміни;

– Replace – скільки зробити змін: одну, всі, жодну.

Для алгоритму заповнення шаблонів документів було побудовано блок-схему. На рисунку нижче відображено блок-схему алгоритму заповнення шаблонів документів (рис. 3.8).

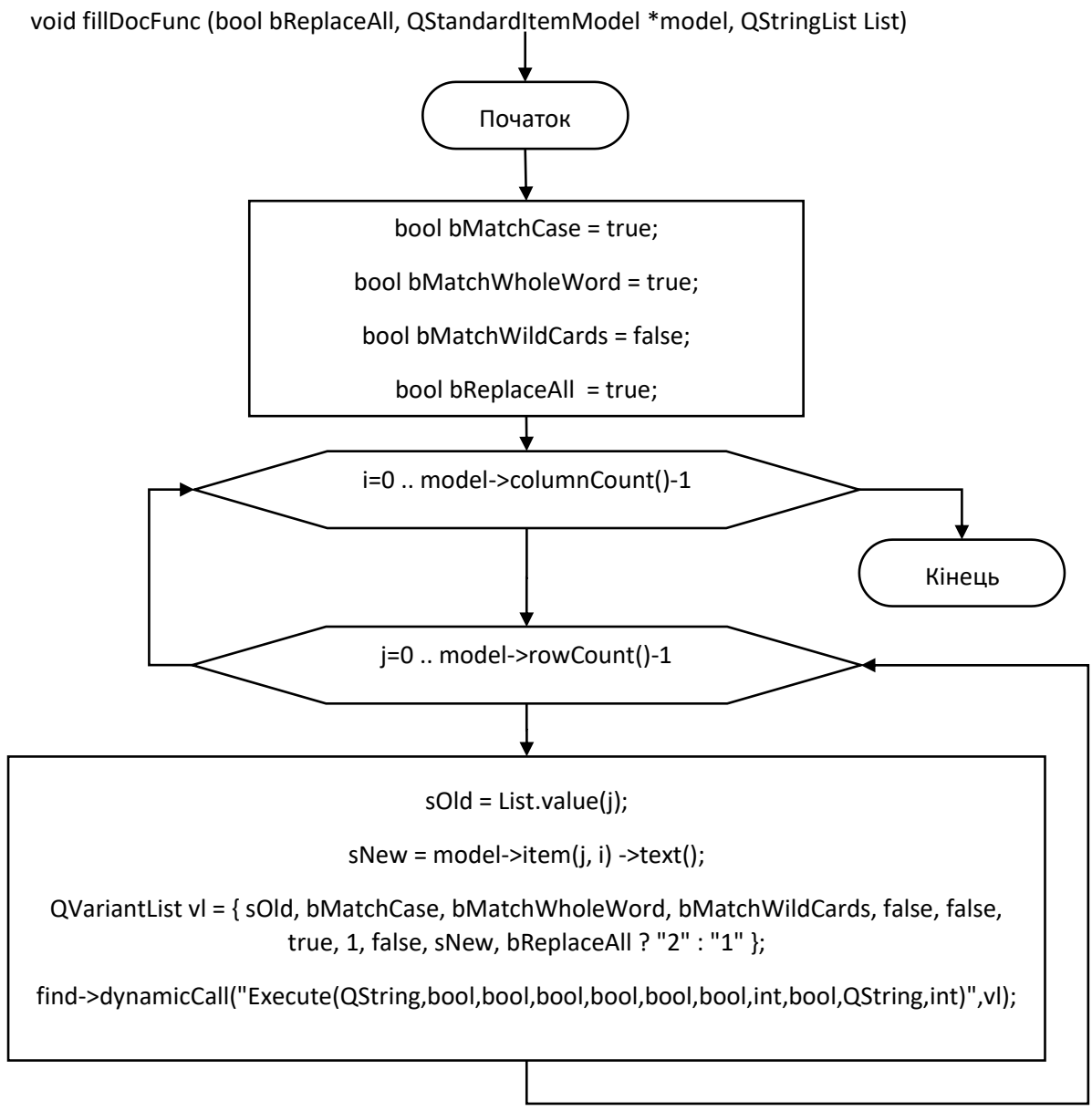


Рисунок 3.8 – Алгоритм заповнення шаблонів документів

Лістинг представлено у додатку В.7.

3.2.8 Список включених файлів delegate

Список слугує для додавання модулів делегатів. Делегати слугують для можливості введення та відмолювання індивідуальних елементів всередині уявлень. У додатку, модулі делегатів використовуються для відображення різних елементів керування, буд то кнопка або випадаючий список, всередині уявлень таблиць, які за замовчуванням відображають тільки поля для вводу. Це робить процес заповнення таблиць більш зрозумілішим та зручним.

Кожен модуль списку складається з однойменних файлів заголовку та сирцевого коду форматів .h і .cpp відповідно.

Кожен модуль має конструктор, деструктор, функцію створення віджета, функцію встановлення інформації у віджет, функцію встановлення інформації у модель, функцію зміни геометрії віджета.

Список модулів делегатів:

– delegatebutton – відображає кнопку;

– delegatecombobox – відображає випадаючий список;

– delegatedate – відображає віджет дати;

– delegatedoublespinbox – відображає віджет для чисел з плаваючою крапкою;

– delegatespinbox – відображає віджет для цілих чисел.

Лістинг представлено у додатку В.8.

3.3 Висновки за розділом

У цьому розділі була описана розробка програмного засобу: був створений проєкт, його код та розбитий на окремі модулі. Також була побудована блок-схема алгоритму парсингу даних.

4 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ І ЗАСОБИ БЕЗПЕКИ

4.1 Опис реалізації системи

Для того, щоб почати роботу з додатком, необхідно запустити виконавчий файл у корні папки з додатком, який називається – «document_system.exe». Цей файл запустить додаток, та буде відображене головне вікно. На рисунку 4.1 відображено початок робот з додатком.

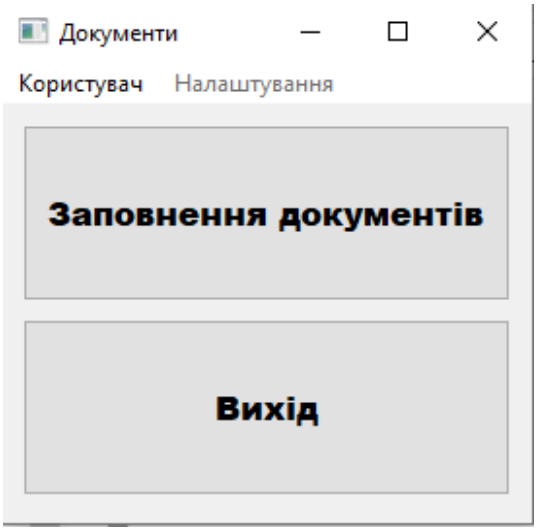


Рисунок 4.1 – Початок робот з додатком «Документи»

Якщо користувач натисне на кнопку «Вихід», відбудеться вихід з додатку, а при натисненні кнопки «Заповнення документів», відкриється вікно «Заповнення документів». На рисунку нижче відображене вікно (рис. 4.2).

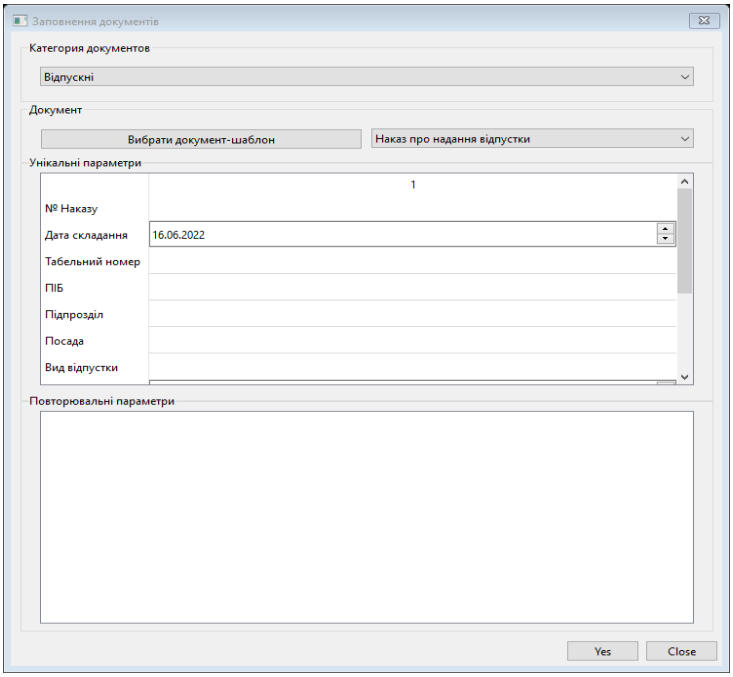


Рисунок 4.2 – Випадок, коли користувач натиснув кнопку «Заповнення документів»

Перш за все, необхідно вибрати каталог документів, далі – вибрати документ з каталогу. Після цього, якщо ваш шаблон документу відрізняється від стандартного, тоді необхідно натиснути на кнопку «Вибрати документ-шаблон». Після цього відкриється вікно вибору файлу шаблону, яке зображене на рисунку 4.3.

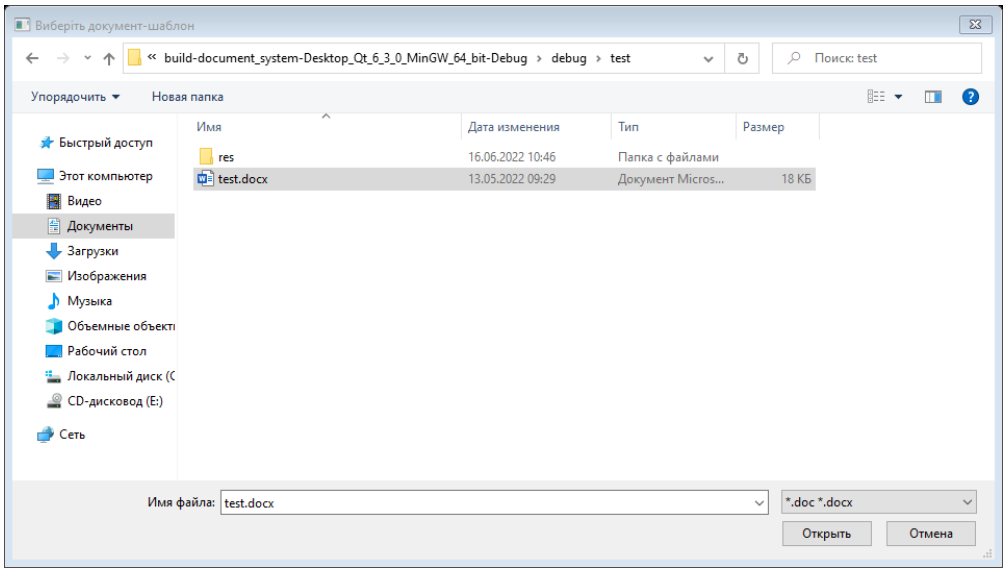


Рисунок 4.3 – Вікно вибору файлу

Після цього, необхідно заповнити поля таблиць «Унікальні параметри» та «Повторювальні параметри». На рисунку 4.4 відображено вікно з заповненими таблицями.

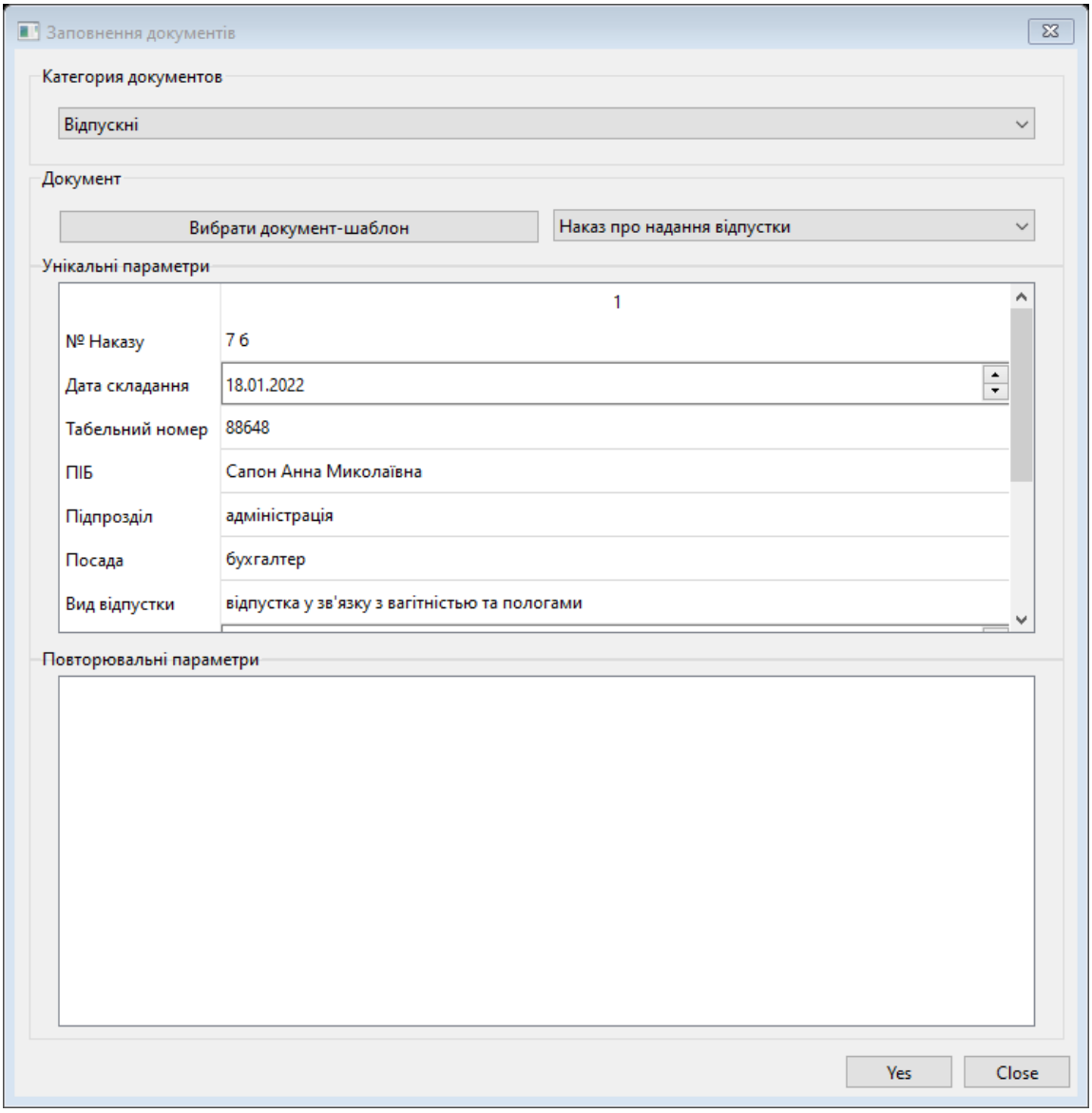


Рисунок 4.4 – Користувач заповнив таблиці

При натисканні кнопки «Yes», після деякого часу, додаток відкриє оброблений шаблон, а користувач зможе закрити вікно на кнопку «Close», та перейти у головне вікно. Результат зображено на рисунку 4.5.

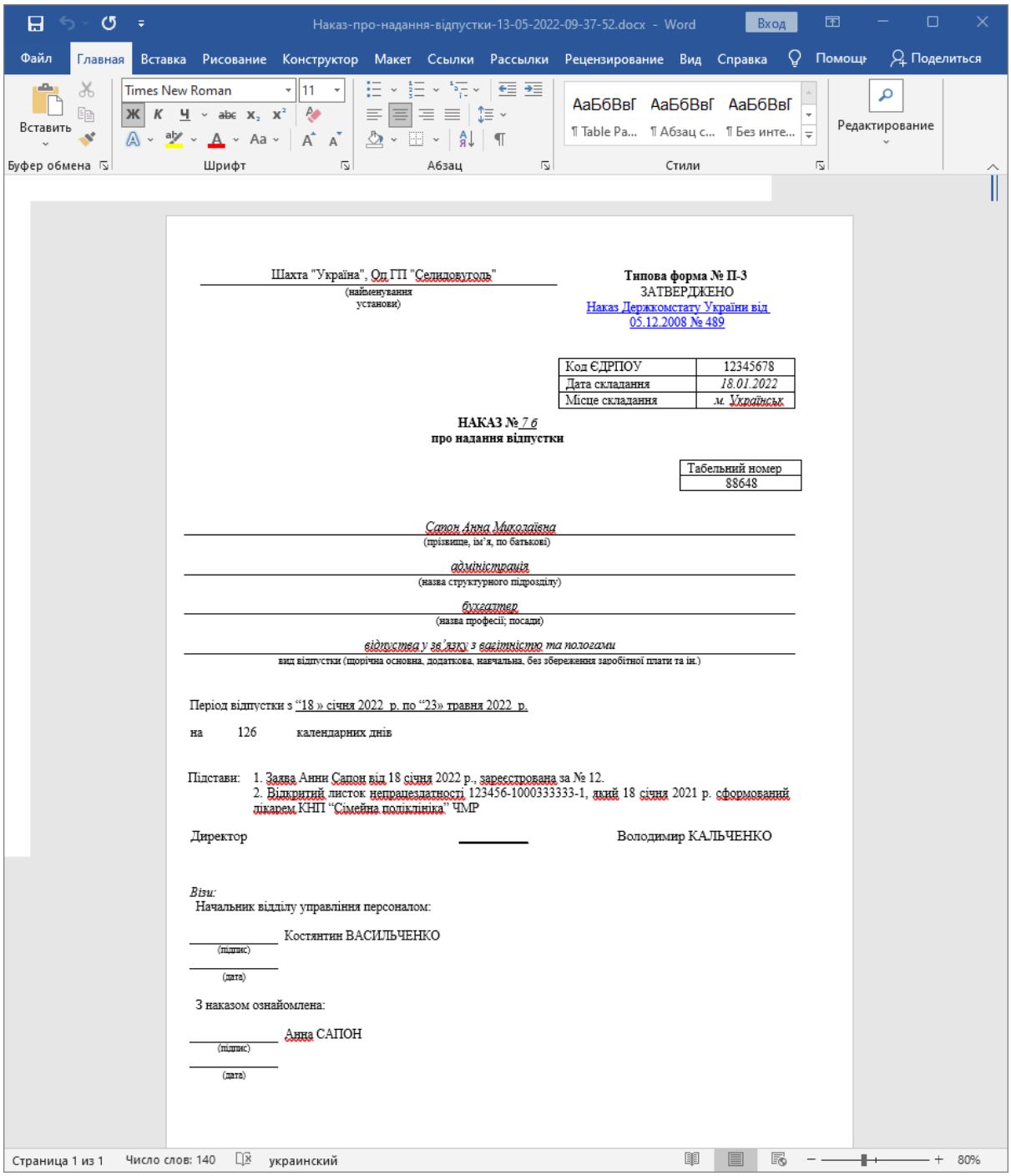


Рисунок 4.5 – Результат роботи

Якщо, користувач володіє паролем адміністратора, тоді він має можливість увійти в додаток як адміністратор, після введення паролю. Процес входу адміністратора в додаток, зображено на рисунках 4.6 та 4.7.

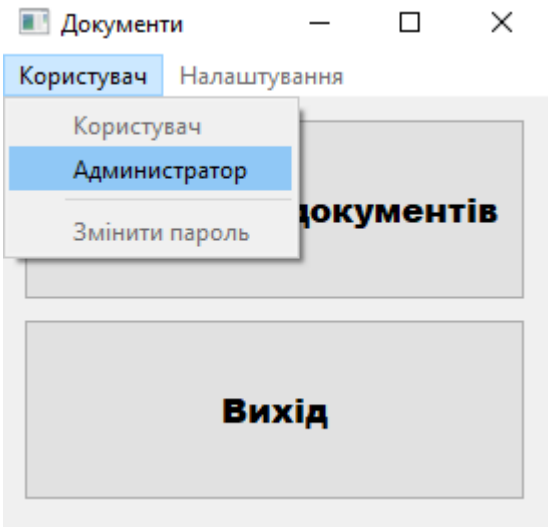


Рисунок 4.6 – Меню «Користувач»

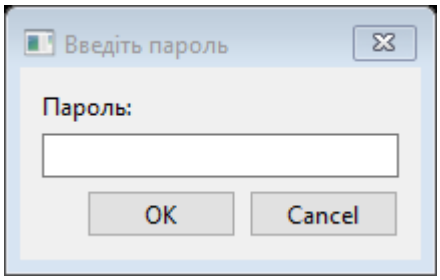


Рисунок 4.7 – Випадок, коли користувач вибрав пункт «Адміністратор»

Після того, як адміністратор увійшов у систему завдяки паролю, він може перейти до профілю користувача за допомогу вибору пункту «Користувач» у меню «Адміністратор». Для цього, йому не знадобиться знову вводити пароль.

Крім цього, адміністратору стають доступними функції користувача та адміністратора водночас.

Одна з функцій адміністратора – це змінна паролю адміністратора. Для того, щоб це здійснити, необхідно зайти у меню «Адміністратор» та вибрати пункт «Змінити пароль». Після цього, буде відображене вікно «Введіть старий пароль» для введення теперішнього паролю. Якщо пароль буде введено правильно, тоді відобразиться вікно «Введіть новий пароль» для вводу нового паролю, інакше – відобразиться вікно з помилкою «Невірний пароль». На

рисунках 4.8-4.12 відображено меню «Адміністратор» та процес зміни паролю.

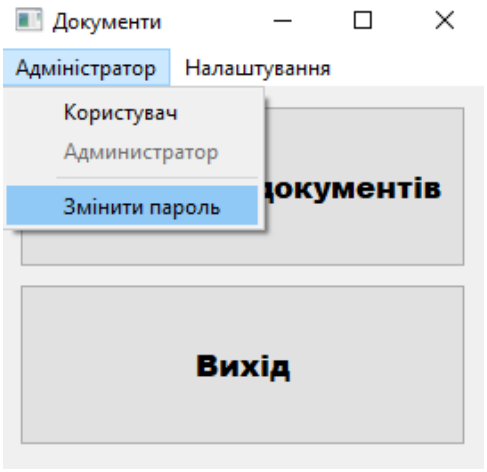


Рисунок 4.8 – Меню «Адміністратор»

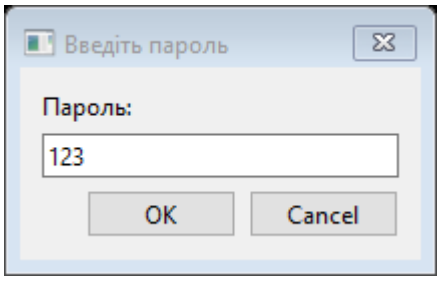


Рисунок 4.9 – Випадок, коли адміністратор вибрав пункт «Змінити пароль»

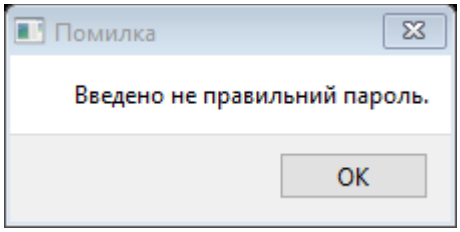


Рисунок 4.10 – Випадок, коли адміністратор ввів не правильний пароль

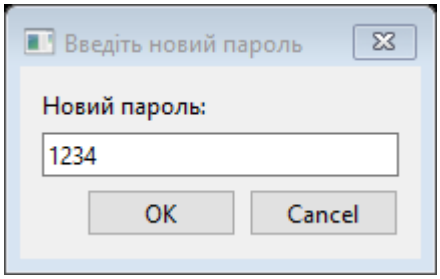


Рисунок 4.11 – Випадок, коли адміністратор ввів правильний пароль

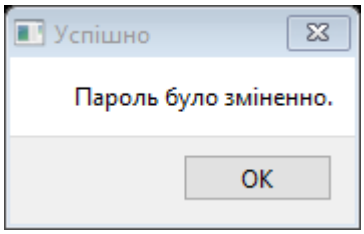


Рисунок 4.11 – Повідомлення, про успішну операцію зміни пароля

Також, адміністратору доступне меню «Налаштування» та його 2 пункти – «База даних системи додатку» та «База даних підприємства», котрі відкривають вікно «Підключення до бази даних». Вікно слугує для заповнення шляху до баз даних системи додатку або підприємства, відповідно до вибраного пункту.

У вікні треба вказати тип доступу до бази даних: «Локальний доступ» або «Доступ через Інтернет». Якщо було вибрано «Локальний доступ», тоді необхідно вказати розташування бази даних за допомогою кнопки «Вибрати файл», яка відкриє вікно для пошуку файлів, або, за допомогою кнопки «За замовчуванням», яке підставить шлях за замовчуванням, якщо вибирається шлях до бази даних системи додатку, або за допомогою поля для ручного вводу. У випадку, коли було вибрано «Доступ через Інтернет», тоді необхідно вказати у поле «Хост» – ім’я або адресу сервера, а у поле «Ім’я бази даних» – ім’я бази даних. Після цього, якщо необхідно, треба вказати ім’я користувача та пароль у полях «Ім’я користувача» та «Пароль» відповідно.

Як поля будуть заповнені, необхідно натиснути на кнопку «Ok» для підтвердження змін та «Cancel», щоб їх відмінити. Обидва варіанти закривають вікно, та повертають у вікно «Документи». На рисунках 4.12-14 продемонстровано меню «Налаштування» та вікно «Підключення до бази даних».

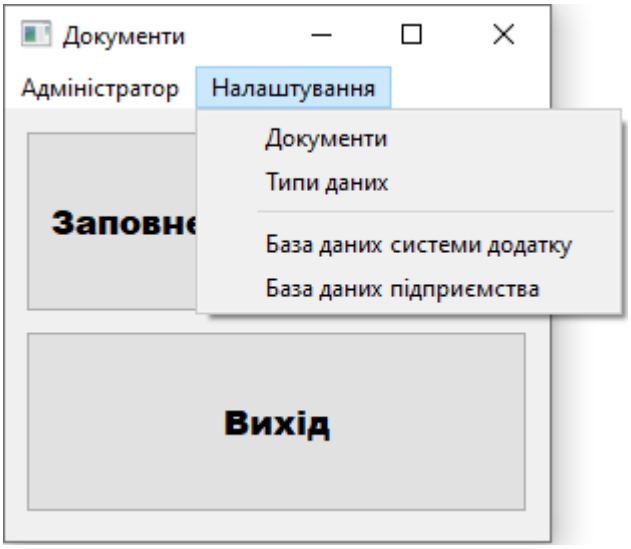


Рисунок 4.12 – Меню «Налаштування»

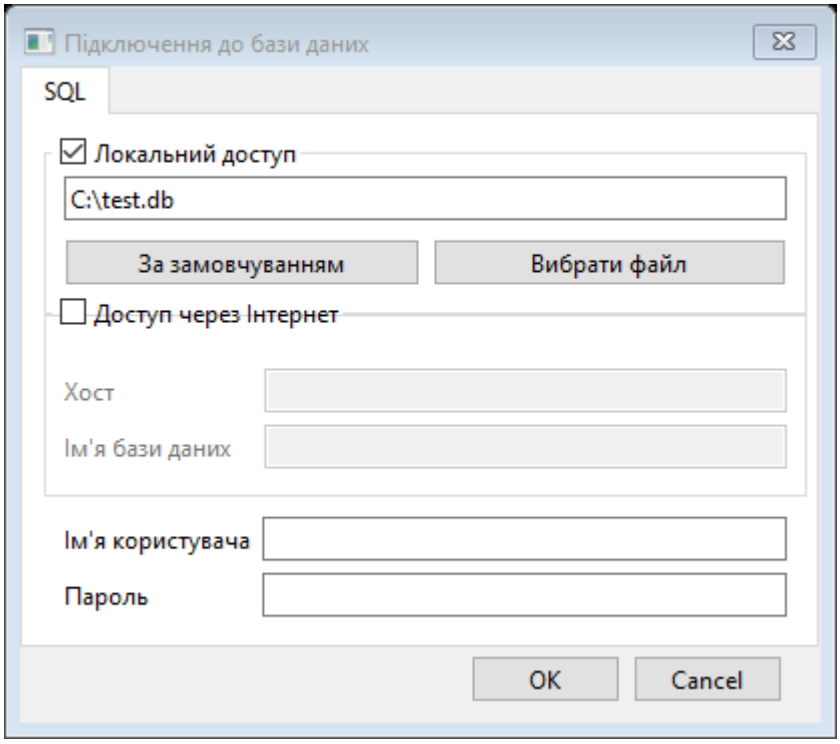


Рисунок 4.13 – Випадок, коли адміністратор вибрав пункт «База даних системи додатку» або «База даних підприємства»

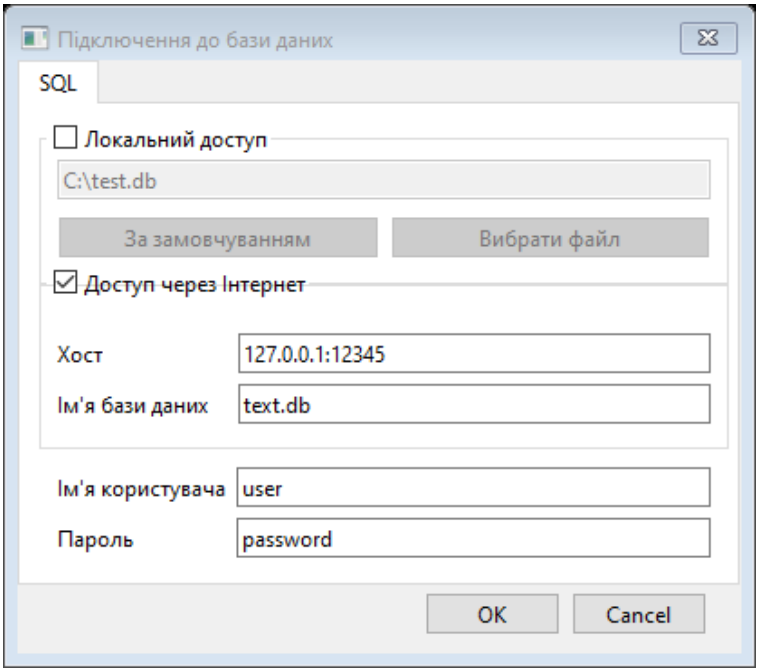


Рисунок 4.14 – Інформація, введена для доступу до бази даних через Інтернет

Окрім даних пунктів в меню «Налаштування», також є пункт «Типи даних», при натисканні на котрий відкриває вікно «Налаштування типів даних». На рисунку 4.15 зображене відкрите вікно.

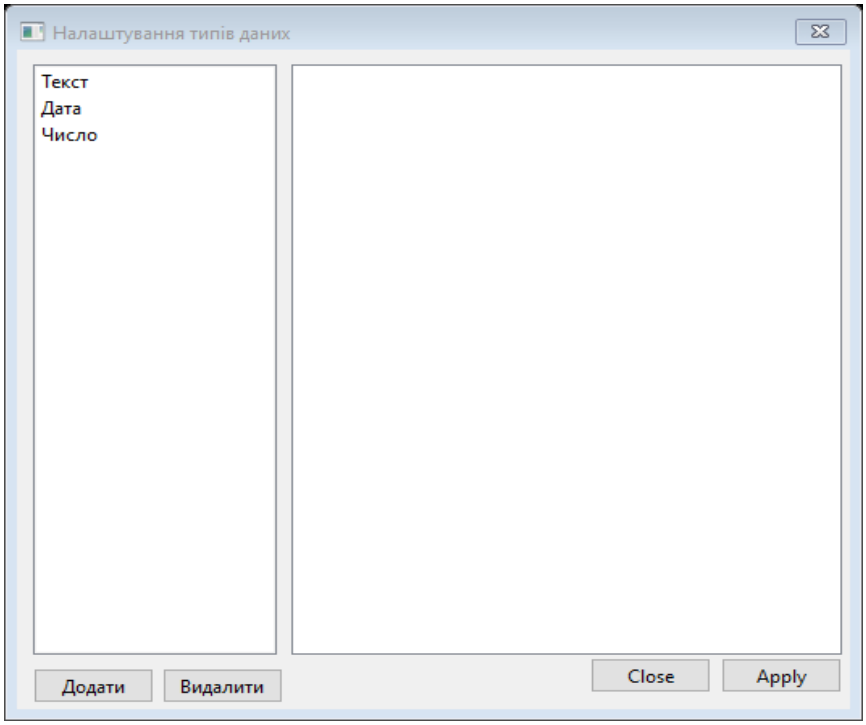


Рисунок 4.15 – Випадок, коли адміністратор вибрав пункт «Типи даних»

У вікні «Налаштування типів даних» зліва зображено список атрибутів. Для того, щоб змінити назву атрибута, необхідно двічі натиснути на ім’я атрибуту в списку, після чого, на місті атрибута, відкриється поле для редагування імені, а після виходу з нього – назва атрибута буде збережена.

Також, під списком розташовані дві кнопки: «Додати» – для додавання нового атрибуту, та «Видалити» – для видалення атрибута. На рисунках 4.16-4.17 відображено процес створення атрибута «Гривня».

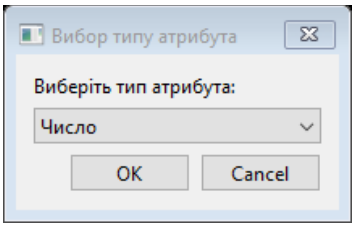


Рисунок 4.16 – Вікно вибору типу нового атрибута

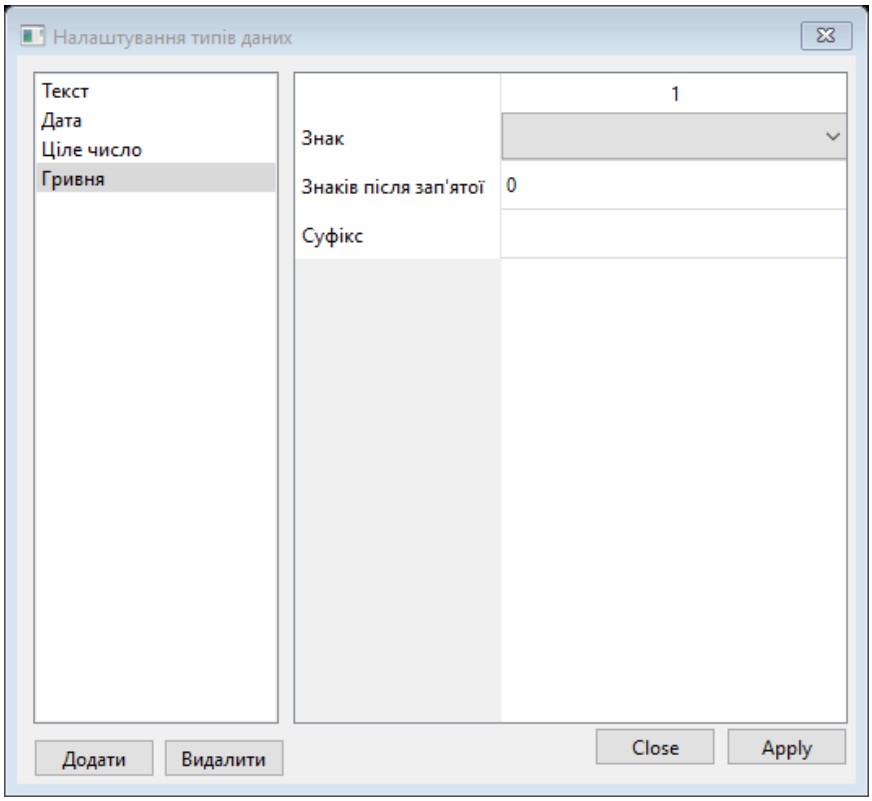


Рисунок 4.17 – Результат додавання нового атрибута

У правій частині вікна розташовується таблиця для заповнення властивостей атрибуту. Також, у вікні розташовані кнопки «Apply», для збереження введених властивостей атрибуту та «Close» для закриття вікна, та переходу до головного вікна. На рисунку 4.18 продемонстровано процес заповнення властивостей атрибуту «Гривня».

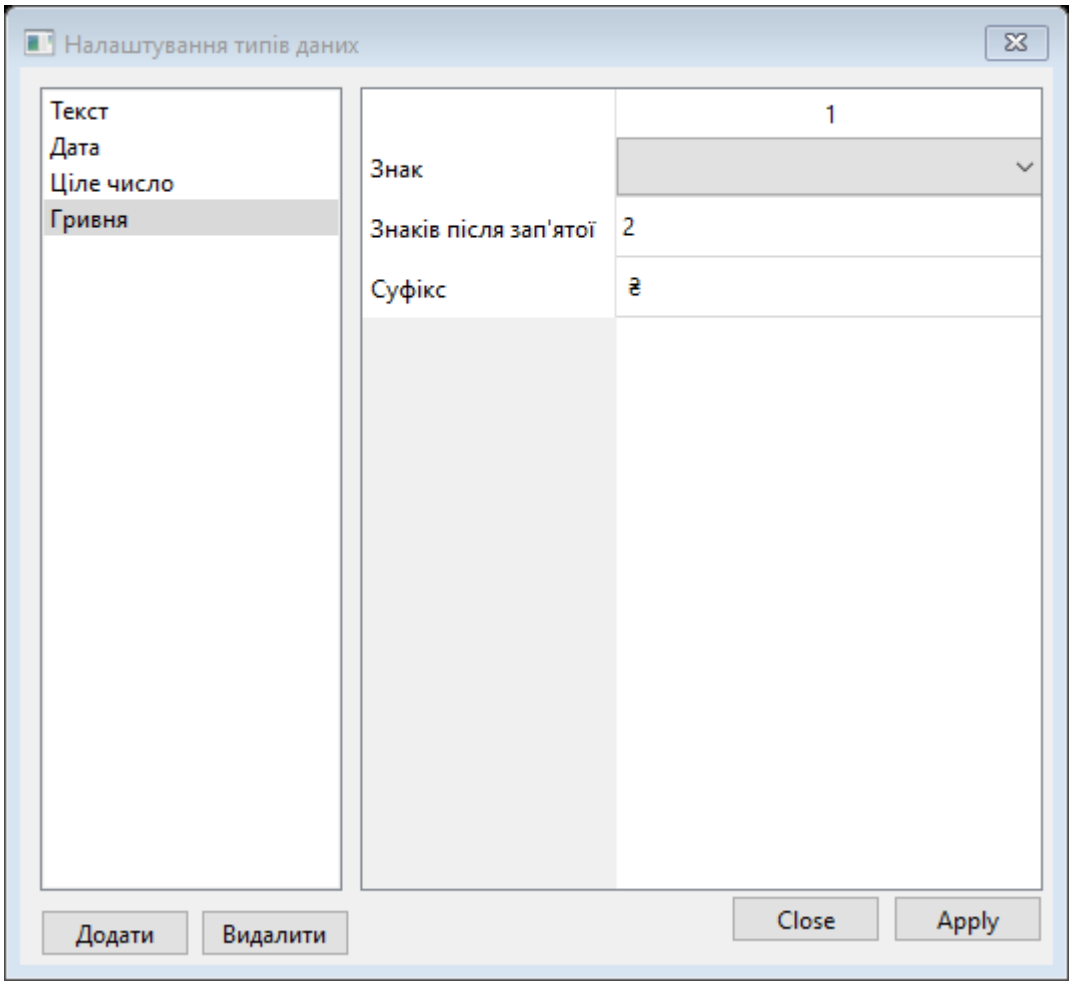


Рисунок 4.18 – Результат заповнення властивостей

Також, у меню «Налаштування» є пункт «Документи», при натисканні на котрий відкриває вікно «Створювання документів».

На рисунку 4.19 продемонстровано відкрите вікно.

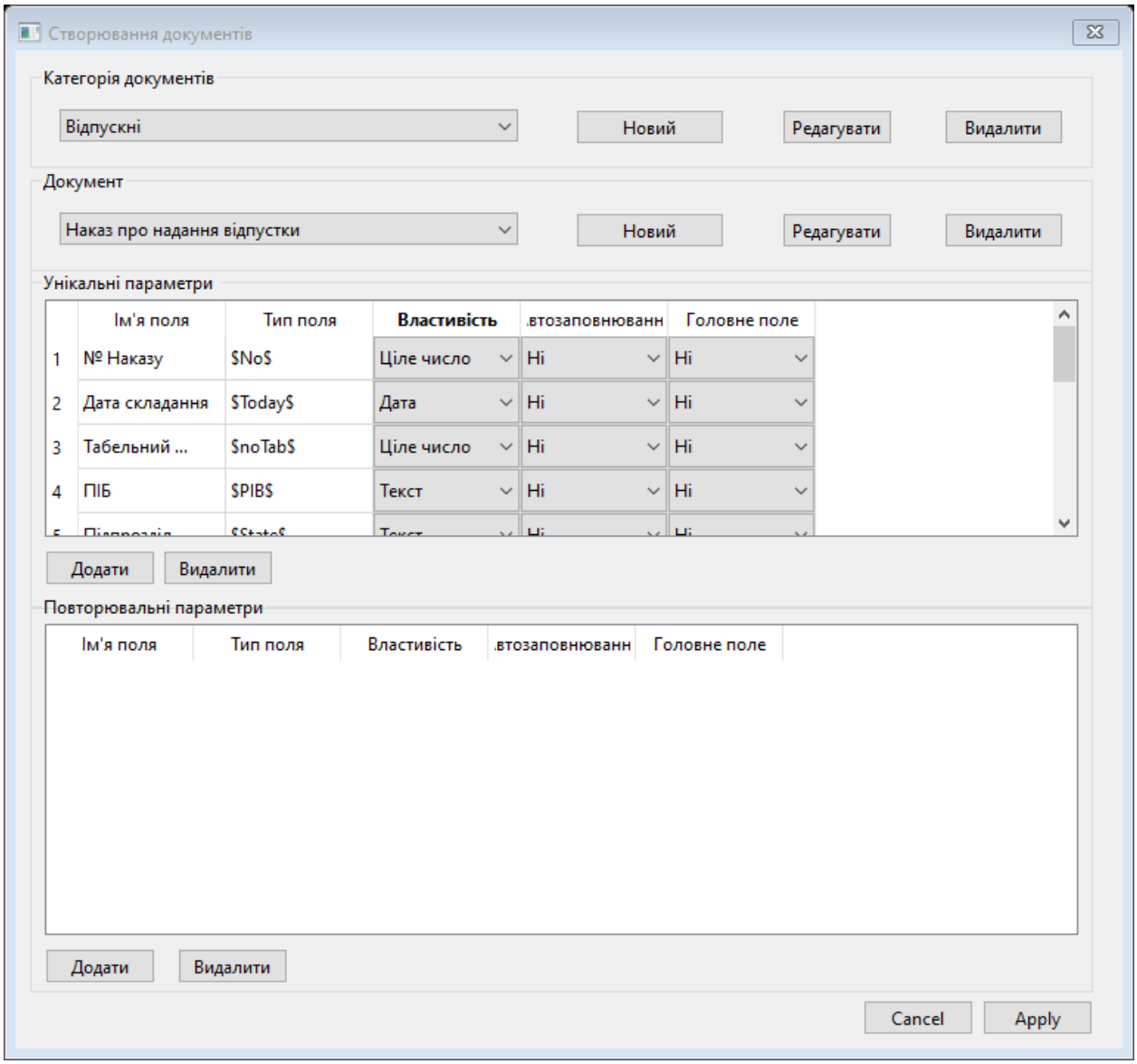


Рисунок 4.19 – Випадок, коли адміністратор вибрав пункт «Створювання документів»

Перш за все, необхідно або вибрати каталог документів, або створити новий. Також, є можливість видалення каталогу або його редагування. Після цього, треба або вибрати вже створений шаблон документів, або його створити. Також, як і для каталогів, є можливість редагування назви документів або його видалення. «Вибрати документ-шаблон». На рисунку 4.20 зображено створений шаблон документу.

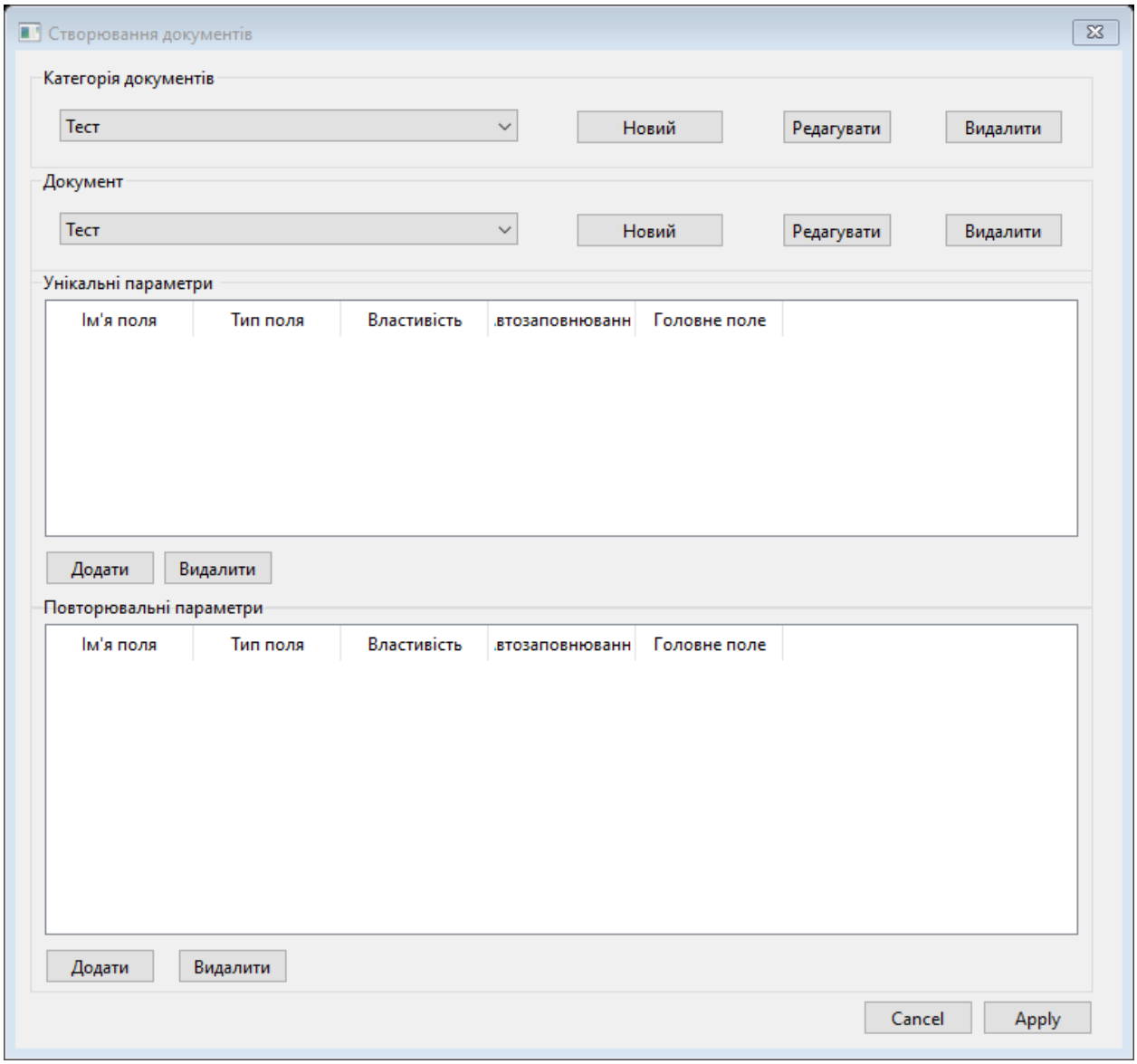


Рисунок 4.20 – Результат створення категорії тадокумента

Потім, необхідно заповнити таблиці унікальних та повторювальних параметрів. Під кожною таблицею є кнопка «Додати» – для додавання поля у відповідну таблицю, та «Видалити» – для видалення поля з відповідної таблиці. Після того, як таблиці будуть заповнені, необхідно натиснути на кнопку «Apply», щоб зберегти поля для шаблону документів, або – «Close», щоб закрити вікно та перейти к головному вікну. На рисунку 4.21 зображено заповнення тестового шаблону документу.

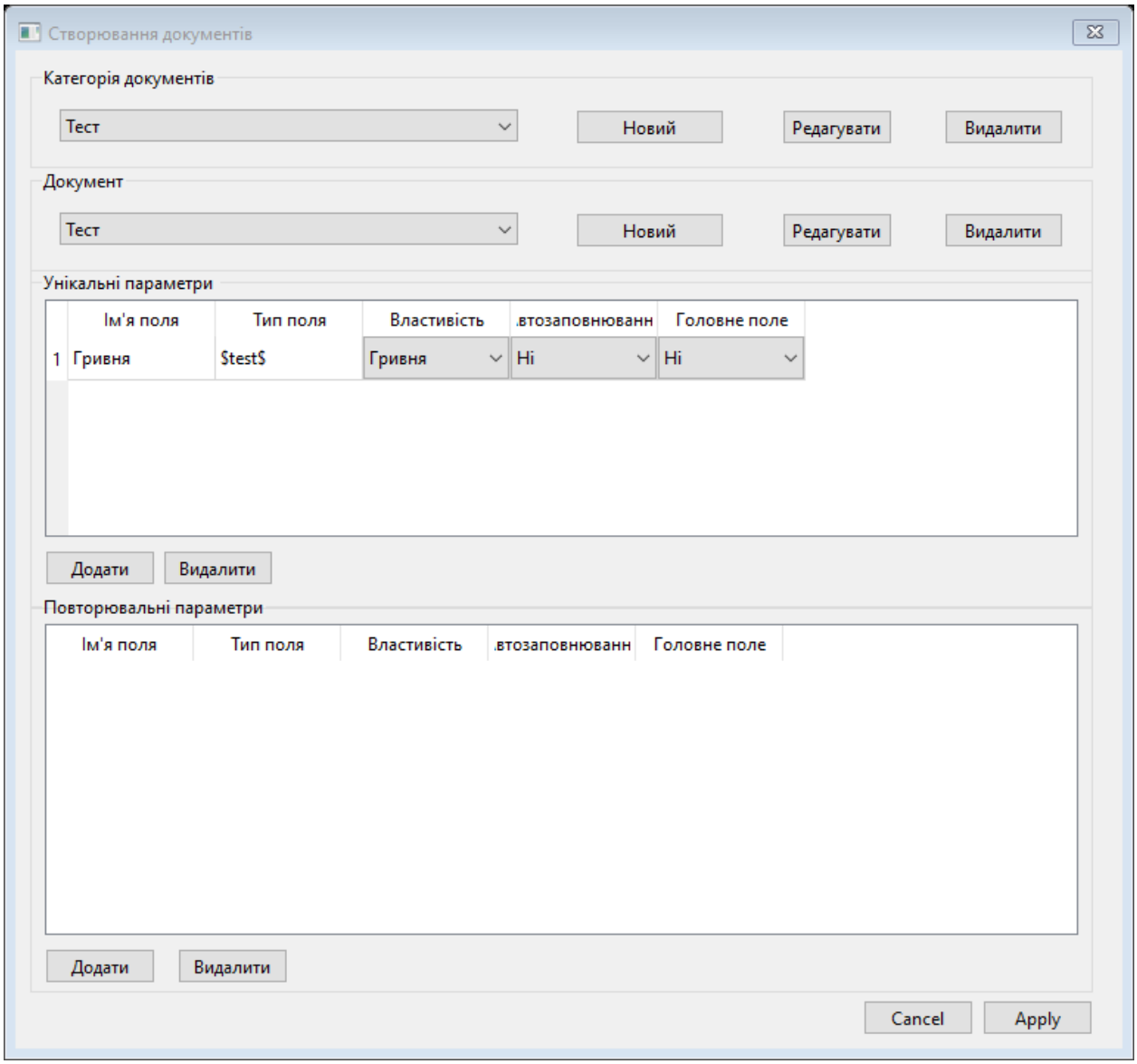


Рисунок 4.21 – Результат додавання параметру

Для того, щоб функцією заповнення документів необхідно мати встановлений пакет програм Microsoft Office та мати заздалегідь створені шаблони документів форматів .doc або .docx. На рисунку 4.22 зображено приклад створеного шаблон документу.

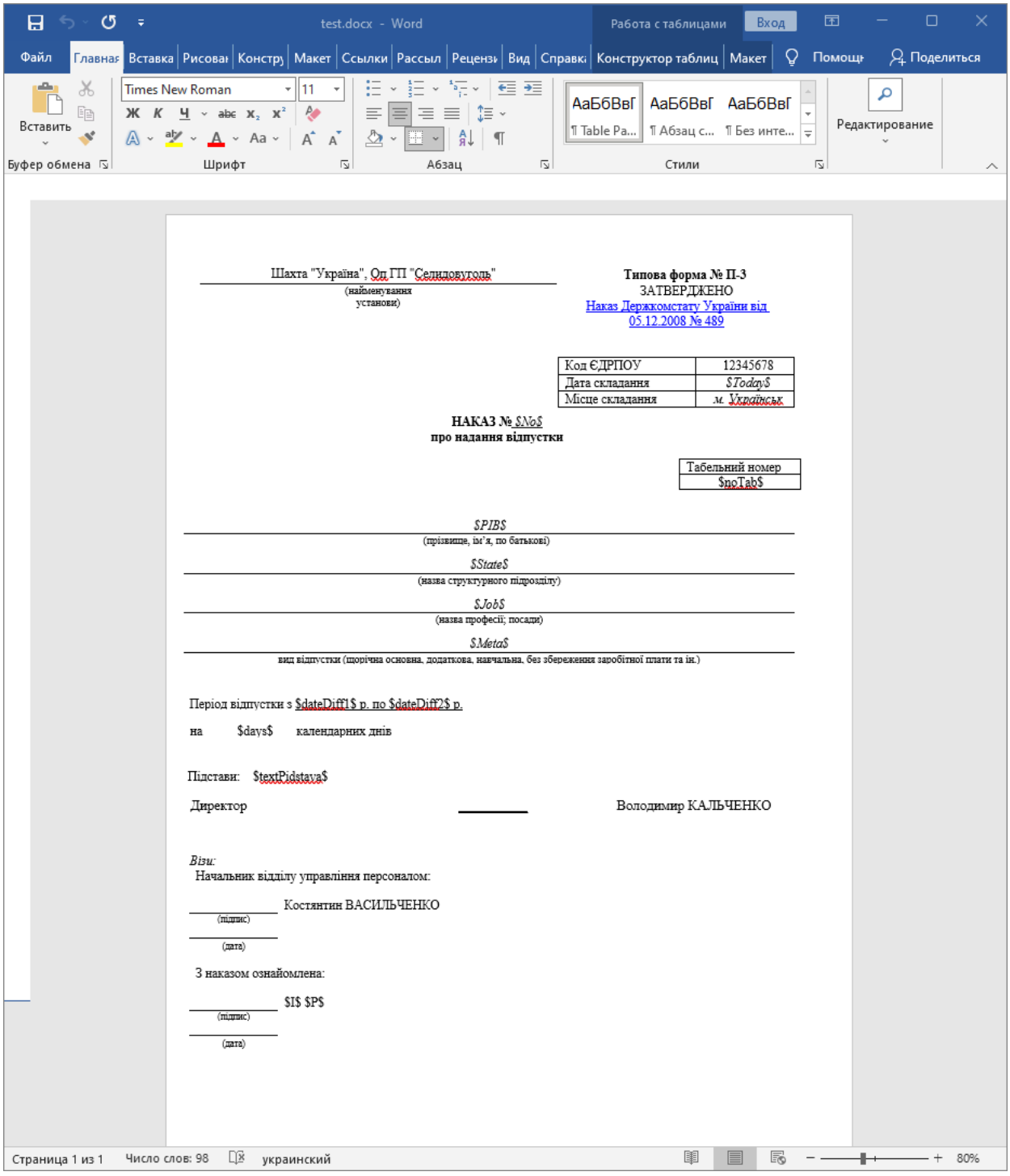


Рисунок 4.22 – Приклад створеного шаблону документа

4.2 Засоби безпеки

Функціонал додатку передбачає дві сутності: адміністратор та користувач. У адміністратора є доступ до меню «Налаштування» та «Адміністратор». Для того, щоб користувач не мав змоги змінювати шлях до бази даних через пункти «База даних системи» та «База даних підприємства»

меню «Налаштування», у пункті «Адміністратор» у меню «Користувач», необхідно стати адміністратором. Для цього, необхідно ввести правильний пароль у вікні «Введіть пароль» у пункті «Адміністратор» у меню «Користувач». Таким чином, якщо пароль є правильним, користувач стає адміністратором.

4.3 Висновки за розділом

У цьому розділі було виконано опис розробленої програмної системи, були описані всі можливі сценарії взаємодії користувача з додатком.

Також були описані засоби захисту, а саме перевірка на адміністратора.

5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

5.1 Функціональне тестування системи

Для перевірки модулів, які були створені під час розробки, необхідно провести тестування. Метод, який був обраний при тестуванні – функціональне тестування [22]. Цей метод тестування потрібен для перевірки правильності реалізації функцій програми. На рисунку 5.1 продемонстровано принцип функціонального тестування.

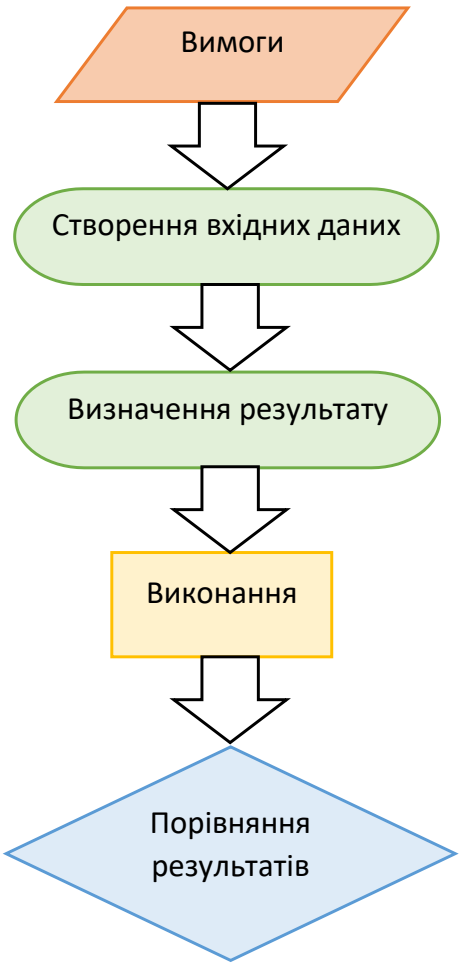


Рисунок 5.1 – Принцип функціонального тестування

Для тестування у середовищі Qt є бібліотека QtTest, котра призначена для написання тестів.

5.2 Тестування підключення баз даних

Дані для функціювання додатку та дані для заміщення знаходяться у базах даних. Додаток передбачає, що дані бази даних можуть бути підключені як локально, так і через мережу Інтернет. Для перевірки підключення було написано два тести для перевірки правильності підключення баз даних. На рисунку 5.2 продемонстровані тести для підключення баз даних.

```
50  void db_test::openDBTest_LOCAL()
51  {
52      QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE", "local");
53      db.setDatabaseName("C:/Users/Dready/Documents/document_system/test.db");
54      QVERIFY2(db.open(),
55               qPrintable(QString("open %1: %2")
56                          .arg(db.databaseName()).arg(db.driverName())));
57      QVERIFY(db.isValid());
58      QCOMPARE(db.isOpen(), true);
59      db.close();
60  }
61
62  void db_test::openDBTest_INTERNET()
63  {
64      QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE", "internet");
65      db.setHostName("localhost");
66      db.setDatabaseName("test.db");
67      QVERIFY2(db.open(),
68               qPrintable(QString("open %1: %2")
69                          .arg(db.databaseName()).arg(db.driverName())));
70      QVERIFY(db.isValid());
71      QCOMPARE(db.isOpen(), true);
72      db.close();
73  }
```

Рисунок 5.2 – Тести підключення до баз даних

На рисунку 5.3 відображено результати тестів.

```
19:26:53: Debugging C:\Users\Dready\Documents\build-test_system
********* Start testing of db_test *********
Config: Using QtTest library 6.3.0, Qt 6.3.0 (x86_64-little_end
PASS   : db_test::openDBTest_LOCAL()
PASS   : db_test::openDBTest_INTERNET()
PASS   : db_test::test_case1()
PASS   : db_test::cleanupTestCase()
Totals: 5 passed, 0 failed, 0 skipped, 0 blacklisted, 37ms
********* Finished testing of db_test *********19:26:54: Debugg
exit code 0.
```

Рисунок 5.3 – Результат виконання тестів

5.3 Тестування заміщення тексту

Для заповнення шаблону документа необхідно його скопіювати та підставити на місце полів, які заміщаються, інше значення. Для тестування заміщення тексту, було написано тест для перевірки (рис. 5.4).

```
77  void db_test::replaceWord()
78  {
79      QString documentDirOld = "C:/Users/Dready/Documents/document_system/test.docx";
80      QString documentDirNew = "C:/Users/Dready/Documents/document_system/test-"
81              +QDateTime::currentDateTime().toString("dd-MM-yyyy-HH-mm-ss")+".docx";
82      QVERIFY(QFile::copy(documentDirOld, documentDirNew));
83
84      QAxObject* wApp = new QAxObject("Word.Application", this);
85      wApp->setProperty("DisplayAlerts", false);
86      auto docs = wApp->querySubObject("Documents");
87      auto doc  = docs->querySubObject("Open(QString)",documentDirNew);
88
89
90      auto active  = wApp->querySubObject("ActiveDocument()");
91      auto content = active->querySubObject("Content()");
92      auto find    = content->querySubObject("Find");
93
94      QString sOld = "sOld";
95      QString sNew = "sNew";
96
97      bool bMatchCase      = false;
98      bool bMatchWholeWord = false;
99      bool bMatchWildCards = false;
100     bool bReplaceAll     = true;
101
102     QVariantList vl = { sOld, bMatchCase, bMatchWholeWord, bMatchWildCards, false, false, true,
103                         1, false, sNew, bReplaceAll ? "2" : "1" };
104     QVERIFY(find->dynamicCall("Execute(QString,bool,bool,bool,bool,bool,bool"
105                               ",int,bool,QString,int)",vl).toBool());
106
107     delete find;
108     delete content;
109     delete active;
110     delete doc;
111     delete docs;
112     delete wApp;
113 }
```

Рисунок 5.4 – Тести заміщення тексту

Результати виконання тесту продемонстровані на рисунку 5.5.

```
19:38:25: Debugging C:\Users\Dready\Documents\build-test_system-D
********* Start testing of db_test *********
Config: Using QtTest library 6.3.0, Qt 6.3.0 (x86_64-little_endia
PASS   : db_test::replaceWord()
PASS   : db_test::test_case1()
PASS   : db_test::cleanupTestCase()
Totals: 4 passed, 0 failed, 0 skipped, 0 blacklisted, 7ms
********* Finished testing of db_test *********19:38:26: Debuggin
exit code 0.
```

Рисунок 5.5 – Результат виконання тестів

5.4 Висновки за розділом

У поточному розділі було описано метод тестування, а саме функціональне тестування.

Було написано два тести для підключення баз даних:

– тест на підключення локальної бази даних;

– тест на підключення бази даних через мережу Інтернет.

Також було написано тест перевірки успішного заміщення тексту.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проведено аналіз вибраної теми, було визначено об’єкт і предмет дослідження, наведена мета робота, методи дослідження та їх обґрунтування. Був проведений аналіз конкурентних продуктів. Для розробки додатку у були вибрані інструменти, засоби і технології, а саме:

– технологія COM;

– технологія OLE;

– метод програмування – об’єктно-орієнтоване програмування;

– графічна мова UML для побудови UML-діаграм.

Були визначені основні завдання роботи.

Під час проєктування системи було описано архітектуру системи, побудовано UML-діаграми:

– використання;

– розміщення.

Також була обрана мова програмування C++ та середовище розробки QtCreator. Для роботи з базами даних було обрано графічний додаток BD Browser.

У ході розробки додатку було сконструйовано інтерфейс, було описано усі модулі програми та функції.

Під час тестування програмної системи було використано функціональне тестування, було виконано два тести підключення баз даних та тест заміщення тексту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ф.О Чмиленко, Л.П. Жук Посібник до вивчення дисципліни «Методологія та організація наукових досліджень» Дніпропетровськ РВВ ДНУ», 2014 – 48 с.

2. Методологія та організація наукових досліджень: навчальний посібник / Б. І. Мокін, О. Б. Мокін. – 2-е вид., змін. та доп. – Вінниця : ВНТУ, 2015. – 317 с.

3. BAS [електронний ресурс] – Режим доступу: https://www.bas-soft.eu/.

4. Сервіс «Вчасно» [електронний ресурс] – Режим доступу: https://vchasno.ua/.

5. . Компанія «Актив-Софт» [електронний ресурс] – Режим доступу: https://aktiv.ua/ua.

6. OLE and Data Transfer [електронний ресурс] – Режим доступу: https://docs.microsoft.com/uk-ua/windows/win32/com/ole-and-data-transfer.

7. SQLite3 [електронний ресурс] – Режим доступу: https://docs.python.org/3/library/sqlite3.html.

8. DB Browser [електронний ресурс] – Режим доступу: https://sqlitebrowser.org.

9. Active Qt [електронний ресурс] – Режим доступу: https://doc.qt.io/qt-5/activeqt-index.html.

10. QFile [електронний ресурс] – Режим доступу: https://doc.qt.io/qt-5/qfile.html.

11. Qt [електронний ресурс] – Режим доступу: https://www.qt.io/.

12. Буч Г. Язык UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. – М. : ДМК, 2013. – 432 с.

13. DrawIO [електронний ресурс] – Режим доступу: https://www.diagrams.net.

ДОДАТОК А
ЗАУВАЖЕННЯ НОРМОКОНТРОЛЕРА

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
ПЗ		Нещільне заповнення сторінок пояснювальної записки
С. 62		Замало пунктів у списку джерел, помилки у оформленні списку
С. 57		Посилання на не існуюче джерело

ДОДАТОК Б
ГРАФІЧНА ЧАСТИНА

Рисунок Б.1 – Титульний слайд

Рисунок Б.2 – Діаграма варіантів використання

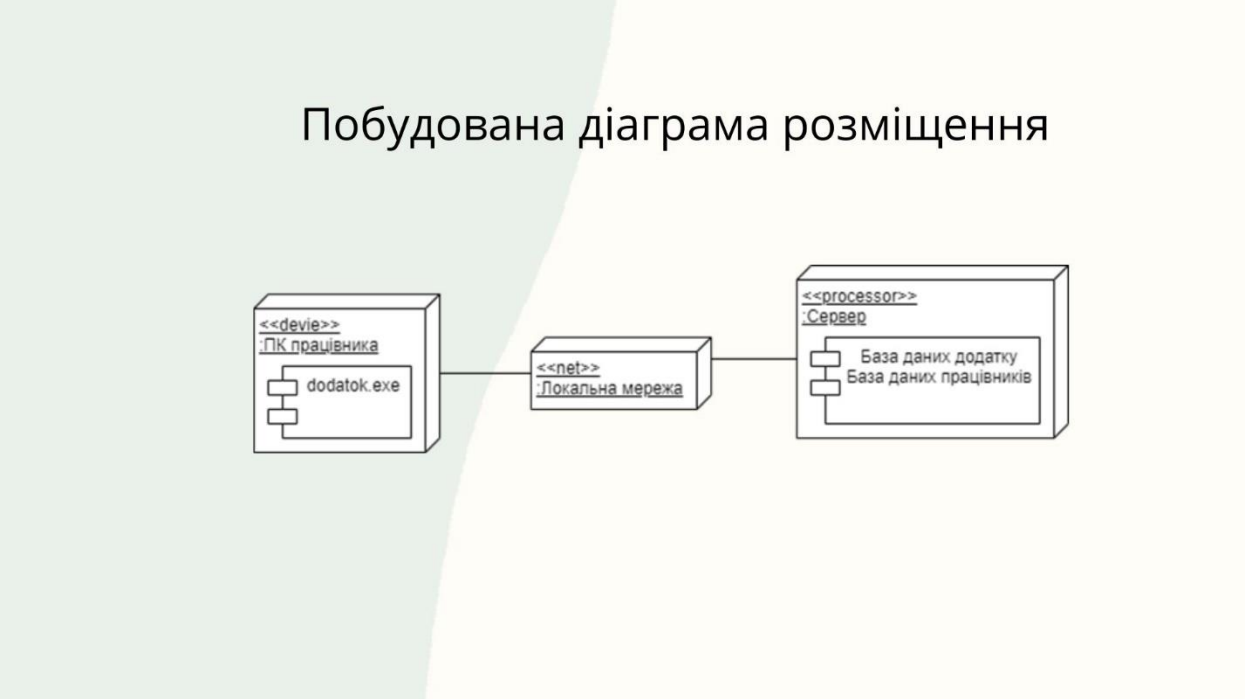


Рисунок Б.3 – Діаграма розміщення

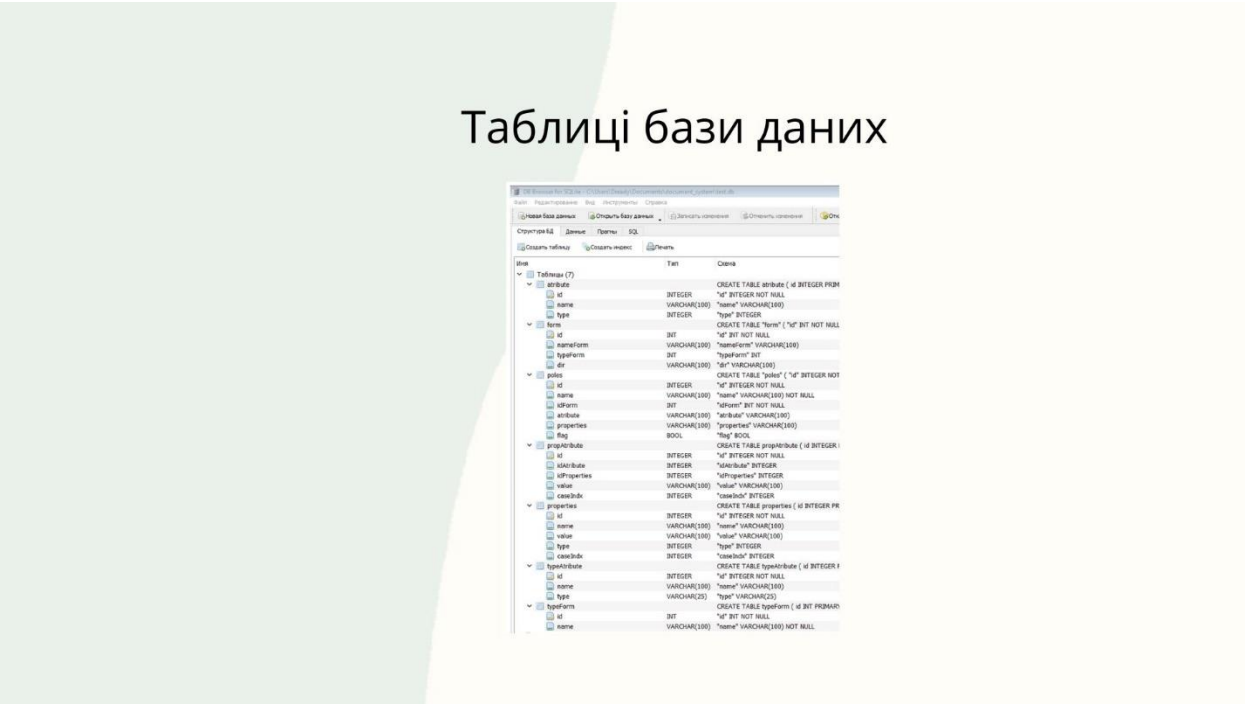


Рисунок Б.4 – Таблиці бази даних

Рисунок Б.5 – Алгоритм заповнення шаблонів

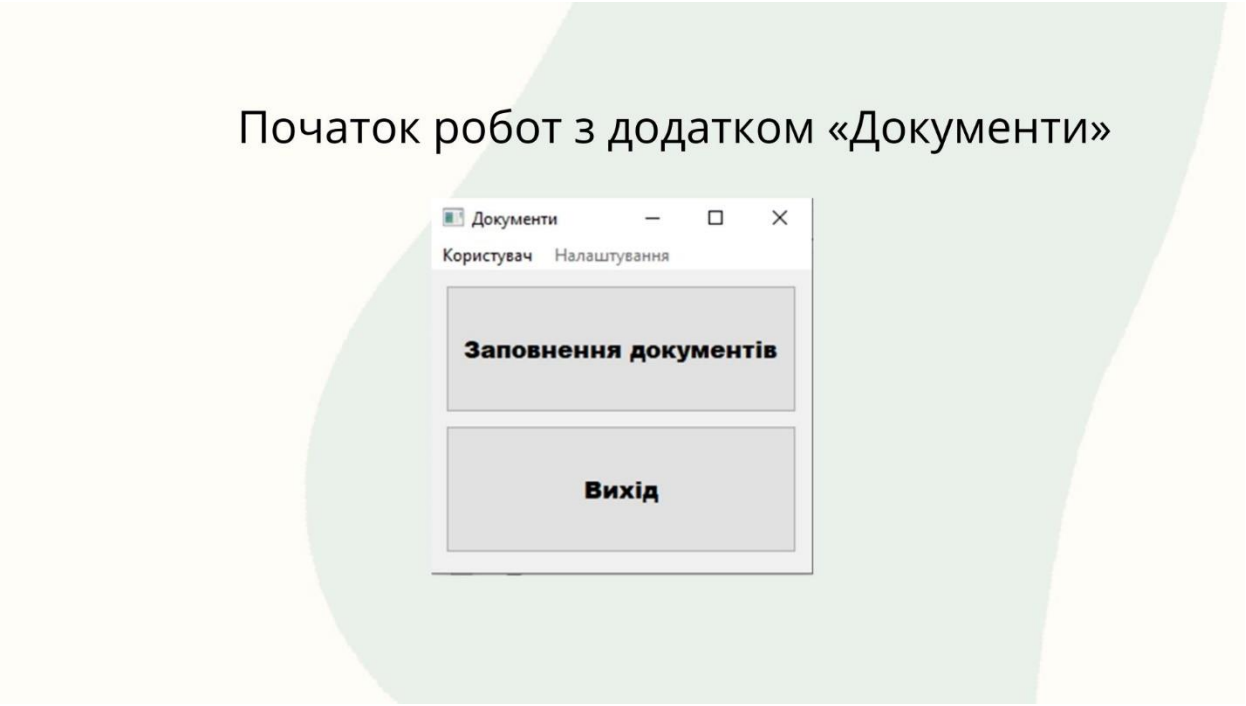


Рисунок Б.6 – Головне вікно додатку «Документи»

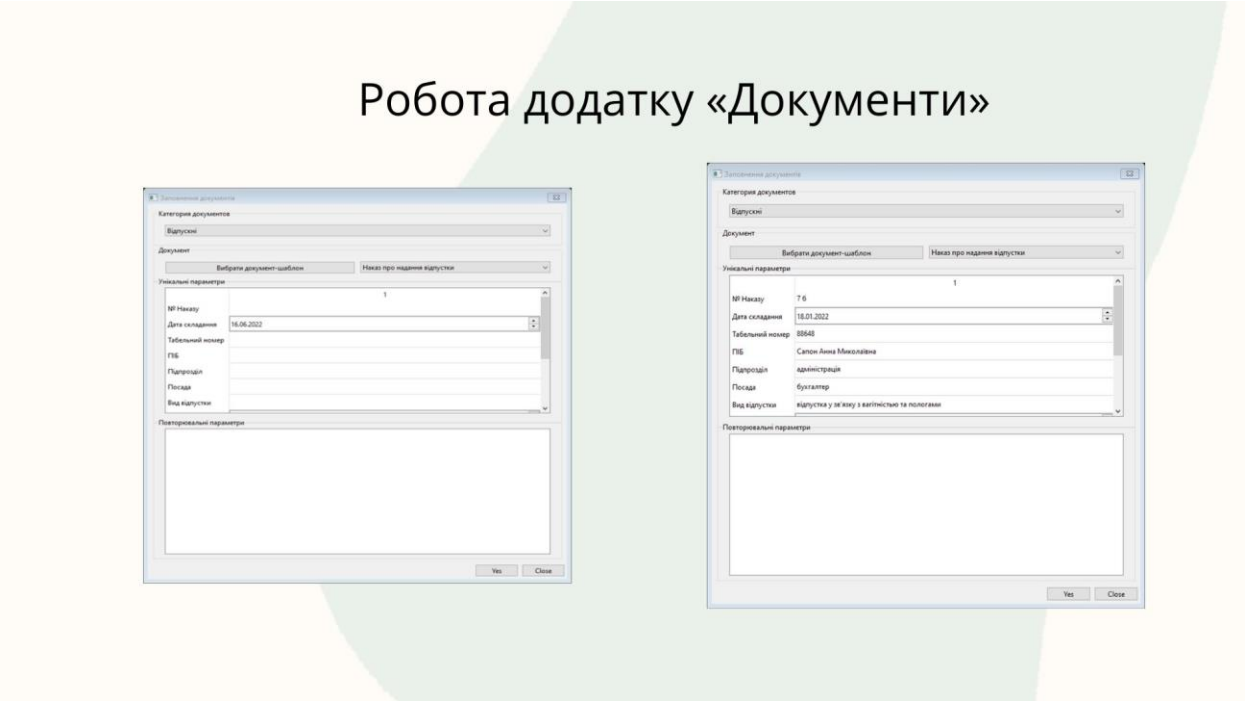


Рисунок Б.7 – Робота додатку «Документи»

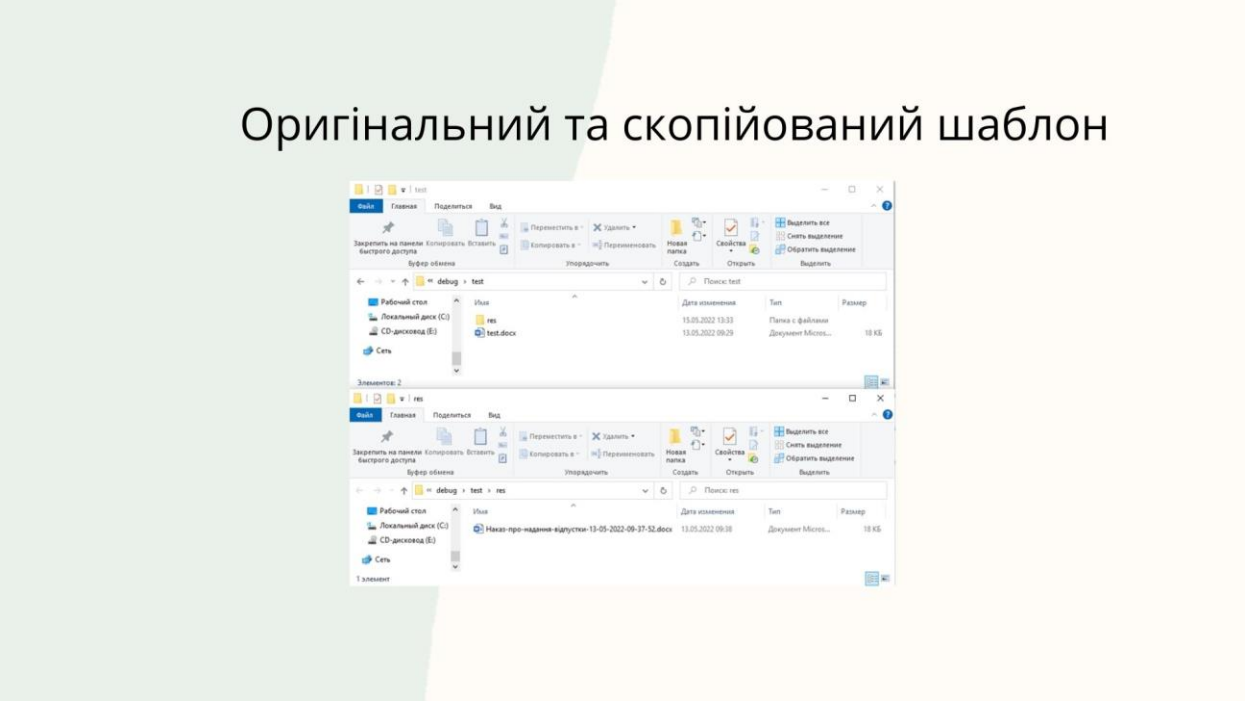


Рисунок Б.8 – Оригінальний та заповнений шаблон

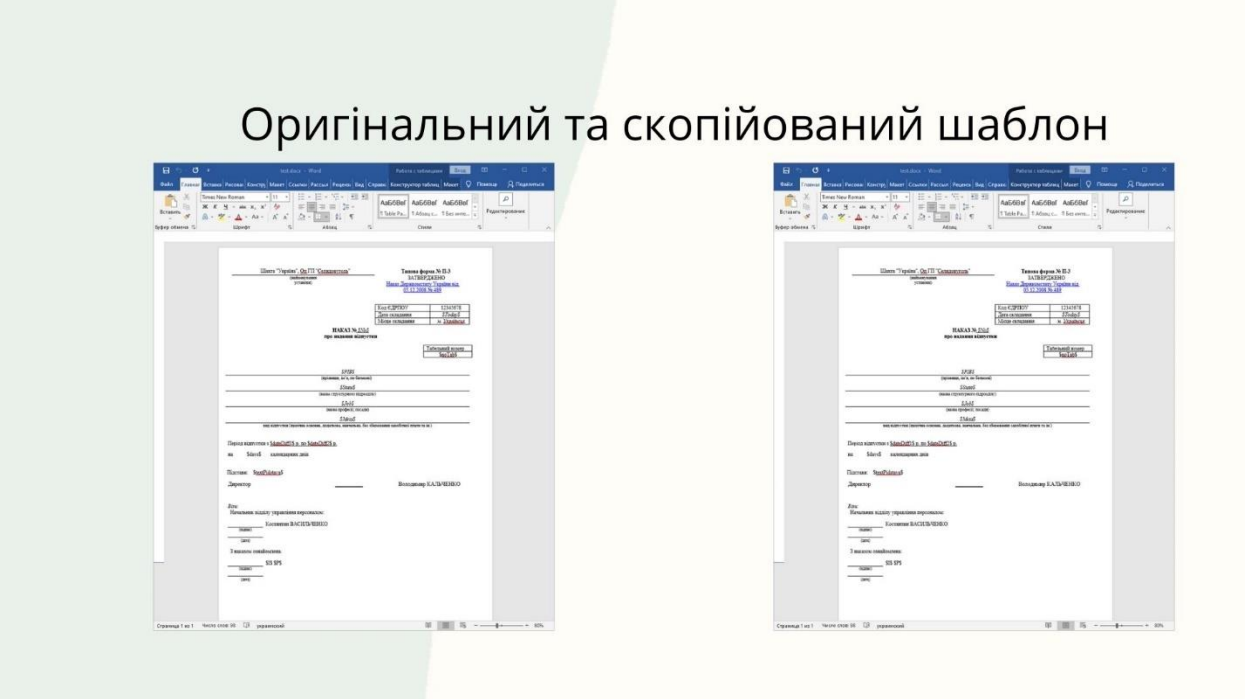


Рисунок Б.9 – Оригінальний та заповнений шаблон

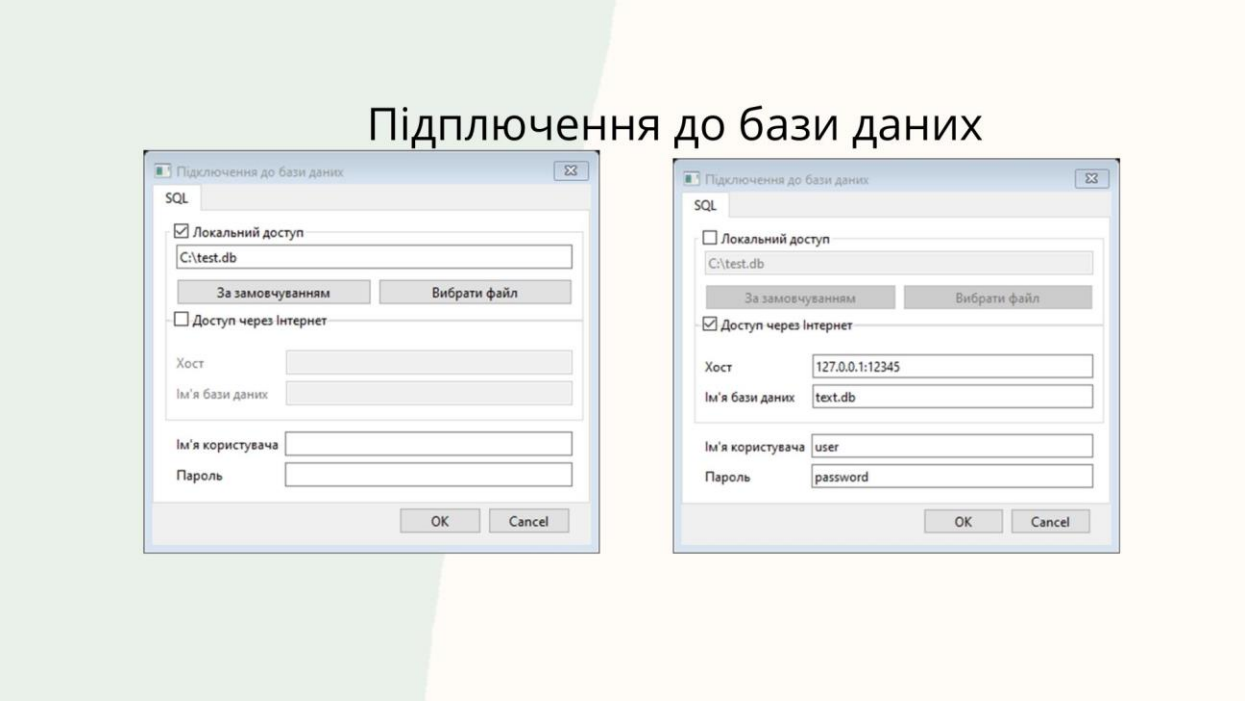


Рисунок Б.10 – Підключення до бази даних

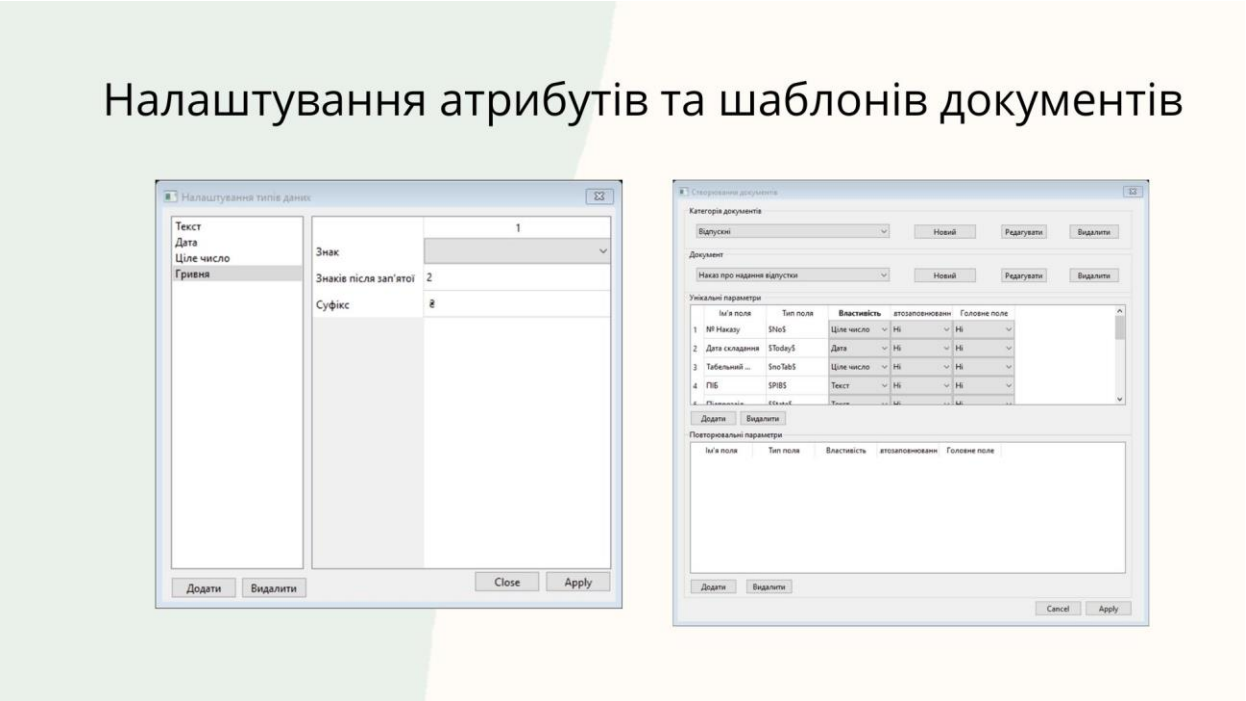


Рисунок Б.11 – Налаштування атрибутів та шаблонів

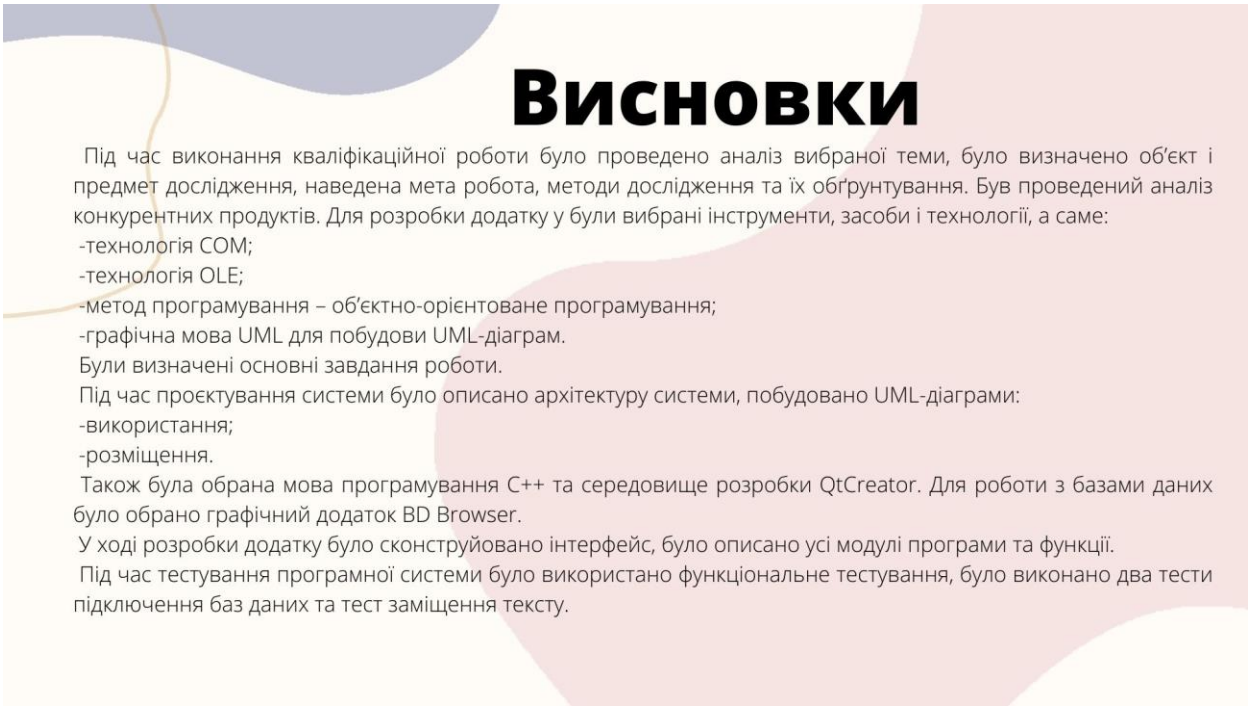


Рисунок Б.12 – Висновки

Додаток В.1

Лістинг модуля document_system.pro

```
QT       += core gui
QT += axcontainer axserver
QT += sql
QT += widgets

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

#Include file(s)
include(delegate.pri)

# 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 += \
    atributeform.cpp \
    createdocform.cpp \
    dbconnect.cpp \
    dbdirform.cpp \
    filldocform.cpp \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    atributeform.h \
    createdocform.h \
    dbconnect.h \
```

```
    dbdirform.h \
    filldocform.h \
    mainwindow.h

FORMS += \
    atributeform.ui \
    createdocform.ui \
    dbdirform.ui \
    filldocform.ui \
    mainwindow.ui

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

Додаток В.2

Лістинг модуля main.cpp

```
#include "mainwindow.h"
#include "dbconnect.h"

#include <QApplication>


int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    DBConnect::connect();
    MainWindow w;
    w.show();

    return a.exec();

}
```

Додаток В.3

Лістинг модуля dbconnect

dbconnect.h

```
#ifndef DBCONNECT_H
#define DBCONNECT_H
#include <QtSql/QSqlDatabase>
#include <QDir>
#include <QVariant>
#include<QDebug>
class DBConnect
{
public:
    DBConnect();
    ~DBConnect();
    static void connect();


private:

    QString setDir;
};
#endif // DBCONNECT_H
```

Dbconnect.cpp

```
#include "dbconnect.h"
#include <QSqlQuery>
#include<QDebug>
#include <QSqlError>

DBConnect::DBConnect()
{
    setDir = QString( QDir::currentPath() + "/test/test.docx" );
```

```
}

void DBConnect::connect()
{

    qDebug() << QString( QDir::currentPath() + "/test/test.docx" );
    QString dir = QString( QDir::currentPath() + "/settings.data" );
    QString line;
    QStringList paramConst =
{"dbSystem","sHostName","sDatabaseName","sUserName","sPassword",

"dbCompany","cHostName","cDatabaseName","cUserName","cPassword"};

    QStringList param;

    int i=0;

/*
    QSqlDatabase db = QSqlDatabase::addDatabase("QODBC");
    db.setDatabaseName("DRIVER={Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm,
*.xlsb)};DBQ=C:/Users/Dready/Documents/document_system/1.xls");
    if(db.open())
    {
        QSqlQuery query("select * from [" + QString("Sheet1") + "$]");
        // Select range, place A1:B5 after $
        while (query.next())
        {
            QString column1= query.value(0).toString();
            qDebug() << column1;
        }
```

```
    }
    else {
        qDebug() << db.lastError().text();
    }
*/

    qDebug() << QDir(QDir::currentPath()).exists("settings.txt");
    qDebug() << QDir(QDir::currentPath()+"/test").exists("test.docx");

    QFile file(dir);
    qDebug () << file.exists() << dir  << QFile::exists(dir);



    if (file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug() << "open";
        QSqlDatabase dbSystem = QSqlDatabase::addDatabase("QSQLITE",
"dbSystem");
        while (!file.atEnd())
        {
            line = file.readLine();
            while(i < 10)
            {
                if(paramConst[i]==line.section(" ", 0, 0))
                    break;
                ++i;
            }
            param.append(line.section(" ", 1, 1));

            qDebug() << param;
```

```
            switch(i)
            case 0:;




        }
        file.close();
    }

    QSqlDatabase dbSystem = QSqlDatabase::addDatabase("QSQLITE",
"dbSystem");
//    db.setHostName("localhost");

dbSystem.setDatabaseName("C:/Users/Dready/Documents/document_system/test.
db");
//    db.setUserName("root");
//    db.setPassword("root");
    if(!dbSystem.open())
    {
        qDebug() << "Unable to create a table1";
    } else dbSystem.close();

/*
    auto Cloud = QSqlDatabase::addDatabase("QODBC");
    Cloud.setDatabaseName("DRIVER={Microsoft Excel Driver (*.xls, *.xlsx,
*.xlsm, *.xlsb)};DBQ=C:/Users/Dready/Documents/document_system/1.xlsx");
    if(Cloud.open())
    {
        QSqlQuery query("select * from [" + QString("Sheet1") + "$A1:B5]", Cloud);
```

```
        // Select range, place A1:B5 after $
        while (query.next())
        {
            QString column1= query.value(0).toString();
            qDebug() << column1;
        }
    }
    else {
        qDebug() << Cloud.lastError().text();
    }
/*
    QSqlDatabase db;
    QString connection = QString(QSqlDatabase::defaultConnection);
    db = QSqlDatabase::addDatabase("QSQLITE", connection);
    QString pathToDB = "C:/Users/Dready/Documents/document_system/test.db";
    db.setDatabaseName(pathToDB);
*//*
    if(!db.open())
    {
       qDebug() << "Unable to create a table1";
    }
*/
//    QSqlQuery query;
/*   Creating of the data base
//    query.exec("drop table form");
//    query.exec("drop table typeForm");
//    query.exec("drop table poles");
//    query.exec("drop table atribute");
//    query.exec("drop table properties");
/*    QString   str1  = "CREATE TABLE form ( "
```

```
            "id INT PRIMARY KEY NOT NULL,"
            "nameForm   VARCHAR(100), "
            "typeForm  INT,"
            "dir VARCHAR(100) "
            ");";
QString   str2  = "CREATE TABLE typeForm ( "
            "id INT PRIMARY KEY NOT NULL, "
            "name   VARCHAR(100) NOT NULL "
            ");";
QString   str3  = "CREATE TABLE poles ( "
            "id INT PRIMARY KEY NOT NULL, "
            "name   VARCHAR(100), "
            "idForm INT NOT NULL, "
            "atribute  VARCHAR(100), "
            "properties VARCHAR(100), "
            "flag  BOOL "
            ");";
QString   str4  = "CREATE TABLE atribute ( "
            "id INTEGER PRIMARY KEY NOT NULL, "
            "name  VARCHAR(100),"
            "type INTEGER "
            ");";
QString   str5  = "CREATE TABLE properties ( "
            "id INTEGER PRIMARY KEY NOT NULL,"
            "name VARCHAR(100), "
            "value VARCHAR(100), "
            "type INTEGER, "
            "caseIndx INTEGER"
            ");";
QString   str6  = "CREATE TABLE typeAtribute ( "
```

```
            "id INTEGER PRIMARY KEY NOT NULL, "
            "name VARCHAR(100), "
            "type INTEGER"
            ");";
QString   str7  = "CREATE TABLE propAtribute ( "
            "id INTEGER PRIMARY KEY NOT NULL, "
            "idAtribute INTEGER, "
            "idProperties INTEGER, "
            "value VARCHAR(100), "
            "caseIndx INTEGER"
            ");";


if (!query.exec(str1)) {
   qDebug() << query.lastError();
}
if (!query.exec(str2)) {
   qDebug() << query.lastError();
}
if (!query.exec(str3)) {
   qDebug() << query.lastError();
}
if (!query.exec(str4)) {
   qDebug() << query.lastError();
}
if (!query.exec(str5)) {
   qDebug() << query.lastError();
}
if (!query.exec(str6)) {
   qDebug() << query.lastError();
```

```
}
if (!query.exec(str7)) {
    qDebug() << query.lastError();
}

query.exec("SELECT * FROM typeAtribute");
query.last();
if(query.at() != 2)
{
    query.exec("DELETE * FROM typeAtribute");
    query.exec("INSERT INTO typeAtribute (id, name, type) "
        "VALUES (0, 'Текст', 0)");
    query.exec("INSERT INTO typeAtribute (id, name, type) "
        "VALUES (1, 'Дата та час', 1)");
    query.exec("INSERT INTO typeAtribute (id, name, type) "
        "VALUES (2, 'Число', 2)");
}

query.exec("SELECT * FROM properties");
query.last();
if(query.at() != 5)
{
    query.exec("DELETE * FROM properties");
    query.exec("INSERT INTO properties (id, name, value, type, caseIndx) "
        "VALUES (0, 'Довжина', 0, 0, 1)");
    query.exec("INSERT INTO properties (id, name, value, type, caseIndx) "
        "VALUES (1, 'Форматирование', 0, 0, 2)");
    query.exec("INSERT INTO properties (id, name, value, type, caseIndx) "
        "VALUES (2, 'Знак', 0, 1, 3)");
    query.exec("INSERT INTO properties (id, name, value, type, caseIndx) "
```

```
            "VALUES (3, 'Знаков после запятой', 2, 1, 1)");
        query.exec("INSERT INTO properties (id, name, value, type, caseIndx) "
            "VALUES (4, 'Суфикс', '', 1, 0)");
        query.exec("INSERT INTO properties (id, name, value, type, caseIndx) "
            "VALUES (5, 'Формат', '', 2, 0)");
    }*/

/*
    //! Добавить драйвер базы данных
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
     //! Устанавливаем имя базы данных
    db.setDatabaseName(QDir::currentPath()+"/test.db");
     //! Открываем базу данных
    if(!db.open())
        qDebug() << "Unable to create a table5";
            //! Следующее выполняет соответствующий оператор sql
        QSqlQuery query;
             //! Очистить таблицу учеников
       query.exec("drop table student");
            //! Создайте новую таблицу учеников, установите id в качестве
первичного ключа и элемент имени
        if(!query.exec("create table student (id int primary key, name vchar,course
int)"))
            qDebug() << "Unable to create a table1";
*/
}
```

Додаток В.4

Лістинг модуля mainwindow

Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

#include <createdocform.h>
#include <filldocform.h>
#include <atributeform.h>
#include <dbdirform.h>

#include <QInputDialog>
#include <QLineEdit>
#include <QFile>
#include <QMessageBox>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class CreateDocForm;
class fillDocForm;
class atributeForm;
class DBDirForm;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
```

```
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();


private slots:

    void on_start_clicked();

    void on_exit_clicked();

    void on_documentAction_triggered();

    void on_typesAction_triggered();

    void on_loginUser_triggered();

    void on_loginAdmin_triggered();

    void on_changePass_triggered();




    void on_dbAction_triggered();

private:
    Ui::MainWindow *ui;

    CreateDocForm *SecondWindow;
    fillDocForm *ThirdWindow;
```

```
    atributeForm *FourWindow;
    DBDirForm *FiveWindow;

    bool FuncPass ();

};
#endif // MAINWINDOW_H
```

Mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    SecondWindow = new CreateDocForm();
    ThirdWindow = new fillDocForm();
    FourWindow = new atributeForm();
    FiveWindow = new DBDirForm();
    ui->loginUser->setEnabled(false);
    ui->loginAdmin->setEnabled(true);
    ui->changePass->setEnabled(false);
    ui->setMenu->setEnabled(false);

}

MainWindow::~MainWindow()
{
    delete ui;
```

```
}



void MainWindow::on_start_clicked()
{
    ThirdWindow->show();
}

void MainWindow::on_exit_clicked()
{
    close();
}

void MainWindow::on_documentAction_triggered()
{
    SecondWindow->show();
}

void MainWindow::on_typesAction_triggered()
{
    FourWindow->show();
}
```

```
void MainWindow::on_loginUser_triggered()
{
   ui->loginUser->setEnabled(false);
   ui->loginAdmin->setEnabled(true);
   ui->changePass->setEnabled(false);
   ui->setMenu->setEnabled(false);
   ui->loginMenu->setTitle("Користувач");
}


void MainWindow::on_loginAdmin_triggered()
{

   if (FuncPass())
   {
      ui->loginUser->setEnabled(true);
      ui->loginAdmin->setEnabled(false);
      ui->changePass->setEnabled(true);
      ui->setMenu->setEnabled(true);
      ui->loginMenu->setTitle("Адміністратор");
   }
}


void MainWindow::on_changePass_triggered()
{
   if (FuncPass())
   {
      bool ok;
      QFile file(QString( QCoreApplication::applicationDirPath() + "/pass.data" ));
```

```
        if(file.open(QIODevice::WriteOnly | QIODevice::Text))
        {
            QString newPass = QInputDialog::getText(this, tr("Введіть новий
пароль"),
                                                tr("Новий пароль:"), QLineEdit::Normal,
                                                "", &ok);



            if (ok && !newPass.isEmpty())
            {
                file.resize(0);

                QTextStream out(&file);
                out << newPass;

                QMessageBox msgBox;
                msgBox.setWindowTitle("Успішно");
                msgBox.setText("Пароль було змінено.");
                msgBox.exec();
            }
            file.close();
        }   else qDebug() << "No File";

    }

}

bool MainWindow::FuncPass ()
{
```

```
    bool ok;
    QFile file(QString( QCoreApplication::applicationDirPath() + "/pass.data" ));
    if(file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        QString pass = file.readAll();
        qDebug() << pass;
        file.close();
        QString checkPass = QInputDialog::getText(this, tr("Введіть пароль"),
                                        tr("Пароль:"), QLineEdit::Normal,
                                        "", &ok);

        if(ok && !checkPass.isEmpty() && pass == checkPass)
            return true;

        QMessageBox msgBox;
        msgBox.setWindowTitle("Помилка");
        msgBox.setText("Введено не правильний пароль.");
        msgBox.exec();
        return false;

    }   else qDebug() << "No File";

    return false;
}

void MainWindow::on_dbAction_triggered()
{
    FiveWindow->show();
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>272</width>
    <height>237</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Документи</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="start">
    <property name="geometry">
     <rect>
      <x>10</x>
      <y>10</y>
      <width>251</width>
      <height>91</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>Arial Black</family>
      <pointsize>13</pointsize>
```

```
  </font>
 </property>
 <property name="text">
  <string>Заповнення документів</string>
 </property>
 <property name="checkable">
  <bool>false</bool>
 </property>
</widget>
<widget class="QPushButton" name="exit">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>110</y>
   <width>251</width>
   <height>91</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <family>Arial Black</family>
   <pointsize>13</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Вихід</string>
 </property>
</widget>
</widget>
<widget class="QMenuBar" name="menubar">
```

```
<property name="geometry">
 <rect>
  <x>0</x>
  <y>0</y>
  <width>272</width>
  <height>22</height>
 </rect>
</property>
<widget class="QMenu" name="loginMenu">
 <property name="title">
  <string>Користувач</string>
 </property>
 <addaction name="loginUser"/>
 <addaction name="loginAdmin"/>
 <addaction name="separator"/>
 <addaction name="changePass"/>
</widget>
<widget class="QMenu" name="setMenu">
 <property name="title">
  <string>Налаштування</string>
 </property>
 <addaction name="documentAction"/>
 <addaction name="typesAction"/>
 <addaction name="separator"/>
 <addaction name="dbAction"/>
 <addaction name="action_2"/>
</widget>
<addaction name="loginMenu"/>
<addaction name="setMenu"/>
</widget>
```

```
<action name="documentAction">
 <property name="text">
  <string>Документи</string>
 </property>
</action>
<action name="typesAction">
 <property name="text">
  <string>Типи даних</string>
 </property>
</action>
<action name="loginUser">
 <property name="text">
  <string>Користувач</string>
 </property>
</action>
<action name="loginAdmin">
 <property name="text">
  <string>Администратор</string>
 </property>
</action>
<action name="changePass">
 <property name="text">
  <string>Змінити пароль</string>
 </property>
</action>
<action name="dbAction">
 <property name="text">
  <string>База даних системи додатку</string>
 </property>
</action>
```

```
  <action name="action">
   <property name="text">
    <string>База даних працівників</string>
   </property>
  </action>
  <action name="action_2">
   <property name="text">
    <string>База даних підприємства</string>
   </property>
  </action>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Додаток В.5

Лістинг модуля atributeform

Atributeform.h

```
#ifndef ATRIBUTEFORM_H
#define ATRIBUTEFORM_H

#include <QDialog>

#include <mainwindow.h>

#include "QStandardItemModel"
#include "QStandardItem"
#include "QStringListModel"
#include "qabstractbutton.h"
#include "qsqlquery.h"
#include <QSqlDatabase>
#include <QSqlTableModel>
#include <QItemSelectionModel>
#include <QListView>


namespace Ui {
class atributeForm;
}

class atributeForm : public QDialog
{
    Q_OBJECT

public:
    explicit atributeForm(QWidget *parent = nullptr);
    ~atributeForm();
```

```
private slots:
    void on_pushButton_add_clicked();

    void on_pushButton_delete_clicked();

    void rowChangedSlot(const QModelIndex &curIndex, const QModelIndex
&prevIndex);

    void tableChangedSlot(const QModelIndex &curIndex, const QModelIndex
&prevIndex);




    void on_buttonBox_accepted();

    void on_buttonBox_clicked(QAbstractButton *button);

private:
    Ui::atributeForm *ui;

    QSqlDatabase dbSystem;
    QSqlQuery query;



    QStandardItemModel *atributeModel = new QStandardItemModel();
```

```
    QStringListModel *typeModel = new QStringListModel();
    QStandardItem *item;

    QStringList verticalHeader;
    QStringList List;
    QStringList items;


};

#endif // ATRIBUTEFORM_H
```

Atributeform.cpp

```
#include "atributeform.h"
#include "qsqlerror.h"
#include "ui_atributeform.h"
#include <QSqlQuery>
#include <QMessageBox>
#include <QInputDialog>
#include <minwindef.h>
#include <QSpinBox>
#include "delegatespinbox.h"
#include <QtWidgets>

atributeForm::atributeForm(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::atributeForm)
{
    ui->setupUi(this);
```

```
//    int i=0;
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    query.prepare("SELECT name FROM atribute");
    query.exec();
    List.clear();
    while(query.next())
    {
        qDebug() << query.value("name").toString() << query.value(0).toString();
//        item = new QStandardItem(QString(query.value("name").toString()));
        List.append(query.value("name").toString());

//        ++i;
    }
    typeModel->setStringList(List);
    ui->listView->setModel(typeModel);

    connect(ui->listView->selectionModel(),
SIGNAL(currentRowChanged(QModelIndex,QModelIndex)),
            this, SLOT(rowChangedSlot(QModelIndex,QModelIndex)));
/*
    horizontalHeader.append("Властивість");
    horizontalHeader.append("Значення");
    horizontalHeader.append("Значення");
    horizontalHeader.append("Значення");
    horizontalHeader.append("Значення");
*/

//    atributeModel->setHorizontalHeaderLabels(verticalHeader);
```

```
//    ui->tableView->horizontalHeader()->setStretchLastSection(true);

//    ui->tableView->horizontalHeader()->setVisible(FALSE);
//    QStringList horizontalHeader;
//    horizontalHeader.append("    ");
//    atributeModel->setHorizontalHeaderLabels(horizontalHeader);
//    ui->tableView->setModel(atributeModel);
    connect(ui->tableView->selectionModel(),
SIGNAL(currentChanged(QModelIndex,QModelIndex)),
            this, SLOT(tableChangedSlot(QModelIndex,QModelIndex)));


    query.prepare("SELECT name FROM typeAtribute");
    query.exec();
    while(query.next())
    {
        qDebug() << query.value("name").toString() << query.value(0).toString();
//        item = new QStandardItem(QString(query.value("name").toString()));
        items.append(query.value("name").toString());
//        ++i;
    }

    verticalHeader.append("Довжина");
    verticalHeader.append("Знак");
    verticalHeader.append("Знаків після зап'ятої");
    verticalHeader.append("Суфікс");
    verticalHeader.append("Формат");

    delegateSpinBox *delegateSpinBox = new class delegateSpinBox(this);
```

```
    ui->tableView->setItemDelegateForRow(0, delegateSpinBox);
    ui->tableView->setItemDelegateForRow(2, delegateSpinBox);

    delegateComboBox *delegateComboBox = new class delegateComboBox(this);
    ui->tableView->setItemDelegateForRow(1, delegateComboBox);

/*
    ui->tableView->hideRow(0);
    ui->tableView->hideRow(1);
    ui->tableView->hideRow(2);
    ui->tableView->hideRow(3);
    ui->tableView->hideRow(4);
*/

}

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

void atributeForm::on_pushButton_add_clicked()
{
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    int listSize = List.size();

    bool ok;
    QString item = QInputDialog::getItem(this, tr("Вибор типу атрибута"),
                                         tr("Виберіть тип атрибута:"), items,
```

```
                                0, FALSE, &ok);
    if (ok && !item.isEmpty())
    {
/*
        qry.prepare("SELECT * FROM typeForm");
        qry.exec();
        int id = qry.size();
*/
        List.append("Новий атрибут");
        typeModel->setStringList(List);
        ui->listView->setModel(typeModel);

        query.prepare("INSERT INTO atribute (id, name, type) "
                      "VALUES (:id, :name, :type)");



        int index = items.indexOf(item);
        query.bindValue(":id", listSize);
        query.bindValue(":name", "Новий атрибут");
        query.bindValue(":type", index);
        query.exec();
        ui->listView->selectionModel()->setCurrentIndex( ui->listView->model()-
>index(listSize,0), QItemSelectionModel::SelectionFlag::Select );

        QStringList valueList, caseIndxList;

        query.exec("SELECT id FROM propAtribute");
        query.last();
        int i = query.value("id").toInt();
```

```
    if (i>0)
       i++;

    query.exec("SELECT value, type, caseIndx FROM properties WHERE type =
"+QString::number(index));
    while (query.next())
    {
       valueList.append(query.value("value").toString());
       caseIndxList.append(query.value("caseIndx").toString());
    }

    query.prepare("INSERT INTO propAtribute (id, idAtribute, idProperties,
value, caseIndx) "
                  "VALUES (?, ?, ?, ?, ?)");


    for(int j=0; valueList.size()>j; i++, j++)
    {
       query.bindValue(0, i);
       query.bindValue(1, listSize);
       query.bindValue(2, j);
       query.bindValue(3, valueList.at(j));
       query.bindValue(4, caseIndxList.at(j));


       if(!query.exec()){
          qDebug() << query.lastError().text();
       }
    }
```

```
    }
/*
    List.append("Новий атрибут");
    typeModel->setStringList(List);
    ui->listView->setModel(typeModel);

    query.prepare("INSERT INTO atribute (id, name) "
                  "VALUES (:id, :name)");




    query.bindValue(":id", listSize);
    query.bindValue(":name", "Новий атрибут");
    query.exec();
    ui->listView->selectionModel()->setCurrentIndex( ui->listView->model()-
>index(listSize,0), QItemSelectionModel::SelectionFlag::Select );
*/
}


void atributeForm::on_pushButton_delete_clicked()
{
    QMessageBox msgBox;
    msgBox.setWindowTitle("Попередження");
    msgBox.setText("Увага!");
    msgBox.setIcon(QMessageBox::Information);
    msgBox.setInformativeText("Ви впевнені видалити атрибут "+ui->listView-
>currentIndex().data().toString()+"?");
```

```
    msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
    msgBox.setDefaultButton(QMessageBox::Ok);
    int ret = msgBox.exec();

    switch (ret)
    {
        case QMessageBox::Save:
            return;
            break;
        case QMessageBox::Ok:
            break;
        default:
            return;
            break;
    }



}

void atributeForm::rowChangedSlot(const QModelIndex &curIndex, const
QModelIndex &prevIndex)
{
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);

    QSqlQuery qry("dbSystem", dbSystem);

    qDebug() << "ChangedSlot" << curIndex << prevIndex << curIndex.row() <<
prevIndex.row();
```

```
    int typeAtribute;
    int i=0;

    atributeModel->setColumnCount(1);

    atributeModel->setVerticalHeaderLabels(verticalHeader);

    ui->tableView->setModel(atributeModel);

//    delegateSpinBox spinBoxDelegate;
//    ui->tableView->setItemDelegateForRow(0, NULL);
//    ui->tableView->setItemDelegateForRow(0, defaultDelegate);
//    ui->tableView->setItemDelegateForRow(0, &spinBoxDelegate);
//    ui->tableView->setItemDelegateForRow(2, &spinBoxDelegate);

//    verticalHeader.clear();

//    atributeModel->clear();
//    atributeModel->setVerticalHeaderLabels(verticalHeader);

    query.prepare("SELECT * FROM atribute WHERE id =
"+QString::number(curIndex.row()));
    query.exec();
    query.next();
    typeAtribute = query.value("type").toInt();

//    query.prepare("SELECT * FROM properties WHERE type =
"+QString::number(typeAtribute)+" UNION "
//                  "SELECT * FROM propAtribute WHERE idAtribute =
"+QString::number(curIndex.row()));
```

```
    query.exec("SELECT * FROM properties WHERE type =
"+QString::number(typeAtribute));
    qry.exec("SELECT * FROM propAtribute WHERE idAtribute =
"+QString::number(curIndex.row()));
/*
    switch(qry.value("caseIndx").toInt()) {
    case 0:
        atributeModel->verticalHeaderItem(0)-
>setText(query.value("name").toString());
    case 1:

    case 2:

     default:

    }
*/


//    delegateSpinBox delegate;
    QModelIndex index;
    ui->tableView->edit(QModelIndex());
    ui->tableView->setModel(atributeModel);
    for (int j=0; j<atributeModel->rowCount(); j++)
    {
        ui->tableView->hideRow(j);
    }
```

```
//    ui->tableView->setItemDelegate(NULL);

    while (query.next() && qry.next())
    {
//        verticalHeader.append(query.value("name").toString());

 /*
        item = new QStandardItem(query.value("name").toString());
        item->setEditable(FALSE);
        atributeModel->setItem(i,0,item);
*/
        ui->tableView->showRow(query.value("id").toInt());
        switch(qry.value("caseIndx").toInt()) {
        case 1:
/*
                QSpinBox *spin = new QSpinBox();
                spin->setMinimum(-1);
                spin->setValue(qry.value("value").toInt());
                spin->setButtonSymbols(QAbstractSpinBox::NoButtons);
                atributeModel->setItem(i,1,spin);

                ui->tableView->setItemDelegateForRow(i, &delegate);
                index = atributeModel->index(i, 1, QModelIndex());
                atributeModel->setData(index, QVariant((i + 1) * (1 + 1)));
*/

//            ui->tableView->setItemDelegateForRow(0, &delegate);
            index = atributeModel->index(i, 0, QModelIndex());
            atributeModel->setData(index, qry.value("value").toInt());
```

```
            ui->tableView->openPersistentEditor(index);

            qDebug() << QString(qry.value("value").toString());
            break;
        case 2:

            break;
        case 3:

            break;
        default:
            item = new QStandardItem(QString(qry.value("value").toString()));
            qDebug() << QString(qry.value("value").toString());

    }
/*
        QStringList list = {"Чет/Ніч", "Чет/Ранок", "Чет/Вечір", "Нечет/Ніч",
"Нечет/Ранок", "Нечет/Вечір"};
            Personal_verticalHeader.append(" ");
            Personal_Setings_model-
>setVerticalHeaderLabels(Personal_verticalHeader);
            Personal_item = new QStandardItem(QString(""));
            Personal_Setings_model->setItem(Personal_Setings_model->rowCount()-
1, 0, Personal_item);
            QDateEdit *date = new QDateEdit();
            date->setDate(QDate::currentDate());
            date->setCalendarPopup(true);
            QComboBox *combo = new QComboBox();
            combo->addItems(list);
            combo->setCurrentIndex(0);
```

```
        QDoubleSpinBox *spin = new QDoubleSpinBox();
        spin->setMaximum(999999.99);
        spin->setButtonSymbols(QAbstractSpinBox::NoButtons);
        spin->setValue(0);
        ui->Table_Personal->setIndexWidget(Personal_Setings_model-
>index(Personal_Setings_model->rowCount()-1, 1), date);
        ui->Table_Personal->setIndexWidget(Personal_Setings_model-
>index(Personal_Setings_model->rowCount()-1, 2), combo);
        ui->Table_Personal->setIndexWidget(Personal_Setings_model-
>index(Personal_Setings_model->rowCount()-1, 3), spin);
        DATE.append(date);
        COMBO.append(combo);
        SPIN_salary.append(spin);
        ui->Table_Personal->update();


*/




//      item = new QStandardItem(QString(qry.value("value").toString()));
//      atributeModel->setItem(i,0,item);

//      atributeModel->verticalHeaderItem(i)-
>setText(query.value("name").toString());

      ++i;
      qDebug() << QString(qry.value("value").toString());
```

```
    }
//    atributeModel->setVerticalHeaderLabels(verticalHeader);
//    atributeModel->setColumnCount(1);
//    ui->tableView->update();



//    ui->tableView->setModel(atributeModel);
//    delegateSpinBox spinBoxDelegate;
//    ui->tableView->setItemDelegateForRow(0, &comboBoxDelegate);

 //   QModelIndex index1 = atributeModel->index(0, 0, QModelIndex());
 //   atributeModel->setData(index1, QVariant(5));
 //   qDebug() << atributeModel->data(index1) << index1;

//    ui->tableView->hideRow(1);
//    ui->tableView->hideRow(2);
//    ui->tableView->showRow(0);
//    ui->tableView->setItemDelegate(NULL);



}

void atributeForm::tableChangedSlot (const QModelIndex &curIndex, const
QModelIndex &prevIndex)
{
/*    if (prevIndex.row() != -1)
    {
        QModelIndex indx = ui->tableView->model()->index(prevIndex.row(),1);
//        ui->tableView->update();
```

```
//        item = new QStandardItem(atributeModel->itemFromIndex(prevIndex)-
>data().toString());


        query.prepare("UPDATE propAtribute SET value = :value WHERE
idAtribute = :idAtribute AND idProperties = :idProperties");


        query.bindValue(":value", indx.data().toString());
        query.bindValue(":idAtribute", QString::number(ui->listView-
>currentIndex().row()));
        query.bindValue(":idProperties", QString::number(prevIndex.row()));
        qDebug() << indx.data().toString() << QString::number(prevIndex.row()) <<
QString::number(ui->listView->currentIndex().row()) << prevIndex;
        if( !query.exec() )
        {
            qDebug() << "a" << query.lastError();
        }
    }


*/
}




void atributeForm::on_buttonBox_accepted()
{
```

```
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    int row=0;
    QModelIndex indx;
//        ui->tableView->update();
//        item = new QStandardItem(atributeModel->itemFromIndex(prevIndex)-
>data().toString());

    while(row < ui->tableView->model()->rowCount())
    {
        indx = ui->tableView->model()->index(row,1);

        query.prepare("UPDATE propAtribute SET value = :value WHERE
idAtribute = :idAtribute AND idProperties = :idProperties");



        query.bindValue(":value", indx.data().toString());
        query.bindValue(":idAtribute", QString::number(ui->listView-
>currentIndex().row()));
        query.bindValue(":idProperties", QString::number(row));
        qDebug() << indx.data().toString() << QString::number(row) <<
QString::number(ui->listView->currentIndex().row());
        if( !query.exec() )
        {
            qDebug() << "a" << query.lastError();
        }
        row++;
    }
}
```

```
void atributeForm::on_buttonBox_clicked(QAbstractButton *button)
{
   QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
   QSqlQuery query("dbSystem", dbSystem);
   if(button->text() == "Apply")
   {
      qDebug() << "Apply";

      int row=0;
      QModelIndex indx;
   //       ui->tableView->update();
   //       item = new QStandardItem(atributeModel->itemFromIndex(prevIndex)-
>data().toString());

      while(row < ui->tableView->model()->rowCount())
      {
         indx = ui->tableView->model()->index(row,1);

         query.prepare("UPDATE propAtribute SET value = :value WHERE
idAtribute = :idAtribute AND idProperties = :idProperties");


         query.bindValue(":value", indx.data().toString());
         query.bindValue(":idAtribute", QString::number(ui->listView-
>currentIndex().row()));
         query.bindValue(":idProperties", QString::number(row));
         qDebug() << indx.data().toString() << QString::number(row) <<
QString::number(ui->listView->currentIndex().row());
         if( !query.exec() )
         {
```

```
            qDebug() << "a" << query.lastError();
        }
        row++;
    }
    row = 0;
    while (row < ui->listView->model()->rowCount())
    {



        row++;
    }
  }



}
```

atributeform.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>atributeForm</class>
 <widget class="QDialog" name="atributeForm">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>516</width>
    <height>448</height>
   </rect>
  </property>
```

```
<property name="windowTitle">
 <string>Налаштування типів даних</string>
</property>
<property name="locale">
 <locale language="Ukrainian" country="Ukraine"/>
</property>
<widget class="QDialogButtonBox" name="buttonBox">
 <property name="geometry">
  <rect>
   <x>170</x>
   <y>410</y>
   <width>341</width>
   <height>31</height>
  </rect>
 </property>
 <property name="locale">
  <locale language="Ukrainian" country="Ukraine"/>
 </property>
 <property name="orientation">
  <enum>Qt::Horizontal</enum>
 </property>
 <property name="standardButtons">
  <set>QDialogButtonBox::Apply|QDialogButtonBox::Close</set>
 </property>
 <property name="centerButtons">
  <bool>false</bool>
 </property>
</widget>
<widget class="QTableView" name="tableView">
 <property name="geometry">
```

```
  <rect>
   <x>170</x>
   <y>10</y>
   <width>341</width>
   <height>401</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="pushButton_add">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>420</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Додати</string>
 </property>
</widget>
<widget class="QPushButton" name="pushButton_delete">
 <property name="geometry">
  <rect>
   <x>90</x>
   <y>420</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
```

```
 <property name="text">
  <string>Видалити</string>
 </property>
</widget>
<widget class="QListView" name="listView">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>10</y>
   <width>151</width>
   <height>401</height>
  </rect>
 </property>
</widget>
</widget>
<resources/>
<connections>
 <connection>
  <sender>buttonBox</sender>
  <signal>accepted()</signal>
  <receiver>atributeForm</receiver>
  <slot>accept()</slot>
  <hints>
   <hint type="sourcelabel">
    <x>248</x>
    <y>254</y>
   </hint>
   <hint type="destinationlabel">
    <x>157</x>
    <y>274</y>
```

```
   </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>atributeForm</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>316</x>
     <y>260</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

Додаток В.6

Лістинг модуля createdocform

Createdocform.h

```
#ifndef CREATEDOCFORM_H
#define CREATEDOCFORM_H

#include <QDialog>

#include <mainwindow.h>

#include "QStandardItemModel"
#include "QStandardItem"
#include <QSqlDatabase>
#include <QSqlTableModel>
#include <delegatecombobox.h>


namespace Ui {
class CreateDocForm;
}

class CreateDocForm : public QDialog
{
    Q_OBJECT

public:
    explicit CreateDocForm(QWidget *parent = nullptr);
    ~CreateDocForm();
    void toTable(QStandardItemModel *model, int table);
    void fromTable(QStandardItemModel *model, int table);

private slots:
```

```
    void on_katCreate_clicked();

    void on_katChange_clicked();

    void on_katDelete_clicked();

    void on_docCreate_clicked();

    void on_docChange_clicked();

    void on_docDelete_clicked();

    void on_katalog_currentIndexChanged();

    void on_document_currentIndexChanged();

    void on_addButton_clicked();

    void on_delButton_clicked();

    void on_buttonBox_accepted();

private:
    Ui::CreateDocForm *ui;

    QSqlDatabase dbSystem;

    QStandardItemModel *docModel = new QStandardItemModel();
    QStandardItemModel *katModel = new QStandardItemModel();
    QStandardItemModel *model = new QStandardItemModel();
```

```
    QStandardItemModel *model1 = new QStandardItemModel();
    QStandardItem *item;

    QStringList verticalHeader;
    QStringList horizontalHeader;

    int katI, docI;
    QList< QPair<QString, int> > data;
    QList< QPair<QString, int> > flag = { {"Hi", 0}, {"Так", 1}};
};

#endif // CREATEDOCFORM_H
```

Createdocform.cpp

```
#include "createdocform.h"
#include "ui_createdocform.h"

#include <QSqlQuery>
#include <QSqlTableModel>
#include <QtGui>
#include <QInputDialog>
#include <QMessageBox>
#include <QCheckBox>
#include <QRadioButton>
#include "dbconnect.h"
#include<QDebug>
#include <QSqlError>

typedef QList<QPair<QString, int> > List;

CreateDocForm::CreateDocForm(QWidget *parent) :
```

```
    QDialog(parent),
    ui(new Ui::CreateDocForm)
{
    ui->setupUi(this);
    int i=0;
    katI=0;
    //  Заголовки столбцов
     horizontalHeader.append("Ім'я поля");
     horizontalHeader.append("Тип поля");
     horizontalHeader.append("Властивість");
     horizontalHeader.append("Автозаповнювання");
     horizontalHeader.append("Головне поле");

     model->setHorizontalHeaderLabels(horizontalHeader);
     ui->tableView->setModel(model);
     model1->setHorizontalHeaderLabels(horizontalHeader);
     ui->tableView_2->setModel(model1);



     QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
     QSqlQuery query("dbSystem", dbSystem);

     query.prepare("SELECT * FROM atribute");
     query.exec();
     while(query.next())
     {
        data.append(qMakePair(query.value("name").toString(),i));
        ++i;
     }
```

```
    QStringList list;
    if (dbSystem.isOpen())
    {
       qDebug() << "1";
    }

    i=0;

    docI=query.exec("COUNT FROM form");
    query.prepare("SELECT * FROM typeForm");
    query.exec();
    while (query.next())
    {
       //list.join(query.value("name").toString());
       //ui->katalog->addItem(query.value(0).toString());
       qDebug() << query.value("name").toString() << query.value(0).toString();
       item = new QStandardItem(QString(query.value("name").toString()));
       katModel->setItem(i,item);
       ++i;
       ++katI;
    }
    ui->katalog->setModel(katModel);
   //ui->katalog->addItems(list);



    delegateComboBox *delegateComboBox = new class
delegateComboBox(this);
    ui->tableView->setItemDelegateForColumn(2, delegateComboBox);
```

```
    ui->tableView->setItemDelegateForColumn(3, delegateComboBox);
    ui->tableView->setItemDelegateForColumn(4, delegateComboBox);
}

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

void CreateDocForm::on_katCreate_clicked()
{
   bool ok;
   QString text = QInputDialog::getText(this, tr("Добавить каталог"),
                                        tr("Каталог:"), QLineEdit::Normal,
                                        "", &ok);
   if (ok && !text.isEmpty())
   {
      QSqlQuery query;
/*
      qry.prepare("SELECT * FROM typeForm");
      qry.exec();
      int id = qry.size();
*/
      query.prepare("INSERT INTO typeForm (id, name) "
                    "VALUES (:id, :name)");
```

```
        query.bindValue(":id", katI);
        query.bindValue(":name", text);
        query.exec();
        ui->katalog->addItem(text);
        ++katI;
    }
}


void CreateDocForm::on_katChange_clicked()
{

}


void CreateDocForm::on_katDelete_clicked()
{
    QMessageBox msgBox;
    msgBox.setWindowTitle("Попередження");
    msgBox.setText("Увага!");
    msgBox.setIcon(QMessageBox::Information);
    msgBox.setInformativeText("Ви впевнені видалити каталог "+ui->katalog-
>currentText()+"?");
    msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
    msgBox.setDefaultButton(QMessageBox::Ok);
    int ret = msgBox.exec();

    switch (ret)
    {
        case QMessageBox::Save:
```

```
            return;
            break;
        case QMessageBox::Ok:
            break;
        default:
            return;
            break;
    }

    QStringList list;
    QSqlQuery query, query1, query2, query3;
    query1.prepare("DELETE FROM poles WHERE idForm = (SELECT
nameForm FROM form WHERE typeForm = "+ui->katalog->currentText()+")");
    query2.prepare("DELETE FROM form WHERE typeForm = "+ui->katalog-
>currentText());
    query3.prepare("DELETE FROM typeForm WHERE name = "+ui->katalog-
>currentText());
    query1.exec();
    query2.exec();
    query3.exec();

    query.prepare("SELECT name FROM typeForm)");
    query.exec();

    while (query.next())
    {
        list.join(query.value("name").toString());
    }
    ui->katalog->clear();
    ui->katalog->addItems(list);
```

```
}


void CreateDocForm::on_docCreate_clicked()
{
   bool ok;
   QString text = QInputDialog::getText(this, tr("Добавить документ"),
                                        tr("Документ:"), QLineEdit::Normal,
                                        "", &ok);
   if (ok && !text.isEmpty())
   {
      QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
      QSqlQuery query("dbSystem", dbSystem);



      query.prepare("INSERT INTO form (id, nameForm, typeForm) "
                    "VALUES (:id, :nameForm, :typeForm)");



      query.bindValue(":id", docI);
      query.bindValue(":nameForm", text);
      query.bindValue(":typeForm", ui->katalog->currentIndex());
      query.exec();
      ++docI;
   }
}
```

```
void CreateDocForm::on_docChange_clicked()
{

}


void CreateDocForm::on_docDelete_clicked()
{
    QMessageBox msgBox;
    msgBox.setWindowTitle("Попередження");
    msgBox.setText("Увага!");
    msgBox.setIcon(QMessageBox::Information);
    msgBox.setInformativeText("Ви впевнені видалити документ "+ui-
>document->currentText()+"?");
    msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
    msgBox.setDefaultButton(QMessageBox::Ok);
    int ret = msgBox.exec();

    switch (ret)
    {
        case QMessageBox::Save:
            return;
            break;
        case QMessageBox::Ok:
            break;
        default:
            return;
            break;
    }
```

```
    QSqlQuery query1, query2;
    query1.prepare("DELETE FROM poles WHERE idForm = "+ui->document-
>currentText());
    query2.prepare("DELETE FROM form WHERE nameForm = "+ui->document-
>currentText());
    query1.exec();
    query2.exec();
}


void CreateDocForm::on_katalog_currentIndexChanged()
{
    ui->document->clear();
    int i=0;
    QStringList list;
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    query.prepare("SELECT * FROM form WHERE typeForm =
"+QString::number(ui->katalog->currentIndex()));
    query.exec();
    qDebug() << QString::number(ui->katalog->currentIndex()) << ui->katalog-
>currentIndex();
    while (query.next())
    {
 //       list.join(query.value("nameForm").toString());
        item = new QStandardItem(QString(query.value("nameForm").toString()));
        docModel->setItem(i,item);
        qDebug() << query.value("nameForm").toString() <<
query.value(0).toString();
        ++i;
```

```
    }
    ui->document->setModel(docModel);


}


void CreateDocForm::on_document_currentIndexChanged()
{
    toTable(model, 0);
    toTable(model1, 1);
}


void CreateDocForm::on_addButton_clicked()
{
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);

//    verticalHeader.append(" ");
//    model->setVerticalHeaderLabels(verticalHeader);

    item = new QStandardItem(QString(""));
    model->setItem(model->rowCount()-1, 0, item);

    item = new QStandardItem(QString(""));
    model->setItem(model->rowCount()-1, 1, item);

//    item = new QStandardItem(QString("QString"));
//    model->setItem(model->rowCount()-1, 2, item);
```

```
//    item = new QStandardItem(QString(""));
//    model->setItem(model->rowCount()-1, 3, item);
//    ui->tableView->edit(QModelIndex());



    QStringList listAtribute;
    query.prepare("SELECT name FROM atribute");
    query.exec();
    listAtribute.clear();
    while(query.next())
    {
        qDebug() << query.value("name").toString() << query.value(0).toString();
        listAtribute.append(query.value("name").toString());

    }

/*
    QComboBox *combo = new QComboBox();
    combo->addItems(listAtribute);



    QCheckBox *check = new QCheckBox();
    QRadioButton *radio = new QRadioButton();

    ui->tableView->setIndexWidget(model->index(model->rowCount()-1, 3),
combo);
```

```
    ui->tableView->setIndexWidget(model->index(model->rowCount()-1, 3),
check);

    ui->tableView->setIndexWidget(model->index(model->rowCount()-1, 4),
radio);
*/
/*
    QModelIndex index;
    index = model->index(model->rowCount()-1, 2, QModelIndex());
    model->setData(index, listAtribute);
    ui->tableView->openPersistentEditor(index);
    qDebug() << listAtribute << model->rowCount()-1 << index;
*/

    ui->tableView->update();
}


void CreateDocForm::on_delButton_clicked()
{
    verticalHeader.removeAt(ui->tableView->currentIndex().row());
    model->setVerticalHeaderLabels(verticalHeader);
}


void CreateDocForm::on_buttonBox_accepted()
{
    fromTable(model, 0);
    fromTable(model, 1);
}
```

```
void CreateDocForm::toTable(QStandardItemModel *model, int table)
{

    model->clear();
    model->setHorizontalHeaderLabels(horizontalHeader);
    int i=0;
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    query.prepare("SELECT * FROM poles INNER JOIN form ON idForm =
form.id INNER JOIN typeForm ON form.typeForm = typeForm.id WHERE
idForm = "+QString::number(ui->document->currentIndex())+" AND model =
"+QString::number(table));
    query.exec();
/*
    QSqlQuery qry("dbSystem", dbSystem);
    qry.prepare("SELECT * FROM atribute");
    qry.exec();
*/
    while (query.next())
    {
 //       list.join(query.value("nameForm").toString());
        item = new QStandardItem(QString(query.value("name").toString()));
        model->setItem(i,0,item);
        item = new QStandardItem(QString(query.value("atribute").toString()));
        model->setItem(i,1,item);


        item = new QStandardItem();
        item->setData(data[query.value("properties").toInt()].first, Qt::DisplayRole);
```

```
        item->setData(data[query.value("properties").toInt()].second);
        item->setData( QVariant::fromValue(data), Qt::UserRole +2);

        model->setItem(i, 2, item);
        QModelIndex index = model->index(i, 2, QModelIndex());
        ui->tableView->openPersistentEditor(index);

        item = new QStandardItem();
        item->setData(flag[query.value("autoFill").toInt()].first, Qt::DisplayRole);
        item->setData(flag[query.value("autoFill").toInt()].second);
        item->setData( QVariant::fromValue(flag), Qt::UserRole +2);


        model->setItem(i, 3, item);
        QModelIndex index1 = model->index(i, 3, QModelIndex());
        ui->tableView->openPersistentEditor(index1);

        item = new QStandardItem();
        item->setData(flag[query.value("flag").toInt()].first, Qt::DisplayRole);
        item->setData(flag[query.value("flag").toInt()].second);
        item->setData( QVariant::fromValue(flag), Qt::UserRole +2);
        model->setItem(i, 4, item);
        QModelIndex index2 = model->index(i, 4, QModelIndex());
        ui->tableView->openPersistentEditor(index2);

        qDebug() << query.value("name").toString() << query.value(2).toString();
        ++i;
    }
/*
    verticalHeader.clear();
```

```
    for(int i = 0; i < model->rowCount(); i++)
        verticalHeader.append(model->takeVerticalHeaderItem(i)->text());
*/
}

void CreateDocForm::fromTable(QStandardItemModel *model, int table)
{
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);

    for (int i = 0; i < model->rowCount(); i++)
    {
        query.prepare("INSERT INTO poles (name, idForm, atribute, properties,
autoFill, flag, table) "
                      "VALUES (:name, :idForm, :atribute, :properties, :autoFill, :flag,
:table)");
        query.bindValue(":name", model->item(i,0)->text());
        query.bindValue(":idForm", ui->document->currentIndex());
        query.bindValue(":atribute", model->item(i,1)->text());
        query.bindValue(":properties", model->item(i,2)->text().toInt());
        query.bindValue(":autoFill", model->item(i,3)->text().toInt());
        query.bindValue(":flag", model->item(i,4)->text().toInt());
        query.bindValue(":model", table);
        query.exec();
    }
}
```

createdocform.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>CreateDocForm</class>
```

```
<widget class="QDialog" name="CreateDocForm">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>751</width>
   <height>677</height>
  </rect>
 </property>
 <property name="windowTitle">
  <string>Створювання документів</string>
 </property>
 <widget class="QDialogButtonBox" name="buttonBox">
  <property name="geometry">
   <rect>
    <x>390</x>
    <y>640</y>
    <width>341</width>
    <height>32</height>
   </rect>
  </property>
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
  <property name="standardButtons">
   <set>QDialogButtonBox::Apply|QDialogButtonBox::Cancel</set>
  </property>
 </widget>
 <widget class="QGroupBox" name="katGroup">
  <property name="geometry">
```

```
 <rect>
  <x>10</x>
  <y>10</y>
  <width>721</width>
  <height>71</height>
 </rect>
</property>
<property name="title">
 <string>Категорія документів</string>
</property>
<widget class="QComboBox" name="katalog">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>30</y>
   <width>311</width>
   <height>22</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="katCreate">
 <property name="geometry">
  <rect>
   <x>370</x>
   <y>30</y>
   <width>101</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
```

```
  <string>Новий</string>
 </property>
</widget>
<widget class="QPushButton" name="katChange">
 <property name="geometry">
  <rect>
   <x>510</x>
   <y>30</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Редагувати</string>
 </property>
</widget>
<widget class="QPushButton" name="katDelete">
 <property name="geometry">
  <rect>
   <x>620</x>
   <y>30</y>
   <width>81</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Видалити</string>
 </property>
</widget>
</widget>
```

```
<widget class="QGroupBox" name="docGroup">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>80</y>
   <width>721</width>
   <height>71</height>
  </rect>
 </property>
 <property name="title">
  <string>Документ</string>
 </property>
 <widget class="QPushButton" name="docCreate">
  <property name="geometry">
   <rect>
    <x>370</x>
    <y>30</y>
    <width>101</width>
    <height>24</height>
   </rect>
  </property>
  <property name="text">
   <string>Новий</string>
  </property>
 </widget>
 <widget class="QPushButton" name="docChange">
  <property name="geometry">
   <rect>
    <x>510</x>
    <y>30</y>
```

```
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Редагувати</string>
 </property>
</widget>
<widget class="QPushButton" name="docDelete">
 <property name="geometry">
  <rect>
   <x>620</x>
   <y>30</y>
   <width>81</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Видалити</string>
 </property>
</widget>
<widget class="QComboBox" name="document">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>30</y>
   <width>311</width>
   <height>22</height>
  </rect>
 </property>
```

```
 </widget>
</widget>
<widget class="QGroupBox" name="groupBox">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>149</y>
   <width>721</width>
   <height>491</height>
  </rect>
 </property>
 <property name="title">
  <string/>
 </property>
 <widget class="QGroupBox" name="groupBox_2">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>721</width>
    <height>231</height>
   </rect>
  </property>
  <property name="title">
   <string>Унікальні параметри </string>
  </property>
  <widget class="QTableView" name="tableView">
   <property name="geometry">
    <rect>
     <x>10</x>
```

```
   <y>20</y>
   <width>701</width>
   <height>161</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="addButton">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>190</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Додати</string>
 </property>
</widget>
<widget class="QPushButton" name="delButton">
 <property name="geometry">
  <rect>
   <x>90</x>
   <y>190</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Видалити</string>
```

```
  </property>
 </widget>
</widget>
<widget class="QGroupBox" name="groupBox_3">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>220</y>
   <width>721</width>
   <height>271</height>
  </rect>
 </property>
 <property name="title">
  <string>Повторювальні параметри</string>
 </property>
 <widget class="QTableView" name="tableView_2">
  <property name="geometry">
   <rect>
    <x>10</x>
    <y>20</y>
    <width>701</width>
    <height>211</height>
   </rect>
  </property>
 </widget>
 <widget class="QPushButton" name="addButton_2">
  <property name="geometry">
   <rect>
    <x>10</x>
    <y>240</y>
```

```
      <width>75</width>
      <height>24</height>
     </rect>
    </property>
    <property name="text">
     <string>Додати</string>
    </property>
   </widget>
   <widget class="QPushButton" name="delButton_2">
    <property name="geometry">
     <rect>
      <x>100</x>
      <y>240</y>
      <width>75</width>
      <height>24</height>
     </rect>
    </property>
    <property name="text">
     <string>Видалити</string>
    </property>
   </widget>
  </widget>
 </widget>
</widget>
<resources/>
<connections>
 <connection>
  <sender>buttonBox</sender>
  <signal>accepted()</signal>
  <receiver>CreateDocForm</receiver>
```

```
  <slot>accept()</slot>
  <hints>
   <hint type="sourcelabel">
    <x>248</x>
    <y>254</y>
   </hint>
   <hint type="destinationlabel">
    <x>157</x>
    <y>274</y>
   </hint>
  </hints>
 </connection>
 <connection>
  <sender>buttonBox</sender>
  <signal>rejected()</signal>
  <receiver>CreateDocForm</receiver>
  <slot>reject()</slot>
  <hints>
   <hint type="sourcelabel">
    <x>316</x>
    <y>260</y>
   </hint>
   <hint type="destinationlabel">
    <x>286</x>
    <y>274</y>
   </hint>
  </hints>
 </connection>
</connections>
</ui>
```

Додаток В.7

Лістинг модуля filldocform

Filldocform.h

```
#ifndef FILLDOCFORM_H
#define FILLDOCFORM_H

#include <QDialog>

#include <mainwindow.h>

#include "QStandardItemModel"
#include "QStandardItem"
#include <QSqlDatabase>
#include <QSqlTableModel>
#include <delegatebutton.h>
#include <delegatecombobox.h>
#include <delegatedate.h>
#include <delegatedoublespinbox.h>
#include <delegatespinbox.h>



namespace Ui {
class fillDocForm;
}

class fillDocForm : public QDialog
{
    Q_OBJECT

public:
    explicit fillDocForm(QWidget *parent = nullptr);
```

```
    ~fillDocForm();

private slots:
    void on_katalog_currentIndexChanged();

    void on_document_currentIndexChanged();

    void on_buttonBox_accepted();

    void on_fileSelectButton_clicked();

private:
    Ui::fillDocForm *ui;

    QSqlDatabase dbSystem;

    QStandardItemModel *docModel = new QStandardItemModel();
    QStandardItemModel *katModel = new QStandardItemModel();
    QStandardItemModel *model = new QStandardItemModel();
    QStandardItemModel *model1 = new QStandardItemModel();
    QStandardItem *item;

    QStringList verticalHeader;
    QStringList verticalHeader1;
    QStringList horizontalHeader;
    QStringList horizontalHeader1;
    QStringList List;
    QStringList List1;
    QString documentDir;
```

```
    delegateSpinBox *delegateSpinBox = new class delegateSpinBox(this);
    delegateDoubleSpinBox *delegateDoubleSpinBox = new class
delegateDoubleSpinBox(this);
    delegateComboBox *delegateComboBox = new class delegateComboBox(this);
    delegateDate *delegateDate = new class delegateDate(this);
};

#endif // FILLDOCFORM_H
```

Filldocform.cpp

```
#include "filldocform.h"
#include "ui_filldocform.h"

#include<QDebug>
#include <QSqlError>
#include <QSqlQuery>
#include <QSqlTableModel>
#include <QtGui>
#include "dbconnect.h"
#include <QAxObject>
#include <QFile>
#include <QFileDialog>

typedef QList<QPair<QString, int> > List;

fillDocForm::fillDocForm(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::fillDocForm)
{
    ui->setupUi(this);
    int i=0;
```

```
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    dbSystem.open();
    QSqlQuery query("dbSystem", dbSystem);
    query.prepare("SELECT name FROM typeForm");
    query.exec();
    while(query.next())
    {
        qDebug() << query.value("name").toString() << query.value(0).toString();
        item = new QStandardItem(QString(query.value("name").toString()));
        katModel->setItem(i,item);
        ++i;
    }

    ui->katalog->setModel(katModel);



//    dbSystem.close();
}

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

void fillDocForm::on_katalog_currentIndexChanged()
{
    ui->document->clear();
    int i=0;
    QStringList list;
```

```
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    query.prepare("SELECT * FROM form WHERE typeForm =
"+QString::number(ui->katalog->currentIndex()));
    query.exec();
    qDebug() << "1";
    while (query.next())
    {
 //       list.join(query.value("nameForm").toString());
        item = new QStandardItem(QString(query.value("nameForm").toString()));
        docModel->setItem(i,item);
        qDebug() << query.value("nameForm").toString() <<
query.value(0).toString();
        ++i;
    }
    ui->document->setModel(docModel);

}



void fillDocForm::on_document_currentIndexChanged()
{
    model->clear();
    model1->clear();
    verticalHeader.clear();
    verticalHeader1.clear();
    horizontalHeader.clear();
    horizontalHeader.append("1");
    horizontalHeader1.clear();
    horizontalHeader1.append("1");
```

```
    model->setHorizontalHeaderLabels(horizontalHeader);
    model1->setHorizontalHeaderLabels(horizontalHeader);
    List.clear();
    List1.clear();



    int i=0;
    QModelIndex index;
    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    QSqlQuery qry("dbSystem", dbSystem);
    QSqlQuery qrySup("dbSystem", dbSystem);
    query.prepare("SELECT * FROM poles INNER JOIN form ON idForm =
form.id INNER JOIN typeForm ON form.typeForm = typeForm.id WHERE
idForm = "+QString::number(ui->document->currentIndex())+" AND model =
"+QString::number(0));
    query.exec();

    while (query.next())
    {
        verticalHeader.append(query.value("name").toString());
        List.append(query.value("atribute").toString());
//        list.join(query.value("nameForm").toString());

        qry.prepare("SELECT * FROM propAtribute WHERE idAtribute =
"+QString::number(query.value("poles.properties").toInt()));
//        qDebug() << QString::number(query.value("poles.properties").toInt()) <<
QString::number(qry.value("idAtribute").toInt());
        qry.exec();
```

```
        qry.first();
        qDebug() << qry.value("idType").toInt() <<
query.value("poles.properties").toInt();
        int fl = qry.value("idType").toInt();
        switch(fl) {
        case 1:

/*
            index = atributeModel->index(i, 0, QModelIndex());
            atributeModel->setData(index, qry.value("value").toInt());
            ui->tableView->openPersistentEditor(index);
*/

            qrySup.prepare("SELECT * FROM propAtribute WHERE idAtribute
="+QString::number(qry.value("id").toInt())+" AND idProperties = 1");
            qrySup.exec();

            if (qrySup.value("value") == 0)
               ui->tableView->setItemDelegateForRow(i, delegateSpinBox);
            else
               ui->tableView->setItemDelegateForRow(i, delegateDoubleSpinBox);


            index = model->index(i, 0, QModelIndex());
            ui->tableView->openPersistentEditor(index);

            qDebug() << QString(qry.value("idType").toString());
            break;

        case 2:
```

```
            item = new
QStandardItem(QDate::currentDate().toString("dd.MM.yyyy"));
            model->setItem(i, 0,item);



            ui->tableView->setItemDelegateForRow(i, delegateDate);

            index = model->index(i, 0, QModelIndex());
            ui->tableView->openPersistentEditor(index);

            qDebug() << QString(qry.value("idType").toString());
            break;

        case 3:



            qDebug() << QString(qry.value("idType").toString());
            break;
        default:

            item = new QStandardItem("");
            model->setItem(i, 0, item);
//            item = new QStandardItem(QString(qry.value("value").toString()));
            qDebug() << QString(qry.value("idType").toString());

        }
```

```
        ++i;
    }



    model->setVerticalHeaderLabels(verticalHeader);
    ui->tableView->setModel(model);
}


void fillDocForm::on_buttonBox_accepted()
{
    if(documentDir.length()!=0) {
    qDebug() << QString( QCoreApplication::applicationDirPath() +
"/test/test.docx" );
    QString documentDirNew = QCoreApplication::applicationDirPath() +
"/test/res/"+ui->document->currentText().replace(" ", "-")+"-
"+QDateTime::currentDateTime().toString("dd-MM-yyyy-HH-mm-ss")+".docx";
    QFile::copy(documentDir, documentDirNew);

    QSqlDatabase dbSystem = QSqlDatabase::database("dbSystem");
    QSqlQuery query("dbSystem", dbSystem);
    QSqlQuery qry("dbSystem", dbSystem);
    QSqlQuery qrySup("dbSystem", dbSystem);

    auto wApp = new QAxObject("Word.Application");
    auto docs = wApp->querySubObject("Documents");
    auto doc  = docs->querySubObject("Open(QString)",documentDirNew);
    if (nullptr == doc)
        qFatal("File not found");
```

```
    auto active  = wApp->querySubObject("ActiveDocument()");
    auto content = active->querySubObject("Content()");
    auto find    = content->querySubObject("Find");

    QString sOld;
    QString sNew;
    QStringList prop;
    QDate date;

    bool bMatchCase      = true;
    bool bMatchWholeWord = true;
    bool bMatchWildCards = false;
    bool bReplaceAll     = true;

    query.prepare("SELECT * FROM poles INNER JOIN form ON idForm =
form.id INNER JOIN typeForm ON form.typeForm = typeForm.id WHERE
idForm = "+QString::number(ui->document->currentIndex())+" AND model =
"+QString::number(0));
    query.exec();


    for(int i = 0; i < model->rowCount(); i++)
    {
        query.next();

        sOld = List.value(i);

        qry.exec("SELECT * FROM propAtribute WHERE idAtribute =
"+QString::number(query.value("properties").toInt()));
```

```
//      qry.exec();
        qry.first();
        int fl = qry.value("idType").toInt();
        prop.clear();
        switch(fl) {
        case 1:

/*
            index = atributeModel->index(i, 0, QModelIndex());
            atributeModel->setData(index, qry.value("value").toInt());
            ui->tableView->openPersistentEditor(index);
*/
//          qry.next();

            qrySup.prepare("SELECT * FROM propAtribute WHERE idAtribute
="+QString::number(qry.value("id").toInt())+" AND idProperties = 0");
            qrySup.exec();
            qrySup.first();



            prop.append(qrySup.value("value").toString());

            qrySup.prepare("SELECT * FROM propAtribute WHERE idAtribute
="+QString::number(qry.value("id").toInt())+" AND idProperties = 2");
            qrySup.exec();
            qrySup.first();

            prop.append(qrySup.value("value").toString());

            sNew = model->item(i,0)->text();
```

```
            sNew = prop.at(0) + sNew;
            sNew = sNew + prop.at(1);



            qDebug() << QString(qry.value("idType").toString());
            break;

        case 2:

//            qry.next();
            sNew = model->item(i,0)->text();
            date = QDate::fromString(sNew, "dd.MM.yyyy");
            sNew = date.toString(qry.value("value").toString());



            qDebug() << QString(qry.value("idType").toString()) <<
qry.value("value").toString();
            break;

        case 3:


            qDebug() << QString(qry.value("idType").toString());
            break;
        default:
```

```
            sNew = model->item(i,0)->text();
        qDebug() << QString(qry.value("idType").toString());

    }



    qDebug() << sOld << sNew;
    QVariantList vl = { sOld, bMatchCase, bMatchWholeWord,
bMatchWildCards, false, false, true, 1, false, sNew, bReplaceAll ? "2" : "1" };
    find-
>dynamicCall("Execute(QString,bool,bool,bool,bool,bool,bool,int,bool,QString,int
)",vl);
  }


  bReplaceAll = false;

  for(int i = 0; i < model1->columnCount(); i++)
  {
    for(int j = 0; j < model1->rowCount(); j++)
    {
      sOld = List1.value(j);
      sNew = model1->item(j, i) ->text();
      qDebug() << sOld << sNew;
      QVariantList vl = { sOld, bMatchCase, bMatchWholeWord,
bMatchWildCards, false, false, true, 1, false, sNew, bReplaceAll ? "2" : "1" };
      find-
>dynamicCall("Execute(QString,bool,bool,bool,bool,bool,bool,int,bool,QString,int
)",vl);
```

```
        }
    }

    wApp->setProperty("Visible", true);
    delete find;
    delete content;
    delete active;
    delete doc;
    delete docs;
    delete wApp;
    } else qDebug() << "No document";
}


void fillDocForm::on_fileSelectButton_clicked()
{
    documentDir = QFileDialog::getOpenFileName(0, "Виберіть документ-шаблон", "", "*.doc *.docx");
    qDebug() << documentDir << QCoreApplication::applicationDirPath() + "/test/res/"+ui->document->currentText()+"-"+QDateTime::currentDateTime().toString(Qt::ISODate)+".docx";
}
```

filldocform.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>fillDocForm</class>
 <widget class="QDialog" name="fillDocForm">
  <property name="geometry">
   <rect>
```

```
  <x>0</x>
  <y>0</y>
  <width>730</width>
  <height>719</height>
 </rect>
</property>
<property name="windowTitle">
 <string>Заповнення документів</string>
</property>
<widget class="QGroupBox" name="katGroup">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>10</y>
   <width>711</width>
   <height>71</height>
  </rect>
 </property>
 <property name="title">
  <string>Категория документов</string>
 </property>
 <widget class="QComboBox" name="katalog">
  <property name="geometry">
   <rect>
    <x>20</x>
    <y>30</y>
    <width>671</width>
    <height>22</height>
   </rect>
  </property>
```

```
 </widget>
</widget>
<widget class="QGroupBox" name="docGroup">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>80</y>
   <width>711</width>
   <height>71</height>
  </rect>
 </property>
 <property name="title">
  <string>Документ</string>
 </property>
 <widget class="QComboBox" name="document">
  <property name="geometry">
   <rect>
    <x>360</x>
    <y>30</y>
    <width>331</width>
    <height>22</height>
   </rect>
  </property>
 </widget>
 <widget class="QPushButton" name="fileSelectButton">
  <property name="geometry">
   <rect>
    <x>20</x>
    <y>30</y>
    <width>331</width>
```

```
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Вибрати документ-шаблон</string>
 </property>
</widget>
</widget>
<widget class="QDialogButtonBox" name="buttonBox">
 <property name="geometry">
  <rect>
   <x>565</x>
   <y>690</y>
   <width>161</width>
   <height>24</height>
  </rect>
 </property>
 <property name="standardButtons">
  <set>QDialogButtonBox::Close|QDialogButtonBox::Yes</set>
 </property>
</widget>
<widget class="QGroupBox" name="groupBox">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>410</y>
   <width>711</width>
   <height>271</height>
  </rect>
 </property>
```

```
<property name="title">
 <string>Повторювальні параметри</string>
</property>
<widget class="QTableView" name="tableView_2">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>20</y>
   <width>671</width>
   <height>241</height>
  </rect>
 </property>
</widget>
</widget>
<widget class="QGroupBox" name="groupBox_2">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>140</y>
   <width>711</width>
   <height>281</height>
  </rect>
 </property>
 <property name="title">
  <string>Унікальні параметри</string>
 </property>
 <widget class="QTableView" name="tableView">
  <property name="geometry">
   <rect>
    <x>20</x>
```

```
      <y>20</y>
      <width>671</width>
      <height>241</height>
     </rect>
    </property>
   </widget>
  </widget>
  <zorder>katGroup</zorder>
  <zorder>docGroup</zorder>
  <zorder>buttonBox</zorder>
  <zorder>groupBox_2</zorder>
  <zorder>groupBox</zorder>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Додаток В.8

Лістинг модуля delegate

delegate.pri

```
# -------------------------------------------------------
# operate objects
# -------------------------------------------------------
# Header files
HEADERS += \
    $$PWD/delegatebutton.h \
    $$PWD/delegatecombobox.h \
    $$PWD/delegatedate.h \
    $$PWD/delegatedoublespinbox.h \
    $$PWD/delegatespinbox.h

# Source files
SOURCES += \
    $$PWD/delegatebutton.cpp \
    $$PWD/delegatecombobox.cpp \
    $$PWD/delegatedate.cpp \
    $$PWD/delegatedoublespinbox.cpp \
    $$PWD/delegatespinbox.cpp
```

delegatebutton.h

```
#ifndef DELEGATEBUTTON_H
#define DELEGATEBUTTON_H

#include <QItemDelegate>
#include <QPushButton>


class delegateButton    :   public QItemDelegate
```

```
{
   Q_OBJECT
public:
   delegateButton(QObject *parent = 0);
   virtual ~delegateButton(){};

   QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem
&option,
                         const QModelIndex &index) const override;

   void setEditorData(QWidget *editor, const QModelIndex &index) const
override;
   void setModelData(QWidget *editor, QAbstractItemModel *model,
                     const QModelIndex &index) const override;

   void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem
&option,
                             const QModelIndex &index) const override;
};

#endif // DELEGATEBUTTON_H
```

delegatebutton.cpp

```
#include "delegatebutton.h"

#include <QPushButton>
#include <QtGui>
#include <QDialog>

delegateButton::delegateButton(QObject *parent)
```

```
    : QItemDelegate(parent)
{
}

QWidget *delegateButton::createEditor(QWidget *parent,
                                      const QStyleOptionViewItem &/* option */,
                                      const QModelIndex &/* index */) const
{

    QPushButton  *editor = new QPushButton(parent);

    return editor;
}

void delegateButton::setEditorData(QWidget *editor,
                                   const QModelIndex &index) const
{
    int value = index.model()->data(index, Qt::EditRole).toInt();

    QPushButton *pushButton = static_cast<QPushButton*>(editor);


    pushButton->setText(QString::number(value));
}

void delegateButton::setModelData(QWidget *editor, QAbstractItemModel
*model,
                                  const QModelIndex &index) const
{
```

```
    QString value = "a";
    model->setData(index, value, Qt::EditRole);
}

void delegateButton::updateEditorGeometry(QWidget *editor,
                                          const QStyleOptionViewItem &option,
                                          const QModelIndex &/* index */) const
{
    editor->setGeometry(option.rect);
}
```

delegatespinbox.h

```
#ifndef DELEGATESPINBOX_H
#define DELEGATESPINBOX_H

#include <QItemDelegate>
#include <QSpinBox>

class delegateSpinBox  : public QItemDelegate
{
    Q_OBJECT
public:
    delegateSpinBox(QObject *parent = nullptr);
    virtual ~delegateSpinBox(){};
```

```
    QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem
&option,
                          const QModelIndex &index) const override;

    void setEditorData(QWidget *editor, const QModelIndex &index) const
override;
    void setModelData(QWidget *editor, QAbstractItemModel *model,
                      const QModelIndex &index) const override;

    void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem
&option,
                              const QModelIndex &index) const override;
};

#endif // DELEGATESPINBOX_H
```

delegatespinbox.cpp

```
#include "delegatespinbox.h"
#include <QSpinBox>

delegateSpinBox::delegateSpinBox(QObject *parent)
    : QItemDelegate(parent)
{

}

QWidget *delegateSpinBox::createEditor(QWidget *parent,
                                       const QStyleOptionViewItem &/* option */,
                                       const QModelIndex &/* index */) const
{
```

```
    QSpinBox *editor = new QSpinBox(parent);
    editor->setFrame(false);
    editor->setMaximum(999999);

    return editor;
}

void delegateSpinBox::setEditorData(QWidget *editor,
                                    const QModelIndex &index) const
{
    int value = index.model()->data(index, Qt::EditRole).toInt();

    QSpinBox *spinBox = static_cast<QSpinBox*>(editor);
    spinBox->setValue(value);
}

void delegateSpinBox::setModelData(QWidget *editor, QAbstractItemModel
*model,
                                   const QModelIndex &index) const
{
    QSpinBox *spinBox = static_cast<QSpinBox*>(editor);
    spinBox->interpretText();
    int value = spinBox->value();

    model->setData(index, value, Qt::EditRole);
}

void delegateSpinBox::updateEditorGeometry(QWidget *editor,
                                           const QStyleOptionViewItem &option,
```

```
                                const QModelIndex &/* index */) const
{
   editor->setGeometry(option.rect);
}
                                delegatedoublecombobox.h
#ifndef DELEGATEDOUBLESPINBOX_H
#define DELEGATEDOUBLESPINBOX_H

#include <QItemDelegate>
#include <QSpinBox>

class delegateDoubleSpinBox : public QItemDelegate
{
   Q_OBJECT
public:
   delegateDoubleSpinBox(QObject *parent = nullptr);
   virtual ~delegateDoubleSpinBox(){};

   QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem
&option,
                         const QModelIndex &index) const override;

   void setEditorData(QWidget *editor, const QModelIndex &index) const
override;
   void setModelData(QWidget *editor, QAbstractItemModel *model,
                     const QModelIndex &index) const override;

   void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem
&option,
```

```
                    const QModelIndex &index) const override;
};

#endif // DELEGATEDOUBLESPINBOX_H
```

delegateboublecombobox.cpp

```
#include "delegatedoublespinbox.h"

delegateDoubleSpinBox::delegateDoubleSpinBox(QObject *parent)
   : QItemDelegate(parent)
{
}

QWidget *delegateDoubleSpinBox::createEditor(QWidget *parent,
                            const QStyleOptionViewItem &/* option */,
                            const QModelIndex &/* index */) const
{
   QDoubleSpinBox *editor = new QDoubleSpinBox(parent);
   editor->setFrame(false);
   editor->setMaximum(9999999);

   return editor;
}

void delegateDoubleSpinBox::setEditorData(QWidget *editor,
                           const QModelIndex &index) const
{
   double value = index.model()->data(index, Qt::EditRole).toInt();

    QDoubleSpinBox *spinBox = static_cast<QDoubleSpinBox*>(editor);
```

```
    spinBox->setValue(value);
}

void delegateDoubleSpinBox::setModelData(QWidget *editor,
QAbstractItemModel *model,
                        const QModelIndex &index) const
{

    QDoubleSpinBox *spinBox = static_cast<QDoubleSpinBox*>(editor);
    spinBox->interpretText();
    double value = spinBox->value();
   model->setData(index, value, Qt::EditRole);
}

void delegateDoubleSpinBox::updateEditorGeometry(QWidget *editor,
                                const QStyleOptionViewItem &option,
                                const QModelIndex &/* index */) const
{
   editor->setGeometry(option.rect);
}
                                        delegatedate.h
#ifndef DELEGATEDATE_H
#define DELEGATEDATE_H

#include <QItemDelegate>
#include <QDateEdit>

class delegateDate  : public QItemDelegate
{
```

```
    Q_OBJECT
public:
   delegateDate(QObject *parent = 0);
   virtual ~delegateDate(){};

   QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem
&option,
                         const QModelIndex &index) const override;

   void setEditorData(QWidget *editor, const QModelIndex &index) const
override;
   void setModelData(QWidget *editor, QAbstractItemModel *model,
                     const QModelIndex &index) const override;

   void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem
&option,
                             const QModelIndex &index) const override;
};

#endif // DELEGATEDATE_H
```

delegatedate.cpp

```
#include "delegatedate.h"

delegateDate::delegateDate(QObject *parent)
   : QItemDelegate(parent)
{
}

QWidget *delegateDate::createEditor(QWidget *parent,
```

```
                                const QStyleOptionViewItem &/* option */,
                                const QModelIndex &/* index */) const
{

    QDateEdit  *editor = new QDateEdit(parent);

    return editor;
}

void delegateDate::setEditorData(QWidget *editor,
                                 const QModelIndex &index) const
{

    QDateEdit *dateEdit = static_cast<QDateEdit*>(editor);

    QString value = index.model()->data(index, Qt::EditRole).toString();

    QDate date = QDate::fromString(value, "dd.MM.yyyy");

    dateEdit->setDate(date);

}

void delegateDate::setModelData(QWidget *editor, QAbstractItemModel *model,
                                 const QModelIndex &index) const
{
    QDateEdit *dateEdit = static_cast<QDateEdit*>(editor);
//     spinBox->interpretText();
//     int value = spinBox->value();
```

```
//    QDate value = spinBox->date();
//     int value = spinBox->currentIndex();
    dateEdit->interpretText();
    QDate value = dateEdit->date();
    model->setData(index, value.toString("dd.MM.yyyy"), Qt::EditRole);
}

void delegateDate::updateEditorGeometry(QWidget *editor,
                                        const QStyleOptionViewItem &option,
                                        const QModelIndex &/* index */) const
{
    editor->setGeometry(option.rect);
}
```

delegatecombobox.h

```
#ifndef DELEGATECOMBOBOX_H
#define DELEGATECOMBOBOX_H

#include <QItemDelegate>
#include <QComboBox>

class delegateComboBox  : public QItemDelegate
{
    Q_OBJECT
public:
    delegateComboBox(QObject *parent = 0);
    virtual ~delegateComboBox(){};

    QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem
&option,
```

```
                const QModelIndex &index) const override;

//    void setEditorData(QWidget *editor, const QModelIndex &index) const
override;
   void setModelData(QWidget *editor, QAbstractItemModel *model,
                const QModelIndex &index) const override;

   void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem
&option,
                        const QModelIndex &index) const override;
};

#endif // DELEGATECOMBOBOX_H
                                    delegatecombobx.cpp
#include "delegatecombobox.h"

typedef QList<QPair<QString, int> > List;

delegateComboBox::delegateComboBox(QObject *parent)
   : QItemDelegate(parent)
{
}

QWidget *delegateComboBox::createEditor(QWidget *parent,
                            const QStyleOptionViewItem &/* option */,
                            const QModelIndex &index) const
{

   QComboBox* comboBox = new QComboBox(parent);
```

```
    // Под ролью UserRole +2 мы храним список пар значений
    QVariant vList = index.data(Qt::UserRole +2);
    // Преобразуем Вариант в наш список
    List list = vList.value<List>();

    // Инициализируем наш Комбо
    for(auto& item: list)
    {
        comboBox->addItem(QIcon(), item.first, item.second);
    }

    return comboBox;
}
/*
void delegateComboBox::setEditorData(QWidget *editor,
                                     const QModelIndex &index) const
{
    int value = index.model()->data(index, Qt::EditRole).toInt();


    QComboBox *comboBox = static_cast<QComboBox*>(editor);
//     spinBox->setCurrentIndex(value);
//    comboBox->setText(QString::number(value));
}
*/
void delegateComboBox::setModelData(QWidget *editor, QAbstractItemModel
*model,
                                    const QModelIndex &index) const
```

```
{
    QComboBox *edit = qobject_cast<QComboBox *>(editor);

    // Временно блокируем сигналы изменения модели, потому что он
эмитируется (emit)
    // каждый раз, когда в модель вносят какие-либо изменения
    model->blockSignals(true);

    // Сохраняем выбранное значение из Комбо
    model->setData(index, edit->currentIndex(), Qt::DisplayRole);

    // А теперь нам снова надо внести изменения в модель,
    // но при этом мы хотим, чтобы соответствующий сигнал (itemChanged)
сработал,
    // поэтому вновь включаем сигналы модели
    model->blockSignals(false);

    // Можешь закоментить оба вызова blockSignals и посмотреть что
получится

    // Сохраняем в модель выбранный ID
    // В модель внесли изменения и она эмитирует itemChanged
    // В роли Qt::UserRole +1 у нас ID пары
    model->setData(index, edit->itemData(edit->currentIndex()), Qt::UserRole +1);
}

void delegateComboBox::updateEditorGeometry(QWidget *editor,
                                            const QStyleOptionViewItem &option,
                                            const QModelIndex &/* index */) const
```

```
{
    editor->setGeometry(option.rect);
}
```