НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»**

**спеціальності «121 Інженерія програмного забезпечення»**

на тему: Подійно-орієнтована бібліотека для створення користувацьких інтерфейсів на основі власної мови розмітки (комплексна тема)

Виконав студенти IV курсу, групи ІП-92
(шифр групи)
Грицюк Володимир Васильович _____
(прізвище, ім'я, по батькові) (підпис)
Прошин Назарій Анатолійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викл. Головченко М. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст. викл. Головченко М. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ОТ, к.т.н., доц., Волокита А. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студенти _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

**ЗАВДАННЯ**
**на дипломний проєкт студентам**

Грицюку Володимиру Васильовичу та Прошину Назарію Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Подійно-орінєтована бібліотека для створення користувацьких інтерфейсів на основі власної мови розмітки (комплексна тема)

керівник проєкту Головченко Максим Миколайович, ст. викл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31»травня 2023 р. №2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис і аналіз предметної області, аналіз існуючих технологій та успішних ІТ-проектів, аналіз вимог до програмного забезпечення, постановка задачі.
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних.
3) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.
4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, підтримка програмного забезпечення.

5. Перелік графічного матеріалу

---

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1 | Вивчення рекомендованої літератури | 10.03.2023 | |
| 2 | Аналіз існуючих методів розв’язання задачі | 20.03.2023 | |
| 3 | Постановка та формалізація задачі | 10.04.2023 | |
| 4 | Розробка інформаційного забезпечення | 15.04.2023 | |
| 5 | Алгоритмізація задачі | 20.04.2023 | |
| 6 | Обґрунтування вибору використаних технічних засобів | 30.04.2023 | |
| 7 | Розробка програмного забезпечення | 5.05.2023 | |
| 8 | Налагодження програми | 15.05.2023 | |
| 9 | Виконання графічних документів | 20.05.2023 | |
| 10 | Оформлення пояснювальної записки | 25.05.2023 | |
| 11 | Подання ДП на попередній захист | 02.06.2023 | |
| 12 | Подання ДП рецензенту | 12.06.2023 | |
| 13 | Подання ДП на основний захист | 17.06.2023 | |

Студент _____ Володимир ГРИЦЮК
(підпис) (ініціали, прізвище)

Студент _____ Назарій ПРОШИН
(підпис) (ініціали, прізвище)

Керівник _____ Максим ГОЛОВЧЕНКО
(підпис) (ініціали, прізвище)

# АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 41 таблиць, 36 рисунків та 17 джерел – загалом 94 сторінки.

Дипломний проєкт присвячений розробці подійно-орієнтованої бібліотеки для створення користувацьких інтерфейсів на основі власної мови розмітки.

Мета: спрощення процесу інтеграції засобів розробки графічного інтерфейсу користувача у цільове програмне забезпечення, а також розширення можливостей налаштування контролерів при проєктуванні графічного інтерфейсу користувача за рахунок розробки подійно-орієнтованої бібліотеки, яка використовує зручну та інтуїтивно зрозумілу мову розмітки.

У першому розділі проведено змістовний огляд та аналіз предметної області, а також проаналізовано уже наявні IT-проекти, що вирішують схожі задачі. Представлено діаграму варіантів використання, що містить в собі основні аспекти створення інтерфейсів за допомогою розроблених технологій. Також розроблено вимоги функціональні вимоги, і показано їх взаємозв’язок з варіантами використання у вигляді матриці трасування.

У другому розділі описано моделювання та конструювання програмного забезпечення. Модель бізнес процесу використання розробленого програмного забезпечення представлено за допомогою BPMN діаграми. Наведено опис процесу створення застосунку з графічним інтерфейсом користувача з використанням бібліотеки та мови позмітки. Для реалізації бібліотеки було обрано монолітну архітектуру. Для деталізації архітектури програмного забезпечення наведено діаграми класів, описано призначення всіх сутностей. Також було описано алгоритми парсингу мови розмітки, створення та відображення віджетів на основі параметрів, а також процес обробки взаємодії з віджетами.

У третьому розділі було проведено аналіз якості та тестування програмного забезпечення. За допомогою статичного аналізатора було проаналізовано код програмного забезпечення, отримано високу оцінку. У розділі також описані процеси тестування. Всі тести пройдені успішно,

розроблена бібліотека відповідає всім функціональним та нефункціональним вимогам.

У четвертому розділі описано впровадження програмного забезпечення. Вихідний код бібліотеки розміщения на платформі Github. Наведено інструкцію користувача для розгортання бібліотеки та приклад розробки користувацького інтерфейсу з її використанням.

КЛЮЧОВІ СЛОВА: ПОДІЙНО-ОРІЄНТОВАНА БІБЛІОТЕКА, МОВА РОЗМІТКИ, Windows Api, ВІДЖЕТИ, UI.

# ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 41 tables, 36 figures and 17 sources – in total 94 pages.

The purpose of the diploma project is to develop of an event-oriented library for creating user interfaces based on its own markup language.

The goal is to simplify the process of integrating the graphical user interface development tools into the target software, as well as to expand the possibilities of configuring controllers when designing the graphical user interface due to the development of an event-oriented library that uses a convenient and intuitive markup language.

In the first chapter, a meaningful review and analysis of the subject area was carried out, and already existing IT projects solving similar problems were analyzed. A diagram of use cases is presented, which includes the main aspects of creating interfaces using the developed technologies. Functional requirements are also developed, and their relationship with use cases is shown in the form of a tracing matrix.

The second chapter describes software modeling and construction. The model of the business process of using the developed software is presented using a BPMN diagram. The process of creating an application with a graphical user interface using a library and a markup language is described. A monolithic architecture was chosen for the implementation of the library. Class diagrams are provided to detail the software architecture, and the purpose of all entities is described. Algorithms for parsing the markup language, creating and displaying widgets based on parameters, and the process of handling interactions with widgets were also described.

In the third chapter, quality analysis and software testing were conducted. The software code was analyzed with the help of a static analyzer, and a high score was obtained. The section also describes the testing processes. All tests were passed successfully, the developed library meets all functional and non-functional requirements.

The fourth chapter describes the implementation of the software. The source code of the library is hosted on the Github platform. User instructions for deploying the library and an example of developing a user interface using it are provided.

KEYWORDS: EVENT-ORIENTED LIBRARY, MARKUP LANGUAGE, Windows Api, WIDGETS, UI.

| № з/п | Формат | Позначення | Найменування | Кількість листів | Примітка |
|---|---|---|---|---|---|
| 1 | А4 | | Завдання на дипломний проєкт | 2 | |
| 2 | А4 | КПІ.ІП-9209.045490.01.91 | Технічне завдання | 16 | |
| 3 | А4 | КПІ.ІП-9209.045490.02.81 | Пояснювальна записка | 94 | |
| 4 | А4 | КПІ.ІП-9209.045490.03.12 | Текст програми | 91 | |
| 5 | А4 | КПІ.ІП-9209.045490.04.51 | Програма та методика тестування | 6 | |
| 6 | А4 | КПІ.ІП-9209.045490.05.34 | Керівництво користувача | 8 | |

| Змін. | Арк. | № докум. | Підп. | Дата | КПІ.ІП-9209.045490.00.90 | | |
|---|---|---|---|---|---|---|---|
| Розроб. | | Грицюк В. В. | | | Відомість дипломного проєкту | Літ. | Аркуш 1 / Аркушів |
| Перевір. | | Головченко М.М | | | | | |
| Н.контр. | | Ліщук К.І. | | | | КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-92 | |
| Затв. | | Жаріков Е.В. | | | | | |

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2023 р.

# ПОДІЙНО-ОРІЄНТОВАНА БІБЛІОТЕКА ДЛЯ ПРОЕКТУВАННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА НА ОСНОВІ ВЛАСНОЇ МОВИ РОЗМІТКИ (комплексна тема)

**Технічне завдання**

КПІ.ІП-9209.045490.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Володимир ГРИЦЮК

_____ Назарій ПРОШИН

Київ – 2023

**ЗМІСТ**

# 1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Подійно-орієнтована бібліотека для створення користувацьких інтерфейсів на основі власної мови розмітки.

Галузь застосування: Розроблена бібліотека може бути використана при написанні додатків з користувацьким інтерфейсом на мові програмування C++ під операційну систему Windows.

Наведене технічне завдання поширюється на розробку подійно-орієнтованої бібліотеки, що включає в себе розробку функціоналу для створення графічного інтерфейсу та розробку власної мови розмітки для проектування інтерфейсу. Розроблений продукт використовується для створення графічного інтерфейсу у проєктах на мові програмування C++.

# 2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки подійно-орієнтованої бібліотеки для створення користувацьких інтерфейсів на основі власної мови розмітки є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

# 3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для створення графічного інтерфейсу користувача з використанням мови розмітки у програмному забезпеченні на мові програмування C++.

Метою розробки є спрощення процесу інтеграції засобів розробки графічного інтерфейсу користувача у цільове програмне забезпечення, а також розширення можливостей налаштування контролерів при проєктуванні графічного інтерфейсу користувача за рахунок розробки подійно-орієнтованої бібліотеки, яка використовує зручну та інтуїтивно зрозумілу мову розмітки.

# 4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

### 4.1.1 Користувацького інтерфейсу

– розроблена бібліотека повинна дозволяти користувачу використовуючи специфічну мову розмітки спроектувати інтерфейс користувача;

– бібліотека повинна забезпечувати можливість визначення логіки обробки подій взаємодії з віджетами: натисканні на кнопки, введення тексту в поле для вводу;

– за допомогою бібліотеки, а саме мови розмітки, можна визначати параметри наступних віджетів: текстового напису (Label), кнопки (Button), кнопки вибору опції (RadioButton), кнопки-прапорця (CheckBox), поля для вводу (EditLine), віджета вибору цілочисельних значень (NumberBox);

– у файлі розмітки користувач може визначати наступні параметри текстового напису (Label): розмір, позицію, текст та шрифт;

– у файлі розмітки користувач може визначати наступні параметри кнопки (Button): розмір, позицію, текст та шрифт, можливість реагувати на натискання та подвійне натискання;

– у файлі розмітки користувач може визначати наступні параметри кнопки вибору опції (RadioButton): розмір, позицію, текст та шрифт, можливість реагувати на натискання, початкове значення, а також викоремлювати групи, в межах яких може бути активованою тільки одне кнопка;

– у файлі розмітки користувач може визначати наступні параметри кнопки-прапорця (Button): розмір, позицію, текст та шрифт, початкове значення, можливість реагувати на натискання;

– у файлі розмітки користувач може визначати наступні параметри поля вводу (EditLine): розмір, позицію, текст та шрифт, однорядковість чи багаторядковість;

– у файлі розмітки користувач може визначати наступні параметри віджета для вибору цілочисельного значення (NumberBox): розмір, позицію, початкове, мінімальне та максимальне значення;

– за допомогою бібліотеки, а саме мови розмітки, можна налаштовувати параметри вікна, а саме розмір та заголовок;

– за допомогою бібліотеки можна створити програмне забезпечення, з своєю логікою обробки подій(наприклад калькулятор, програмне забезпечення для розв'язку СЛАР);

– розробка є бібліотекою, тому не передбачає графічного інтерфейсу користувача.

4.1.2 Для користувача:

– розроблена бібліотека повинна давати користувачу можливість проектування інтерфейсу з використанням основних віджетів: текстових написів (Label), кнопок (Button), полей вводу (EditLine), кнопкок-прапорців (CheckBox), кнопки вибору (radioButton);

– створення та відображення вікна на основі параметрів (розмір, заголовок);

– створення та відображення кнопки на основі параметрів (розмір, позиція, здатність реагувати на натискання та подвійне натискання, текст);

– створення та відображення кнопки-прапорця на основі параметрів (розмір, позиція, здатність реагувати на натискання, текст, початкове положення)

– створення та відображення кнопки вибору опції на основі параметрів (розмір, позиція, здатність реагувати на натискання, текст, початкове положення, ідентифікатор групи, в межах якої тільки одна кнопка може бути активованою);

– створення та відображення віджета для вибору цілочисельного значення в певному діапазоні на основі параметрів (розмір, позиція, текст, мінімальне, максимальне та значення за замовчуванням);

– створення та відображення текстового напису на основі параметрів (розмір, позиція, текст);

– створення та відображення поля для вводу на основі параметрів (розмір, позиція, багаторядкове чи однорядкове);

– всі параметри віджетів можуть задаватися специфічною мовою розмітки для проектування графічного інтерфейсу у файлі розмітки з назвою “CppUI”;

– користувач повинен мати можливість програмно визначати поведінку програми при взаємодії з віджетами, а саме при натисканні на кнопки (Button, CheckBox, RadioButton) та при введенні тексту в поле для вводу (EditLine).

4.1.3 До мови розмітки

– синтаксис: Мова розмітки повинна мати чіткий, структурований і легко читаємий синтаксис;

– теги: Мова розмітки повинна мати широкий набір тегів, які можуть розмічати різні типи елементів і структур в документі;

– чистота: Мова розмітка повинна бути зрозумілою та логічною для комп'ютерів і людей. Вона має бути без синтаксичних помилок;

– простота: Мова розмітки повинна мати простий та легко зрозумілий синтаксис, що дозволяє розробникам швидко вивчати та використовувати її. Вона також повинна бути зручною для редагування та підтримки;

– універсальність: Мова розмітки може бути придатною для використання в різних та на різних платформах. Вона повинна мати широку підтримку серед різних програмних засобів та бути незалежною від конкретної платформи;

– розширюваність: Мова розмітка повинна мати можливість розширення за допомогою нових елементів, атрибутів та правил. Це дозволяє отримати змінні потреби розробників та забезпечити сумісність з майбутніми версіями мови;

– граматика: Мова розмітки повинна перевіряти вхідний файл на наявність помилок та, після виявлення таких, припинити парсинг розмітки та сповістити користувача про тип помилки та місце, де ця помилка була здопущена. При цьому, підтримується основні дві команди розмітки – команда створення віджету та команда зміни параметру віджету. Команда створення ввіджету повинна мати наступний вигляд: “| widget [”, команда зміни віджета повинна мати наступний вигляд “/ param : value”;

– зміна параметрів повинна бути доступна лише у “scope” віджета;

– семантика: Мова розмітки повинна підтримувати наступні операнди: “:”, “/”, “]”, “[”,“|”, “#”. “:” – повідомляє, що після цього операнда буде введено значення параметра віджету. “/” – означає, що в цьому рядку буде змінено параметр віджету. “]” – закриває “scope” віджету. “[” – відкриває “scope” віджета “/”. “|” – означає, що в цьому рядку буде сторено новий віджет. “#” – повідомляє, що цей рядок є кометарем і його не потрібно опрацьовувати;

– синтаксис: Мова розмітки повинна обробляти ключові слова: “Button”, “CheckBox”, “EditLine”, “Label”, “NumberBox”, “RadioButton”, “Text”, “sizeVert”, “sizeGorz”, “positionVert”, “positionGorz”, “name”, “clicable”, “doubleClicable”, “defaultValue”, “multiline”, “minValue”, “maxValue”;

– мова розмітки повинна перевіряти наявність параметрів у віджета при запиті зміни параметра віджета користувачем;

– мова розмітки повинна перевіряти можливість створення віджета у скоупі іншого віджета;

– мова розмітки повинна перевіряти структуру команд(створення віджета, зміни параметрів);

– мова розмітки повинна перевіряти введене користувачем значення для кожного параметра

4.1.4 Для адміністратора системи (якщо він передбачений):

Адміністратор системи не передбачений.

4.1.5 Додаткові вимоги:

– Розроблена бібліотека мова розмітки повинна бути зрозумілою для користування;

– бібліотека повинен бути адаптована для розробки програм під операційні систем: Windows 8/8.1/10/11.

4.2 Вимоги до надійності

Передбачити систему перевірки файлу розмітки на можливі помилки. Розробити систему оповіщення про наявність у файлі розмітки.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core Pentium;

– об‘єм ОЗП: 2 Гб;

– C++ компілятор з підтримкою стандарту C++17.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i3 або вище;

– об‘єм ОЗП: 4 Гб або більше;

– C++ компілятор з підтримкою стандарту C++17.

## 4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows 7, Windows 8, Windows 10 і т.д.).

### 4.5.1 Вимоги до вхідних даних

Користувач повинен описувати параметри віджетів у файлі розмітки з назвою “CppUI.txt”.

### 4.5.2 Вимоги до вихідних даних

У файлах “Error.txt”, “Warnings.txt”, “Msg.txt” повинні зберігатися відповідні “логи” роботи компонента парсингу розмітки.

### 4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C++.

### 4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища Visual Studio Code.

### 4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді текстових файлів.

## 4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

## 4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

## 4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

# 5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

## 5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання мови розмітки;

– схема структурна варіантів використання подійно-орієнтованої бібліотеки для проектування графічного інтерфейсу користувача;

– схема структурна класів програмного забезпечення, що реалізує мову розмітки;

– схема структурна класів програмного забезпечення, що реалізує подійно-орієнтовану бібліотеку;

– креслення вигляду екранних застосунку, розробленого з використанням бібліотеки та мови позмітки;

– креслення вигляду екранних прикладу файлу розмітки.

## 5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

# 6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

| № | Назва етапу | Строк | Звітність |
|---|---|---|---|
| 1. | Вивчення рекомендованої літератури | 10.03 | |
| 2. | Аналіз існуючих методів розв’язання задачі | 20.03 | Технічне завдання |
| 3. | Постановка та формалізація задачі | 10.04 | Специфікації програмного забезпечення |
| 4. | Розробка інформаційного забезпечення | 15.04 | Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму) |
| 5. | Алгоритмізація задачі | 20.04 | |
| 6. | Обгрунтування вибору використаних технічних засобів | 30.04 | |
| 7. | Розробка програмного забезпечення | 5.05 | Тексти програмного забезпечення |
| 8. | Налагодження програми | 15.05 | Тести, результати тестування |
| 9. | Виконання графічних документів | 20.05 | Графічний матеріал проекту |
| 10. | Оформлення пояснювальної записки | 25.05 | Пояснювальна записка |
| 11. | Подання ДП на попередній захист | 02.06 | Технічна документація |

| № | Назва етапу | Строк | Звітність |
|---|---|---|---|
| 12. | Подання ДП рецензенту | 12.06 | Технічна документація |
| 13 | Подання ДП на основний захист | 17.06 | Технічна документація |

# 7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

# Пояснювальна записка
## до дипломного проєкту

на тему: Подійно-орієнтована бібліотека для створення користувацьких інтерфейсів на основі власної мови розмітки (комплексна тема)

КПІ.ІП-9209.045490.02.81

Київ – 2023

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| | | |
|---|---|---|
| IDE | – | Integrated Development Environment – інтегроване середовище розробки. |
| API | – | Application programming interface, прикладний програмний Інтерфейс |
| SDK | – | Software development kit |
| IT | – | Інформаційні технології |
| ОС | – | Операційна система. |
| UI | – | User Interface - інтерфейс користувача |
| ПК | - | Портативний комп’ютер |
| WinAPI | - | набір базових функцій інтерфейсів програмування застосунків ОС сімейства Windows |
| ПОП | - | Подійно-орієнтоване програмування |
| ГІК | - | Графічний інтерфейс користувача |
| ПЗ | - | Програмне забезпечення |
| CppUI | - | Подійно-орієнтована бібліотека для проектування інтерфейсів користувача з використанням власної мови розмітки |

# ВСТУП

Відомо, що сфера інформаційних технологій з кожним роком набирає все більших і більших обертів. Напевно кожен другий школяр бачить себе програмістом у майбутньому. Така популярність професії розробників програмного забезпечення не є безпідставною - щодня у світі зростає потреба у наданні інформаційних послуг.

Важливою складовою більшості інформаційних систем є графічний інтерфейс. Саме ця частина робить програмний продукт зручним, зрозумілим, простим у використанні. Користувачі прагнуть отримувати від програмного забезпечення можливість швидко взаємодіяти з ним, не задумуючись про значення конкретних дій, вони мають бути інтуїтивно зрозумілими.

Створення зручного графічного інтерфейсу користувача стало однією з головних задач програмістів в сучасному світі. Застосунки, незалежно від технологій, які вони використовують, обов'язково містять у собі частину, що відповідає за інтерфейс користувача. Інколи, для цього навіть створюються окремі команди.

Розробка програмних продуктів мовою програмування C++ також часто вимагає створення графічного інтерфейсу користувача. Але, наявні рішення є досить громісткими, масивними, це повністю окремі технології. Для освоєння цих інструментів потрібно витратити немало часу. Інколи, щоб почати проектування графічного інтерфейсу для вже наявного програмного продукту, потрібно витратити не один тиждень для переписування логіки з використанням певної технології. Враховуючи швидкість розвитку подій в сучасному світі, особливо актуальним є спрощення і прискорення інтеграції інструментів для проектування графічного інтерфейсу користувача. Саме це є однією з головних цілей розробки даної бібліотеки. Не менш важливим є і зручність, оперативність та варіативність процесу створення інтерфейсу користувача. Створення бібліотеки, що буде використовувати спеціальну мову

розмітки, дозволить значно прискорити і сам процес проектування графічного інтерфейсу користувача.

Отож, актуальність розробки подійно-орієнтованої бібліотеки для створення графічного інтерфейсу користувача полягає у спрощенні інтеграції інструментів для проектування графічного інтерфейсу користувача за рахунок створення бібліотеки, яку можна просто і без зайвих витрат інтегрувати до будь-якого консольного застосунку, написаного на мові програмування C++, а також значне прискорення проектування графічного інтерфейсу за рахунок використання зручної та інтуїтивно простої мови розмітки.

# 1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 1.1 Загальні положення

Робота присвячена розробці програмного забезпечення (подійно-орієнтованої бібліотеки проектування графічних інтерфейсів користувача), яка повинна полегшити та пришвидшити процедуру створення додатків з інтерфейсом користувача та спростити контроль подій, за допомогою яких здійснюється взаємодія користувача з графічним інтерфейсом. Для проектування графічного інтерфейсу використовується розроблена інтуїтивно простою мова розмітки, що дозволяє навіть не дуже досвідченому розробнику сконструювати простий інтерфейс користувача з набагато меншими затратами часу на освоєння технології використання відповідної бібліотеки та синтаксису мови розмітки.

Подійно-орієнтоване програмування (ПОП)[1] – це парадигма програмування, в якій виконання програми керується подіями. Подія може бути викликана дією користувача або зміною стану системи через введення інформації, яка впливає на операційну систему або програмне забезпечення. Головною особливістю ПОП є наявність головного циклу, що постійно очікує на виникнення подій, та розділення коду програми на дві частини: код, який обробляє події, та код, який повідомляє про виникнення подій.

Графічний інтерфейс користувача (ГІК)[2] - це тип інтерфейсу, що дозволяє користувачам взаємодіяти з комп'ютером за допомогою графічних зображень та візуальних елементів. В окремих випадках можуть використовуватися також елементи текстових інтерфейсів, які базуються на введенні та навігації за допомогою тексту. ГІК дозволяє маніпулювати графічними об'єктами. Використання ГІК розповсюджене на багатьох пристроях, які здатні відображати графічну інформацію, але у дипломній роботі ми розглядаємо використання ГІК на персональних комп'ютерах.

Одним з важливих критеріїв "вдалого" графічного інтерфейсу є концепція "роби те, що я маю на увазі". Ця концепція вимагає передбачуваної

роботи системи так, щоб користувач міг інтуїтивно зрозуміти, яку дію виконає програма після його команди. У розробці нашої бібліотеки ми дотримуємося цього принципу. Крім того, нашим завданням було зробити процес створення візуалізації простим і зрозумілим для користувача.

Мова розмітки[3] – це штучна мова, яка використовує набір інструкцій та анотацій для структурування тексту. Хоча перші мови розмітки з'явилися кілька століть тому, вони набули популярності в останні десятиріччя, особливо в галузі інформаційних технологій. За допомогою спеціальних технологій, мова розмітки дозволяє не лише надати тексту структуру, але й створити відповідний графічний інтерфейс. Загальною рисою багатьох мов розмітки є змішання тексту документу з розмітковими інструкціями у одному потоці даних. Цей термін походить з практики розмічування рукописів спеціальними символами, яку виконували кваліфіковані топографи. Кожен символ розмітки мав велике значення. Мови розмітки широко використовуються дизайнерами, редакторами та коректорами. У контексті програмування, мови розмітки використовуються часто при розробці графічних інтерфейсів. Нашим завданням було створити інтуїтивно зрозумілу мову розмітки, яка була б ефективною при парсингу та мала читабельний синтаксис.

Мова розмітки повинна мати чіткий, структурований і легкий для читання синтаксис, що не допускає синтаксичних помилок, повинна бути зрозумілою та логічною, а також мати свою семантику та граматику. Головною відмінністю між мовою програмування та мовою розмітки є те, що за допомогою мови розмітки не можна прописувати логіку програмного забезпечення. Саме таких положень та вимог до мови розмітки було дотримано, тому розроблена мова дійсно є мовою розмітки.

Задача створення додатків з інтерфейсом користувача з кожним роком не втрачає своєї актуальності, оскільки у світі велика кількість розробників та користувачів, які надають перевагу додаткам з інтерфейсом користувача ніж без інтерфейсу, оскільки це покращує та спрощує взаємодію з програмним

застосунком та дає можливість наочно спостерігати за певними подіями та реакціями ПЗ на дії користувача в застосунку. Створення такого типу додатку вручну є задача надскладна та затратна в часі, а наявні техінчні рішення мають високий поріг вимог до параметрів ПК.

Для вирішення цих проблем спеціалісти створили велику кількість технологій: бібліотеки, фреймворки, найвідоміші з них: Qt, WinForms. Найбільш розвинені технології роботи з графічними та системними компонентами, доступ до яких треба мати від найнижчого рівня надає мова програмування C++ та бібліотеки, які створені за допомогою використання цієї мови програмування(в нашому випадку WinAPI).

## 1.2 Змістовний опис і аналіз предметної області

В даній дипломній роботі буде реалізовуватися подійно-орієнтована бібліотека для проектування графічного інтерфейсу користувача на основі власної мови розмітки. Програмним забезпеченням передбачена можливість програмування поведінки віджетів та реакцію віджетів на конкретні дії користувача. Також користувачу буде надаватися можливість створення віджетів, надання віджетам бажаних параметрів та розміщення віджедів відповідно до вподобання користувача. Маючи цей перелік можливостей, користувач має змогу створити програмне забезпечення з UI, яке зможе виконувати різноманітні дії, обмежені лиш двохвимірним ГІК та власними навичками програмування.

Основною проблемою сучасних подійно-орієнтованих бібліотек та фреймворків (Qt [4], WinForms [5], GIMP Toolkit [6], wxWidgets [7]) є незрозумілий інтерфейс роботи з дизайном програмного забезпечення та складне програмування поведінки віджетів. Велика кількість параметрів та віджетів доволі часто обмежують одне одного, а інколи конфлікутують, що призводить до непередбаченої поведінки програмного забезпечення. Інтеграція цих засобів до будь-якого застосунку не завжди можлива, оскільки згадані інструменти є повноцінними технологіями.

У дипломній роботі вирішено ці проблеми за допомогою інтуїтивно зрозумілої мови розмітки та спрощенням програмування віджетів за допомогою інструментів мови програмування C++ та бібліотек, написаних цією мовою. Більшість уваги та зусиль зосереджено на часто використовуваних віджетах та їх параметрах, що дозволяє швидше ознайомитись з функціоналом CppUI.

Для коректного відображення віджетів було розроблено три типи команд в розмітці, в залежності від типу команди, парсер буде або створювати нові об’єкти, або змінювати вже створені об’єкти, або ігнорувати рядок розмітки. Бібліотека повинна розпарсити всі ці команди, і повернути структуру, що містить в собі всі необхідні дані для створення вікон та контролерів. Оскільки і вікно, і контролери можуть містити в собі параметри, значення яких може бути встановлене за замовчуванням, користувачу не потрібно вказувати абсолютно всі доступні поля в файлі розмітки (а тільки необхідні, тобто значення яких не можуть бути визначені без участі користувача). Також, було розроблено функціонал, що дозволяє користувачу визначити алгоритм дій, що буде результатом взаємодії користувача з контролером (наприклад, натискання на кнопку). Для цього кожен контролер містить в собі перелік подій, якій можуть відбутися при взаємодії користувача з даним віджетом. Використовуючи спеціально розроблений інтерфейс, користувач може налаштувати специфічну поведінку застосунку при виникненні тих чи інших подій.

Керування програмним застосунком відбувається за допомогою клавіш клавіатури, та комп’ютерної миші.

## 1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації подійно-орієнтованої бібліотеки для проектування інтерфейсів користувача з використанням

власної мови розмітки(CppUI). Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

### 1.3.1 Аналіз відомих алгоритмічних та технічних рішень

В якості основи для створення мови розмітки та парсингу її використано власний алгоритм парсингу, оскільки на даний момент не існує спеціальних алгоритмів, для роботи з розмітками мовою C++ чи іншою C-подібною мовою, які задовільнять вимоги диплоного проекту, а загальні алгоритми, наприклад алгоритм для обробки документів не підходять через незрозумілість і унікальність свого синтаксису, тому при розробці мови розмітки буде враховано ці недоліки та усунено їх.

В якості основи для створення інтерфейсу спроектовано власну об’єктно-орієнтовану бібліотеку для реалізації представлення таких сутностей, як застосунок, вікно, контролер у вигляді об’єктів. Дане рішення у своїй імплементації використовує функціонал WinAPI, що дозволяє нам без критичних обмежень з точки зору функціональності контролерів, орзанізувати чітку, зрозумілу та структуровану взаємодію з різними об’єктами. Для реалізації подійно-орієнтованого механізму нашої бібліотеки використано доволі добре відому парадигму подійно-керованої архітектури програмного забезпечення та адаптовано її під наші потреби.

Алгоритм створення інтерфейсу користувача складається з наступних кроків:

– заванатаження файла розмітки
– парсинг файла розмітки порядково;
– перевірка рядків на коректність введеної інформації;
– на основі отриманої інофрмації, створення віджетів;
– надання властивостей віджетам;
– надання інформації про віджети та їх властивості бібліотеці WinAPI;

– обробка інструментами бібліотеки WinAPI отриманої інформації та створення інтерфейсу користувача;

Алгоритм реакції на подію:

– відправлення функціоналом WinAPI спеціального повідомлення до handle-об’єкта відповідного віджета;

– аналіз отриманого повідомлення та конкретизація дії, що сталося;

– виклик сигналу, що відповідає отриманому повідомленню;

– сповіщення підписників про дію, що відбулася відповідно до принципу подійно орієнтованої архітектури.

Архітектура програмування описує загальну структуру, організацію та взаємозв'язки між компонентами програмного забезпечення. Існує кілька основних архітектурних підходів у програмуванні, таких як монолітна архітектура, клієнт-серверна архітектура, розподілена архітектура та мікросервісна архітектура. Нижче наведено огляд та аналіз цих архітектур.

У монолітній архітектурі весь програмний продукт розглядається як єдиний блок, в якому всі компоненти і функціональність об'єднані разом. Всі компоненти спільно виконуються в одному процесі і збираються разом в один пакет. Цей підхід простий у розробці та розгортанні, але може стати проблематичним у випадку значних масштабів, погіршення продуктивності або потреби в масштабованості окремих компонентів. У клієнт-серверній архітектурі програмне забезпечення розділяється на дві основні частини: клієнтську та серверну. Клієнтська частина відповідає за взаємодію з користувачем і відображення інформації. Серверна частина забезпечує обробку запитів, бізнес-логіку та збереження даних. Цей підхід дозволяє розділити відповідальності та полегшити масштабування системи. Розподілена архітектура передбачає, що компоненти програми можуть працювати на різних фізичних машинах або в мережі комп'ютерів. Комунікація між компонентами здійснюється за допомогою мережевих протоколів. Цей підхід дозволяє розподілити навантаження, забезпечити більшу масштабованість і високу доступність системи. Мікросервісна

архітектура передбачає розбиття програмного продукту на невеликі незалежні сервіси, які працюють окремо. Кожен сервіс виконує окрему функціональність і може бути розроблений, впроваджений і масштабований незалежно. Взаємодія між сервісами здійснюється через легкі механізми комунікації, наприклад, REST API або повідомлення. Цей підхід забезпечує гнучкість, легкість масштабування та підтримку різних технологій для кожного сервісу [8].

Вибір підходу до архітектури програмування залежить від вимог проекту, його розміру, складності та інших факторів. Кожна архітектура має свої переваги та обмеження, і важливо обирати ту, яка найкраще відповідає потребам конкретного проекту. Зважаючи на вище сказане було обрано монолітний тип архітектури.

Зважаючи на особливості програмного забезпечення, для деталізації внутрішньої архітектури подійно-орієнтованої бібліотеки для розробки продукту може бути обрано наступні патерни проектування: Observer для обробки подій та Factory Method для створення віджетів на основі отриманих параметрів розмітки.

Factory Method [9] – це патерн проектування, який визначає загальний інтерфейс для створення об'єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об'єктів. Він пропонує створювати об'єкти не безпосередньо за допомогою оператора new, а замість цього використовувати особливий фабричний метод. Структура Factory Method складається з наступних елементів: Creator, Client Creator, Concrete Products, Product. Creator (створювач) – це абстрактний клас або інтерфейс, який визначає загальний фабричний метод для створення об'єктів. Цей метод повертає об'єкт інтерфейсу або базового класу. Concrete Products (конкретні продукти) – це класи, які успадковуються від Creator і реалізують фабричний метод для створення конкретних об'єктів. Кожен конкретний продукт має свою власну реалізацію фабричного методу. Client (клієнт) – це клас або модуль, який використовує фабричний метод для створення об'єктів. Клієнтський код

працює з Creator через його інтерфейс, тим самим залежить від абстракції, а не від конкретних класів. Product (продукт) – це загальний інтерфейс або базовий клас, який визначає методи, що будуть реалізовані конкретними продуктами, створюваними фабричним методом.

Factory Method дозволяє розділити процес створення об'єктів від їх використання, що сприяє зменшенню залежностей між класами і полегшує розширення коду. Він також підтримує принцип "програмування на основі інтерфейсів" і дозволяє гнучко змінювати тип створюваних об'єктів за допомогою підкласів.

Observer (спостерігач)[10] – це поведінковий патерн проектування, який створює механізм підписки, що дозволяє об'єктам стежити та реагувати на події, які відбуваються в інших об'єктах. Структура Observer складається з чотирьох елементів: Publisher, Subscriber, Concrete Subscribers, Client. Publisher (видавець) – володіє внутрішнім станом, зміни якого цікаві підписникам. Він містить механізм підписки, який включає список підписників та методи для підписки та відписки підписників. Subscriber (підписник) – це інтерфейс або базовий клас, який визначає метод, яким користується видавець для надсилання сповіщень. Зазвичай цього методу достатньо для отримання сповіщення про подію, але можуть бути і додаткові методи з параметрами для передачі додаткової інформації. Concrete Subscribers (конкретні підписники) – це класи, які реалізують інтерфейс підписника. Вони виконують певні дії або реагують на сповіщення, які надсилаються від видавця. Конкретні підписники мають відповідні реалізації методів інтерфейсу підписника. Client (клієнт) – це клас або модуль, який створює об'єкти видавців та підписників і встановлює зв'язки між ними. Клієнт може реєструвати підписників на оновлення у видавцях та ініціювати події, що призводять до надсилання сповіщень підписникам.

Патерн Спостерігач надає можливість будь-якому об'єкту з інтерфейсом підписника зареєструватися для отримання сповіщень про події, що трапляються в об'єктах-видавцях. Він дозволяє розділити обов'язки між

видавцем і підписниками, забезпечуючи слабку залежність між ними та покращуючи розширюваність та перевикористання коду.

### 1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Розроблена бібліотека написана на мові програмування C++. Дана технологія вибрана тому, що графічний інтерфейс на цій мові проєктується на порядок складніше, ніж на інших мовах програмування, отже актуальність даної роботи збільшується. C++ надає більше можливостей для роботи на низьких рівнях програмування та роботи з пам’яттю, що критично важливо для програмного забезпечення даного типу.

В якості основи для створення інтерфейсу використано WinAPI[11] - ключовий набір API, що доступний на операційних системах Microsoft Windows. Було вибрано саме цю технологію, оскільки вона є безкоштовною та одною з найпопулярніших технологій, отже має велику кількість документації що пришвидшує та спрощує розробку програмного забезпечення. WinAPI дозволяє нам отримати доступ до основного функціоналу операційної системи, що дозволяє створювати графічний інтерфейс, а також відловлювати події, які симулює користувач чи операційна система, і на основі результатів опрацювання цих подій, виконувати відповідні інструкції.

Розглянемо існуючі IDE для взаємодії з мовою C++ та WinAPI. Для розгляду було взято найбільш популярні IDE, оскільки вони активно підтримуються розробниками та користувачами, а саме Visual Studio[12], Visual Studio Code[13], Code::Blox[14], Qt5 Creator, Eclipse[15].

Visual Studio – потужне інтегрованим середовищем розробки з багатьма функціями, такими як розширена підтримка відладки, автодоповнення, інструменти для створення графічних інтерфейсів користувача та інше. Плюси: Велика кількість інструментів і плагінів, міцна підтримка від Microsoft, добра інтеграція з іншими продуктами Microsoft. Мінуси: Може

бути важким для вивчення та використання, особливо для початківців. Ресурсомісткий.

Visual Studio Code – легке інтегроване середовище розробки з підтримкою розширень, можливістю роботи з Git, вбудованим відладчиком і багатьма іншими функціями. Плюси: Легкість використання, широкі можливості розширення, хороша підтримка мови C++ та WinAPI. Мінуси: Може бути менш потужним інструментом порівняно з Visual Studio, залежить від розширень для деяких функціональних можливостей.

Code::Blocks - це відкрите середовище розробки з підтримкою мови C++, включаючи компіляцію, відладку, автодоповнення, налаштування проектів та багато іншого. Плюси: Легкість використання, широка підтримка мови C++, підтримка різних компіляторів. Мінуси: Обмежені можливості порівняно з деякими іншими IDE, може бути менш популярним серед великих комерційних проектів.

Qt Creator є спеціалізованим середовищем розробки для роботи з фреймворком Qt, проте він також підтримує мову C++ та WinAPI. Включає можливості для проектування графічних інтерфейсів користувача, відладки, автодоповнення та інші. Плюси: Хороша інтеграція з фреймворком Qt, візуальне проектування інтерфейсів, підтримка відладки. Мінуси: Обмежена підтримка для не-Qt проектів, може бути важким для вивчення для тих, хто не має досвіду з фреймворком Qt.

Eclipse є потужним інтегрованим середовищем розробки, яке підтримує мову C++ та WinAPI. Включає функції редактора, компіляції, відладки, автодоповнення, керування проектами та інші. Плюси: Широкі можливості розширення, хороша підтримка мови C++ та WinAPI, активна спільнота користувачів. Мінуси: Важкість використання та навчання для новачків, може бути вимогливим до ресурсів комп'ютера.

Ці IDE є популярними серед розробників C++ та WinAPI, проте кожен з них має свої переваги та недоліки. Вибір IDE залежить від особистих вподобань, проектних потреб, рівня експертизи та доступності ресурсів

комп'ютера. Тому, для роботи з кодом булл вибраний code redactor “Visual Studio Code” та IDE Visual studio, оскільки це безкоштовні технологія, що мають велику кількість розширень, що збільшують функціонал ПЗ. Було вибрано саме цю технологі, оскільки вона є безкоштовною та являється однією з найпопулярніших, що спрощує пошук документації та пришвидшує час розробки програмного забезпечення.

“Visual Studio Code” не може сам компілювати та запускати код, тому для цього було обрано CMake[16] – крос-платформенний компілятор, який генерує файли управління на основі інструкцій в файлах CMakeLists.txt. Було вибрано саме цю технологію, оскільки вона є безкоштовною та являється однією з найпопулярніших, що спрощує пошук документації та пришвидшує час розробки програмного забезпечення.

### 1.3.3 Аналіз відомих програмних продуктів

Для проведення аналізу та порівняння зі застосунком, розробленим в дипломній роботі було обрано найпопулярніші програмні застосунки для проектування інтерфейсів користувача є Qt та WinForms, GIMP Toolkit та wxWidgets.

Qt – крос-платформовий інструментарій розробки програмного забезпечення (ПЗ) для створення графічних інтерфейсів користувача, а також кросплатформенних програм, які працюють на різних програмних та апаратних платформах, таких як Linux, Windows, macOS, Android. Але, Qt не обмежується створенням тільки графічних, а вміщує в собі функціональність для взаємодії з базами даних, мережею та багато інших. Це призводить до того, що для освоєння всього функціоналу даного фреймворку потрібно затратити багато ресурсів, що може бути недоцільно, якщо ціллю є тільки проектування простого графічного інтерфейсу. Навіть наявність великої змістовної документації може не надто спростити вивчення описаного фреймфорку.

Перевагами програмного застосунку застосунку є його обмежена безплатна версія, та достатній доступний фунціонал навіть у безплатній версії.

Недоліками цього програмного застосунку є відносно високі вимоги до параметрів ПК та об’єму пам’яті. Крім цього безплатні версія не надає доступу до всіх можливостей продукту. Qt має доволі складний інтерфейс, що являється важким для застосування цього інструменту користувачем без досвіду роботи з цією технологією.

Інтерфейс застосунку зображено на рисунку 1.1.

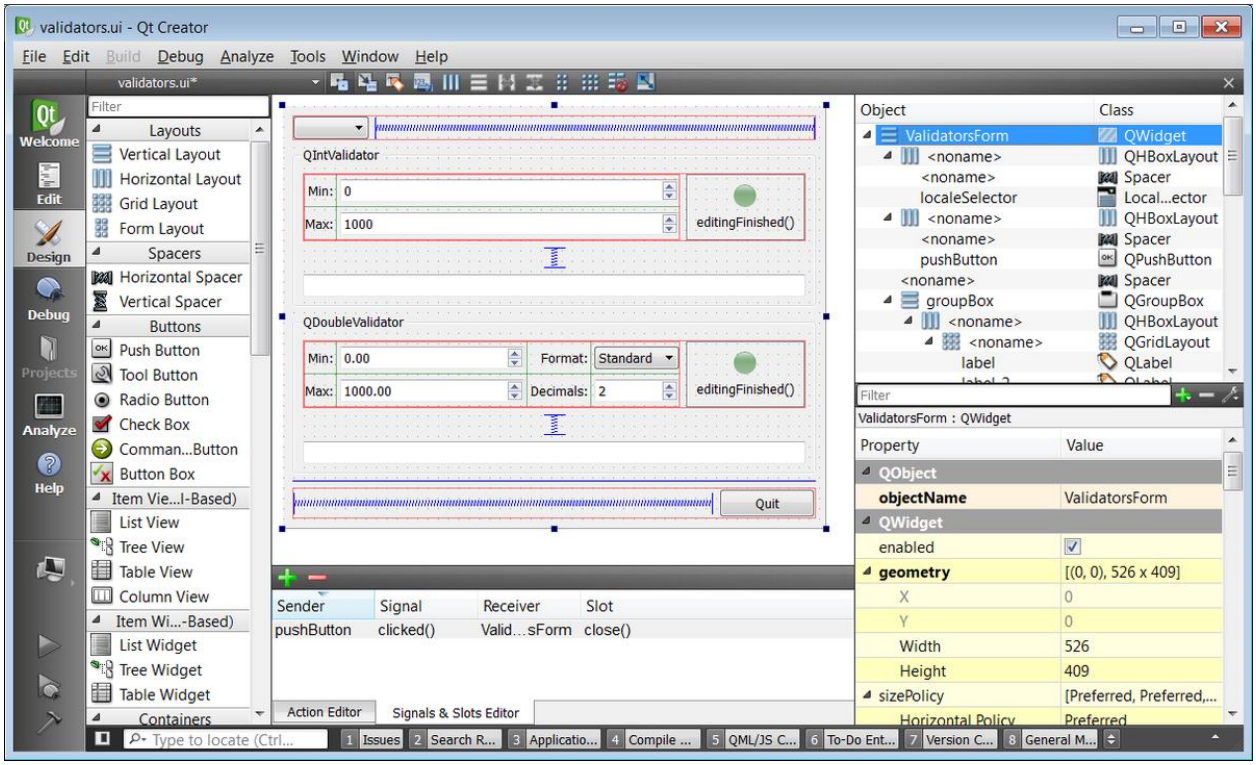



Рисунок 1.1 – Інтерфейс програми Qt

Windows Forms – це платформа інтерфейсу користувача для створення класичних додатків Windows. Вона забезпечує один з найефективніших способів створення класичних програм за допомогою візуального конструктора в Visual Studio. Такі функції, як розміщення візуальних елементів керування шляхом перетягування, полегшують створення класичних додатків. У Windows Forms можна розробляти графічно складні програми, які просто розгортати, оновлювати і з якими зручно працювати як в автономному режимі, так і в мережі. Програми Windows Forms можуть отримувати доступ до локального обладнання та файлової системи комп'ютера, на якому працює програма. Windows Forms представляє собою набір керованих бібліотек, які спрощують виконання стандартних завдань,

таких як читання з файлової системи і запис в неї. За допомогою середовища розробки, наприклад Visual Studio, можна створювати інтелектуальні клієнтські програми Windows Forms, які відображають інформацію, запитують введення користувача і взаємодіють з віддаленими комп'ютерами по мережі.

Перевагами програмного застосунку застосунку є відкритий код, отже цей застосунок є безплатним та велика кількість документації. Також цей програмний застосунок має відносно простий інтерфейс користування

Недоліками цього програмного застосунку є необхідність застосування IDE “Visual Studio”, яка не є безплатною IDE, а безплатна версія має обмежені властивості. Головним недоліком цього програмного застосунку є важкий етап підключення та першого запуску продукту, який використовує цю технологію.

Інтерфейс застосунку зображено на рисунку 1.2.

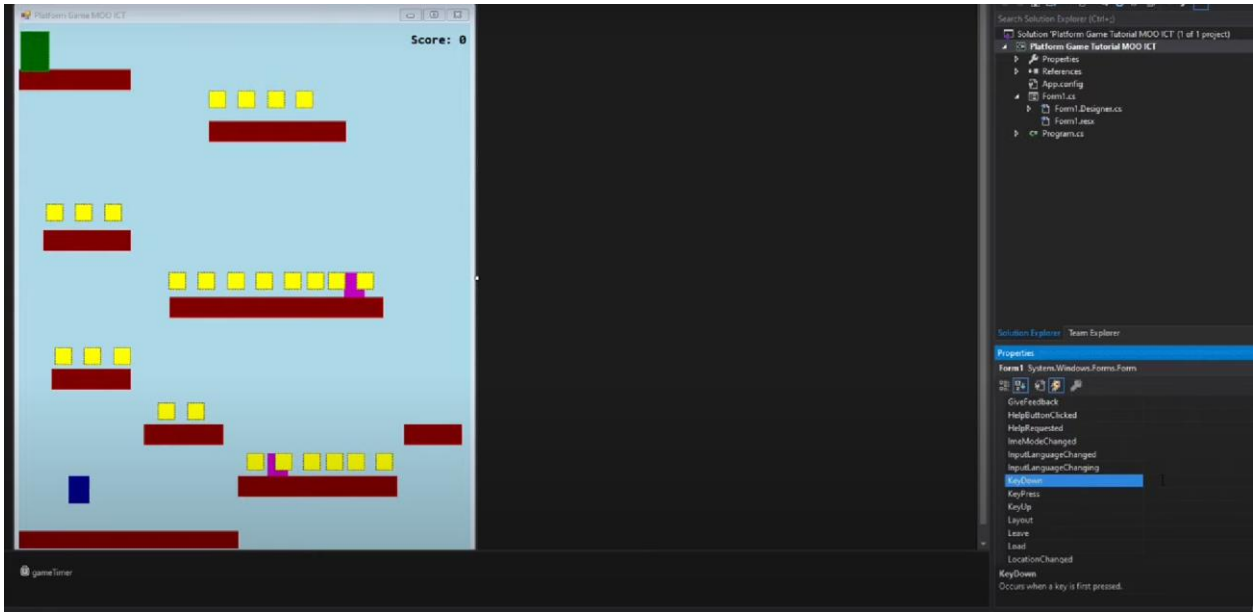



Рисунок 1.2 – Інтерфейс програми WinForms

GIMP Toolkit, відомий також як GTK, є вільною і відкритою графічною бібліотекою, що використовується для розробки графічних інтерфейсів користувача. Основне призначення GTK полягає в тому, щоб забезпечити розробникам потужні інструменти для створення інтерактивних і зручних інтерфейсів програм.

Переваги GTK: GTK є вільною і відкритою технологією, що означає, що його можна використовувати, змінювати і розповсюджувати його

безкоштовно. Це надає розробникам широкі можливості та гнучкість. GTK підтримує різноманітні платформи, такі як Linux, Windows і macOS, що дозволяє розробникам створювати багатоплатформові програми з мінімальними зусиллями. GTK надає багато вбудованих інструментів і елементів керування, що спрощують розробку графічних інтерфейсів. Ви можете легко створювати вікна, кнопки, поля введення, списки та інші елементи. GTK підтримує багато мов програмування, включаючи C, C++, Python, Ruby і багато інших. Це дає можливість розробникам використовувати їх улюблену мову для розробки програм з використанням GTK.

Незважаючи на свої переваги, GTK також має кілька недоліків: GTK вимагає від розробників дуже високого рівня знань. Іноді він може бути складним для освоєння, оскільки має дещо складну архітектуру. GTK пропонує високорівневий інтерфейс, що дозволяє швидше розробляти програми. Однак, це може обмежити контроль над деталями і бути обтяжливим у випадках, коли потрібні специфічні модифікації і налаштування. Розробка програм з використанням GTK може вимагати деяких залежностей від інших бібліотек і пакетів. Це може призвести до збільшення розміру програми і складності розгортання. Візуальний стиль і зовнішній вигляд GTK може бути обмежений і не завжди відповідати сучасним тенденціям дизайну інтерфейсів.

В цілому, GTK є потужним інструментом для розробки графічних інтерфейсів з великою кількістю можливостей, але використання його повинно бути розглянуто в контексті конкретних потреб та навичок розробника.

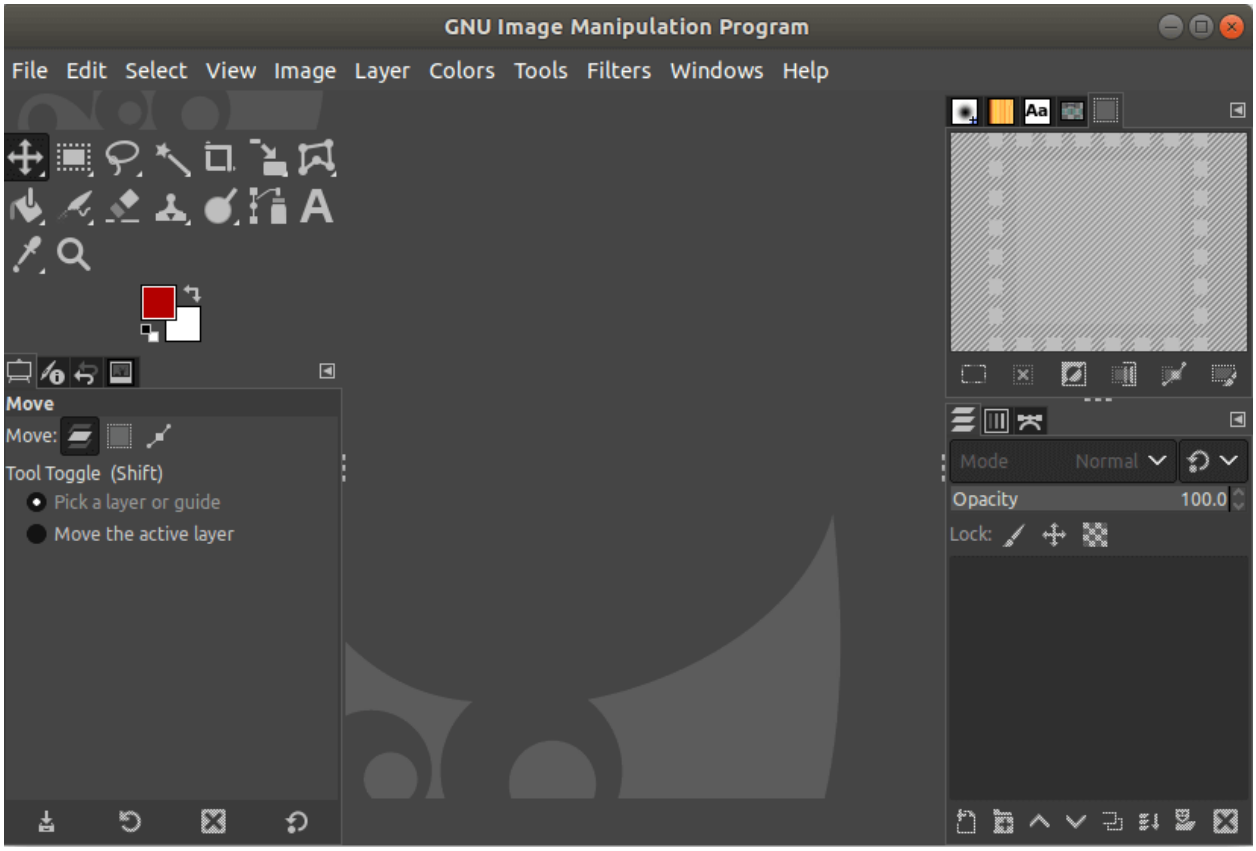



Рисунок 1.3 – Інтерфейс програми GTK

wxWidgets є інструментарієм розробки прикладного програмного забезпечення (ППЗ), який надає можливості для створення кросплатформних графічних інтерфейсів користувача. Ось деякі переваги та недоліки wxWidgets:

Переваги wxWidgets: wxWidgets підтримує багато операційних систем, включаючи Windows, macOS, Linux та багато інших. Це дозволяє розробникам створювати додатки, які працюють на різних платформах, з використанням одного і того ж коду. wxWidgets розповсюджується під ліцензією wxWindows, яка є вільною і відкритою. Це означає, що ви можете використовувати, змінювати і розповсюджувати wxWidgets безкоштовно, а також ви можете змінювати вихідний код під свої потреби. wxWidgets має добре документовану бібліотеку з великою кількістю прикладів і посилань на довідкові матеріали. Це полегшує розробку і дозволяє швидко зрозуміти функціональність і можливості бібліотеки.

Недоліки wxWidgets: Потребує додаткових залежностей: wxWidgets вимагає встановлення додаткових залежностей на різних операційних системах. Це може вплинути на розмір кінцевого додатку і складність його розгортання. Обмежена підтримка для деяких платформ: Незважаючи на те, що wxWidgets підтримує багато операційних систем, може бути обмежена підтримка для певних платформ або конкретних функцій, що можуть призвести до обмежень в розробці деяких додатків. Нижча продуктивність порівняно з деякими альтернативами: У порівнянні з деякими іншими інструментаріями розробки ГІП, wxWidgets може мати меншу продуктивність, особливо при великій кількості елементів керування або складних графічних операцій. Складність вивчення: wxWidgets може бути складним для вивчення для новачків або розробників, які не мають попереднього досвіду роботи з ним. Він вимагає певного часу і зусиль для освоєння його основних концепцій та структури.

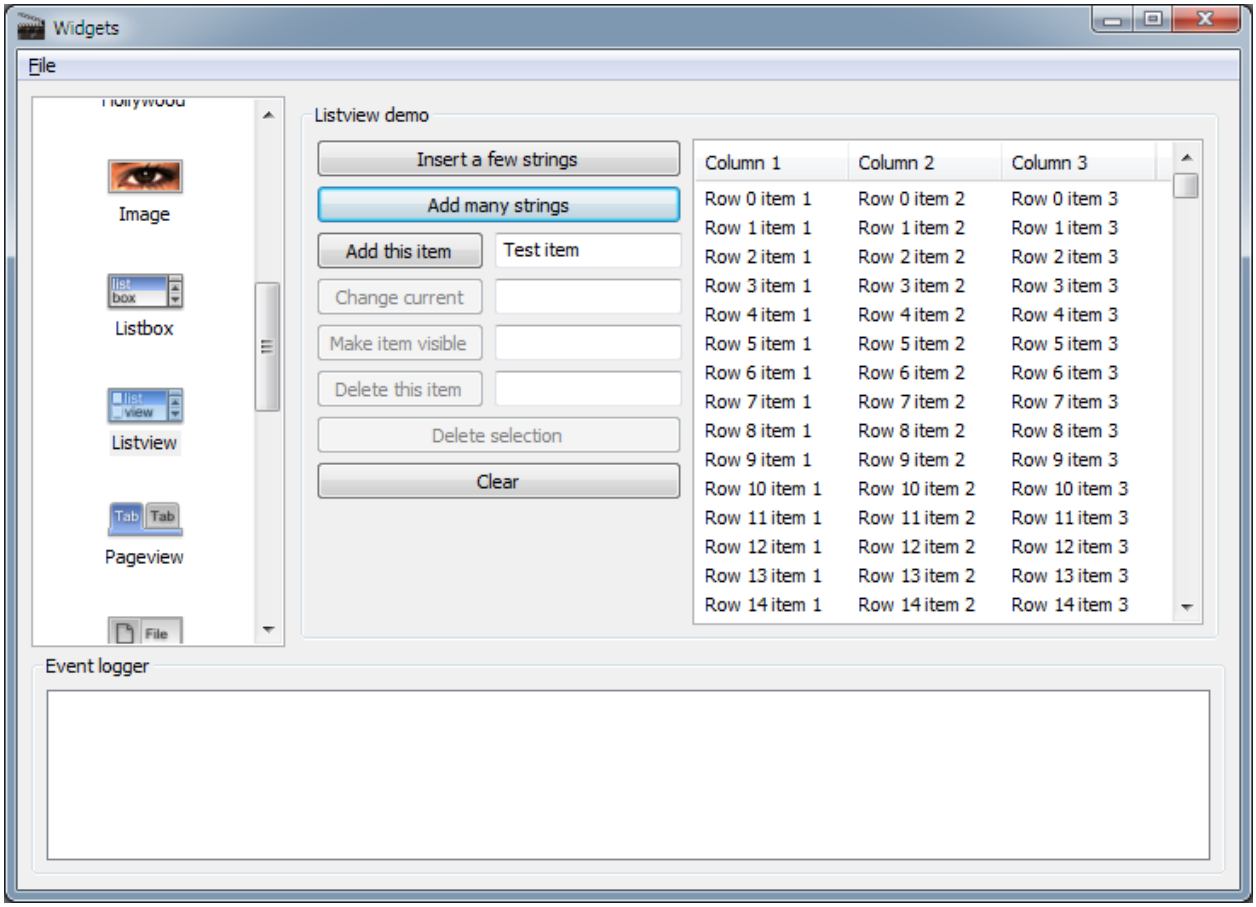



Рисунок 1.4 – Інтерфейс програми wxWidgets

Для порівняння курсової роботи з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

| Функціонал | Дипломний проєкт (CppUI) | Qt | WinForms | GTK | wxWidgets | Пояснення |
|---|---|---|---|---|---|---|
| Обробка подій/ управління контролером | + | + | + | + | + | |
| Налаштування параметрів віджетів | + | + | + | + | + | |
| Програмування сценаріїв реакції на подію | + | + | + | + | + | |
| Підтримка власної мови розмітки | + | - | - | - | - | |
| Можливість конструювати ГІК за допомогою UI | - | + | - | + | + | |

Продовження таблиці 1.1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Підтримка декількома типами ОС | - | + | - | - | + | + | |

## 1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є парсинг мови розмітки та створення повноцінного графічного інтерфейсу користувача на основі параметрів, прочитаних з файлу розмітки, більше функцій можна побачити на рисунку 1.4.

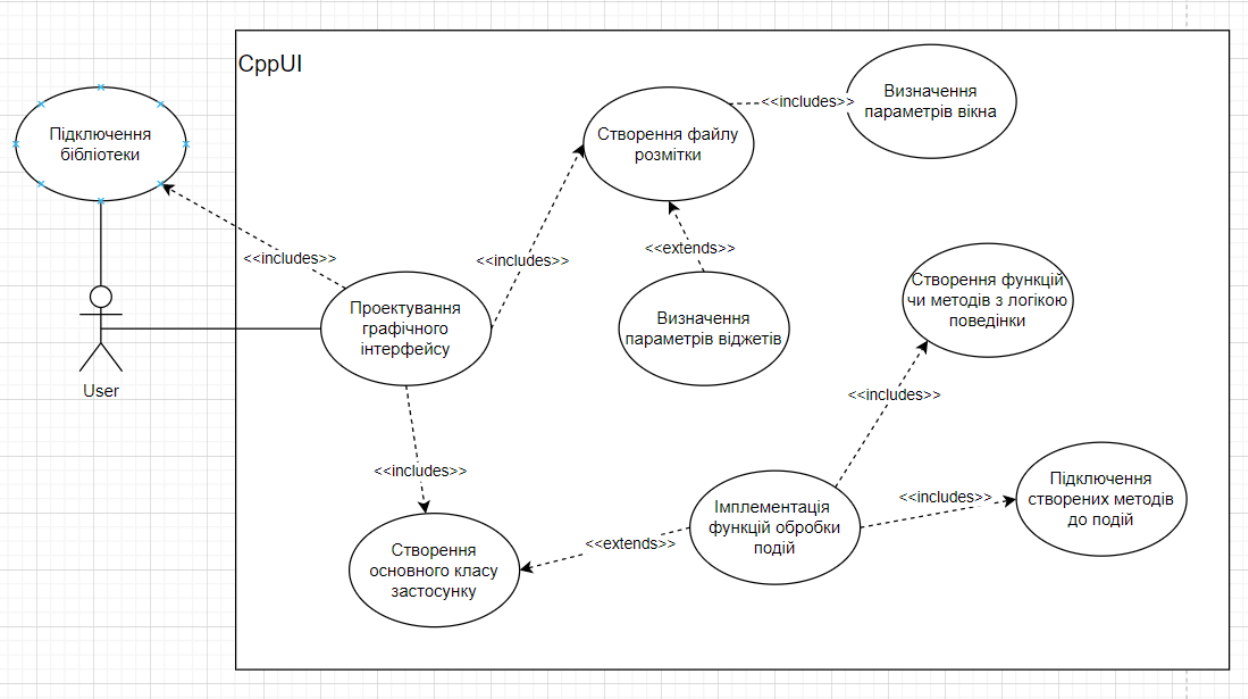



Рисунок 1.5 –Діаграма варіантів використання

В таблицях 1.2 - 1.10 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

| | |
|---|---|
| Use case name | Проектування графічного інтерфейсу користувача |
| Use case ID | UC-01 |
| Goals | Спроектувати графічний інтерфейс користувача |
| Actors | Користувач |

Продовження таблиці 1.2

| | |
|---|---|
| Trigger | Користувач бажає використати розроблену бібліотеку та мову розмітки для проектування графічного інтерфейсу користувача |
| Pre-conditions | - |
| Flow of Events | Користувач ознайомлюється з синтаксисом розробленої мови розмітки та бібліотеки. Відповідно до всіх правил створює файл розмітки для проектування інтерфейсу, за бажанням додає функції обробки подій натиску на кнопки |
| Extension | |
| Post-Condition | Графічний інтерфейс користувача створений |

Таблиця 1.3 - Варіант використання UC-2

| | |
|---|---|
| Use case name | Підключення бібліотеки |
| Use case ID | UC-02 |
| Goals | Підключити API бібліотеки для подальшого проектування ГІК та імплементації функцій обробки подій |
| Actors | Користувач |
| Trigger | Користувач підключає бібліотеку, використовуючи інструменти мови програмування C++ |
| Pre-conditions | - |
| Flow of Events | Користувач підключає бібліотеку до нового або вже створеного проекту |
| Extension | Якщо бібліотека підключена некоректно, при буде показуватись помилка компіляї |
| Post-Condition | Користувач підключив подійно-орієнтовану бібліотеку для проектування графічного інтерфейсу користувача |

Таблиця 1.4 - Варіант використання UC-3

| | |
|---|---|
| Use case name | Створення файлу розмітки |
| Use case ID | UC-03 |
| Goals | Створити файл розмітки, в якому задані основні параметри бажаного інтерфейсу |
| Actors | Користувач |
| Trigger | Користувач розробляє графічний інтерфейс, використовуючи розроблену мову розмітки |
| Pre-conditions | - |
| Flow of Events | Користувач створює новий файл розмітки, відвовідно до синтаксису розробленої мови розмітки, задає параметри графічного інтерфейсу, видповідно до своїх вимог |
| Extension | Якщо файл розмітки заданий невідповідно, то у відповідному файлі виведеться повідомлення про помилки парсингу, а некоректно задані параметри проігноруються, їх значення будуть такими, як задано за замовчуванням |
| Post-Condition | Користувач створив файл розмітки графічного інтерфейсу |

Таблиця 1.5 - Варіант використання UC-4

| | |
|---|---|
| Use case name | Імплементація функцій обробки подій |
| Use case ID | UC-04 |
| Goals | Передбачити поведінку графічного інтерфейсу відповідно до взаємодії з його параметрами |
| Actors | Користувач |
| Trigger | Користувач використовує у спроектованому інтерфейсі певні кнопки, і бажає додати функціонал, що буде спрацьовувати у випадку натискання на них |
| Pre-conditions | Користувач використовує кнопки при проектуванні графічного інтерфейсу |

Продовження таблиці 1.5

| | |
|---|---|
| Flow of Events | Користувач реалізовує функції, які відповідають за поведінку системи у випадку натискання кнопки. |
| Extension | |
| Post-Condition | Користувач реалізував функціонал, що буде виконуватися, кнопку буде натиснуто |

Таблиця 1.6 - Варіант використання UC-5

| | |
|---|---|
| Use case name | Задання параметрів основного вікна |
| Use case ID | UC-05 |
| Goals | Задати параметри основного вікна |
| Actors | Користувач |
| Trigger | Користувач бажає задати значення параметри основного вікна, що відмінні від значень, що встановлюються за замовчуванням |
| Pre-conditions | Підключено бібліотеку |
| Flow of Events | У файлі розмітки користувач може задати параметри головного вікна такі, як заголовок, висоту та ширину вікна |
| Extension | Якщо користувач не бажає задавати значення для параметрів вікна, то будуть використані параметри, що встановлюються за замовчуванням |
| Post-Condition | Задано параметри запуску головного вікна |

Таблиця 1.7 - Варіант використання UC-56

| | |
|---|---|
| Use case name | Визначення параметрів віджетів |
| Use case ID | UC-06 |
| Goals | Додати віджет на головне вікно та задати параметри віджету |
| Actors | Користувач |
| Trigger | Користувач бажає додати віджет до головного вікна та задати параетру цьому віджету |

Продовження таблиці 1.7

| | |
|---|---|
| Pre-conditions | Підключено бібліотеку |
| Flow of Events | Користувач створює розмітку віджета відповідно до синтаксису мову розмітки |
| Extension | |
| Post-Condition | На головне вікно додано віджет |

Таблиця 1.8 - Варіант використання UC-7

| | |
|---|---|
| Use case name | Створення основного класу застосунку |
| Use case ID | UC-07 |
| Goals | Спроектувати графічний інтерфейс користувача |
| Actors | Користувач |
| Trigger | Користувач бажає використати розроблену бібліотеку та мову розмітки для проектування графічного інтерфейсу користувача |
| Pre-conditions | - |
| Flow of Events | Користувач створює основний клас застосунку відповідно до інструкцій, наданих в додатках |
| Extension | |
| Post-Condition | Створено основний клас застосунку |

Таблиця 1.9 - Варіант використання UC-8

| | |
|---|---|
| Use case name | Створення методів чи функцій з поведінкою |
| Use case ID | UC-08 |
| Goals | Додати обробку дії чи події |
| Actors | Користувач |
| Trigger | Користувач бажає додати обробку дії чи події |
| Pre-conditions | Створено основний клас застосунку |
| Flow of Events | Відповідно до інструкцій, наданих в додатках, користувач додає обробку дій чи подій у програмному застосунку |

Продовження таблиці 1.9

| | |
|---|---|
| Extension | |
| Post-Condition | Буде додана обробка дії чи події |

Таблиця 1.10 - Варіант використання UC-9

| | |
|---|---|
| Use case name | Підключення створених методів до подій |
| Use case ID | UC-09 |
| Goals | Підключити обробку дій чи подій до функціоналу ПЗ |
| Actors | Користувач |
| Trigger | Користувач бажає підключити обробку дій чи подій до функціоналу ПЗ |
| Pre-conditions | Користувач створив обробку дій чи подій |
| Flow of Events | Користувач підключає обробку дій чи подій до функціоналу ПЗ відповідно інструкціям, наданих в додатках |
| Extension | |
| Post-Condition | Підключено обробку дій чи подій до фунціоналу ПЗ |

### 1.4.1 Розроблення функціональних вимог

Програмне забезпечення складається з двох компонентів, кожен з яких має свій набір функцій. Загальна модель вимог представлена в таблиці 1.11, Матриця трасування вимог може бути знайдена на . У загальній моделі вимог функціональним вимогам, що описують головні функції застосунку, було присвоєно пріоритет 3, вимогам, що описують функції, без яких програмний застосунок може працювати, але які значно покращують користувацький досвід, було присвоєно пріоритет 2, а всім іншим - пріоритет 1. Оскільки застосунок працює локально та зберігає дані лише на комп’ютері користувача, ризик відсутній для всіх вимог і позначений як "Undefined".

Таблиця 1.11 – Загальна модель вимог

| Title ID | Full Description | Code | Priority | Risk | Status |
|---|---|---|---|---|---|
| 1 | Проектування графічного інтерфейсу користувача | FR_1 | 3 | Undefined | Draft |
| 1.1 | Задання параметрів головного вікна | FR_2 | 3 | Undefined | Draft |
| 1.1.1 | Задання розмірів вікна | FR_3 | 3 | Undefined | Draft |
| 1.2 | Задання параметрів віджетів | FR_4 | 1 | Undefined | Draft |
| 1.2.1 | Задання розмірів віджетів | FR_5 | 1 | Undefined | Draft |
| 1.2.2 | Задання розміщення віджетів | FR_6 | 3 | Undefined | Draft |
| 1.2.3 | Задання специфічних параметрів віджетів | FR_7 | 3 | Undefined | Draft |
| 2 | Обробка дій чи подій | FR_8 | 3 | Undefined | Draft |
| 2.1 | Обробка дій чи подій натискання кнопки клавіатури | FR_9 | 3 | Undefined | Draft |
| 2.2 | Обробка дій чи подій натискання комп’ютерної миші | FR_10 | 1 | Undefined | Draft |
| 3 | Створення віджетів | FR_11 | 1 | Undefined | Draft |
| 3.1 | Створення віджету Label | FR_12 | 3 | Undefined | Draft |
| 3.2 | Створення віджету Button | FR_13 | 3 | Undefined | Draft |
| 3.3 | Створення віджету radioButton | FR_14 | 3 | Undefined | Draft |

Продовження таблиці 1.11

| | | | | | |
|---|---|---|---|---|---|
| 3.4 | Створення віджету CheckBox | FR_15 | 3 | Undefined | Draft |
| 3.5 | Створення віджету EditLine | FR_16 | 3 | Undefined | Draft |
| 3.6 | Створення віджету MultiTextField | FR_17 | 3 | Undefined | Draft |
| 3.7 | Створення віджету NumberBox | FR_18 | 1 | Undefined | Draft |
| 3.8 | Створення віджету Text | FR_19 | 1 | Undefined | Draft |
| 4 | Створення логіки обробки дій та подій | FR_20 | 3 | Undefined | Draft |
| 5 | Створення “scopes” віджетів | FR_21 | 3 | Undefined | Draft |
| 6 | Обробка необхідних операндів мовою розмітки | FR_22 | 3 | Undefined | Draft |
| 6.1 | Обробка операнду “:” | FR_23 | 3 | Undefined | Draft |
| 6.2 | Обробка операнду “/” | FR_24 | 1 | Undefined | Draft |
| 6.3 | Обробка операнду “]” | FR_25 | 1 | Undefined | Draft |
| 6.4 | Обробка операнду “[” | FR_26 | 3 | Undefined | Draft |
| 6.5 | Обробка операнду “\|” | FR_27 | 3 | Undefined | Draft |
| 6.6 | Обробка операнду “#” | FR_28 | 3 | Undefined | Draft |
| 7 | Обробка ключових слів мовою розмітки | FR_28 | 3 | Undefined | Draft |
| 7.1 | Обробка ключового слова “Button” | FR_30 | 1 | Undefined | Draft |
| 7.2 | Обробка ключового слова “CheckBox” | FR_31 | 1 | Undefined | Draft |

Продовження таблиці 1.11

| | | | | | |
|---|---|---|---|---|---|
| 7.3 | Обробка ключового слова “EditLine” | FR_32 | 3 | Undefined | Draft |
| 7.4 | Обробка ключового слова “Label” | FR33 | 3 | Undefined | Draft |
| 7.5 | Обробка ключового слова “NumberBox” | FR_34 | 3 | Undefined | Draft |
| 7.6 | Обробка ключового слова “RadioButton” | FR_35 | 1 | Undefined | Draft |
| 7.7 | Обробка ключового слова “Text” | FR_36 | 1 | Undefined | Draft |
| 7.8 | Обробка ключового слова “sizeGorz” | FR_37 | 3 | Undefined | Draft |
| 7.9 | Обробка ключового слова “sizeVert” | FR_38 | 3 | Undefined | Draft |
| 7.11 | Обробка ключового слова “positionVert” | FR_39 | 3 | Undefined | Draft |
| 7.12 | Обробка ключового слова “positionGorz” | FR_40 | 3 | Undefined | Draft |
| 7.13 | Обробка ключового слова “name” | FR_41 | 1 | Undefined | Draft |
| 7.14 | Обробка ключового слова “clicable” | FR_42 | 1 | Undefined | Draft |
| 7.15 | Обробка ключового слова “doubleClicable” | FR_43 | 3 | Undefined | Draft |
| 7.16 | Обробка ключового слова “defaultValue” | FR_44 | 3 | Undefined | Draft |
| 7.17 | Обробка ключового слова “multiline” | FR_45 | 3 | Undefined | Draft |

Продовження таблиці 1.11

| | | | | | |
|---|---|---|---|---|---|
| 7.18 | Обробка ключового слова “minValue” | FR_46 | 3 | Undefined | Draft |
| 7.19 | Обробка ключового слова “maxValue” | FR_47 | 3 | Undefined | Draft |

Таблиця 1.12 – Матриця трасування

| | UC-01 | UC-02 | UC-03 | UC-04 | UC-05 | UC-06 | UC-07 | UC-08 | UC-09 |
|---|---|---|---|---|---|---|---|---|---|
| FR-1 | + | | | + | | | + | | + |
| FR-2 | | | + | + | | + | | + | |
| FR-3 | | + | | | | | | | |
| FR-4 | + | + | | + | | | + | | + |
| FR-5 | | | + | | + | | | + | |
| FR-6 | + | | | + | | + | | | |
| FR-7 | | | + | | + | + | + | | + |
| FR-8 | + | + | | + | | | | + | |
| FR-9 | | | | | | + | + | | + |
| FR-10 | + | | + | | + | + | | | + |
| FR-11 | | + | | | | | | | |
| FR-12 | + | | | + | | + | + | | |
| FR-13 | | | + | | + | | | | |
| FR-14 | | + | | + | | + | + | + | + |
| FR-15 | | | | | | | | | |
| FR-16 | + | + | | + | | + | + | | |
| FR-17 | + | | + | | | | + | + | + |
| FR-18 | | + | | + | + | | | | |
| FR-19 | + | | | + | | | + | | |
| FR-20 | + | + | + | | | + | | + | + |
| FR-21 | | | | | + | | | | |
| FR-22 | | | + | | | + | | + | + |
| FR-23 | + | + | | + | | + | | | |
| FR-24 | | | + | | | | + | | |
| FR-25 | | + | | | | | | | + |
| FR-26 | + | | | | + | | | | |
| FR-27 | | + | + | | | | + | + | |
| FR-28 | | | | + | | + | | | |
| FR-29 | + | | | + | | + | | | |
| FR-30 | | + | | | | + | | + | |
| FR-31 | | | + | + | | + | + | | + |
| FR-32 | | | | | | | | | |
| FR-33 | + | + | | + | | + | | + | + |
| FR-34 | + | | | | + | | + | | |
| FR-35 | | | + | | + | + | | + | + |
| FR-36 | | + | | + | | | + | | + |

Продовження таблиці 1.12

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| FR-37 | + | | | + | + | | | | |
| FR-38 | | | + | | | + | | | |
| FR-39 | + | + | | + | + | | | + | + |
| FR-40 | | + | | + | | | + | | + |
| FR-41 | + | | + | | + | | | + | |
| FR-42 | + | | + | | + | + | | | |
| FR-43 | | + | | + | | + | | + | |
| FR-44 | | | + | | + | | | | |
| FR-45 | + | | | + | | | + | | + |
| FR-46 | | | + | | + | | | + | |
| FR-47 | + | + | | + | | + | + | | + |

### 1.4.2 Розроблення нефункціональних вимог

Розроблена мова розмітки повинна бути:

– інтуїтивно зрозумілою для користувача;

– бути простою в використанні;

– доступною для редагування в будь якому текстовому редакторі;

– мати чіткі вимоги до оформлення параметрів всередині файлу.

Розроблена бібліотека для проектування графічних інтерфейсів повинна бути:

– інтуїтивно зрозумілою та зручною для реалізації графічного інтерфейсу;

– бути простою в використанні.

– додаток повинен коректно відображатися при різних розширеннях екрану;

#### 1.4.2.1 Вимоги до безпеки

Передбачити контроль введення інформації та захист від некоректних дій користувача.

1.4.2.2 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core Pentium;
– об‘єм ОЗП: 2 Гб;
– графічний прискорювач RTX 560.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i3 або вище;
– об‘єм ОЗП: 4 Гб або вище;
– графічний прискорювач RTX 960 або вище.

1.4.2.3 Вимоги до складу і параметрів технічних засобі

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows(Windows7 та вище).

1.4.2.4 Вимоги до мови розмітки

Мова розмітки повинна мати чіткий, структурований і легко читаємий синтаксис. Мова розмітки повинна мати широкий набір тегів, які можуть розмічати різні типи елементів і структур в документі.Мова розмітка повинна бути зрозумілою та логічною для комп'ютерів і людей. Вона має бути без синтаксичних помилок. Мова розмітки повинна мати простий та легко зрозумілий синтаксис, що дозволяє розробникам швидко вивчати та використовувати її. Вона також повинна бути зручною для редагування та підтримки. Мова розмітки може бути придатною для використання в різних та на різних платформах. Вона повинна мати широку підтримку серед різних програмних засобів та бути незалежною від конкретної платформи. Мова розмітка повинна мати можливість розширення за допомогою нових елементів, атрибутів та правил. Це дозволяє отримати змінні потреби розробників та забезпечити сумісність з майбутніми версіями мови. Мова

розмітки повинна перевіряти вхідний файл на наявність помилок та, після виявлення таких, припинити парсинг розмітки та сповістити користувача про тип помилки та місце, де ця помилка була здопущена. При цьому, підтримується основні дві команди розмітки – команда створення віджету та команда зміни параметру віджету. Команда створення ввіджету повинна мати наступний вигляд: “| widget [”, команда зміни віджета повинна мати наступний вигляд “/ param : value”. Зміна параметрів повинна бути доступна лише у “scope” віджета. Мова розмітки повинна підтримувати наступні операнди: “:”, “/”, “]”, “[”,“|”, “#”. “:” – повідомляє, що після цього операнда буде введено значення параметра віджету. “/” – означає, що в цьому рядку буде змінено параметр віджету. “]” – закриває “scope” віджету. “[” – відкриває “scope” віджета “/”. “|” – означає, що в цьому рядку буде сторено новий віджет. “#” – повідомляє, що цей рядок є кометарем і його не потрібно опрацьовувати. Мова розмітки повинна обробляти ключові слова: “Button”, “CheckBox”, “EditLine”, “Label”, “NumberBox”, “RadioButton”, “Text”, “sizeVert”, “sizeGorz”, “positionVert”, “positionGorz”, “name”, “clicable”, “doubleClicable”, “defaultValue”, “multiline”, “minValue”, “maxValue”. Мова розмітки повинна перевіряти наявність параметрів у віджета при запиті зміни параметра віджета користувачем. Мова розмітки повинна перевіряти можливість створення віджета у скоупі іншого віджета. Мова розмітки повинна перевіряти структуру команд(створення віджета, зміни параметрів). Мова розмітки повинна перевіряти введене користувачем значення для кожного параметра

## 1.5 Постановка задачі

Необхідно розробити подійно-орієнтовану бібліотеку для проектування графічного інтерфейсу користувача на основі власної мови розмітки. Власна мова розмітки буде використовуватися для опису параметрів віджетів. Ця мова повинна бути ефективною, зрозумілою та зручною для розробників. Для обробки розмітки необхідно створити парсер, який зможе аналізувати вхідну розмітку і розпізнавати віджети, їх параметри та взаємозв'язки між ними.

Парсер повинен бути надійним та швидким. Бібліотека повинна надавати можливість визначати обробники подій, які будуть виконуватися при взаємодії користувача з віджетами. Бібліотека повинна мати механізм для підключення хендлерів (методів чи функцій) до події, що дозволяє динамічно додавати та видаляти обробники для певних подій. Результатом проекту буде готова подійно-орієнтована бібліотека з власною мовою розмітки, яка дозволяє легко та зручно інтегрувати себе в будь-який консольний застосунок таспроектувати там графічний інтерфейс користувача.

Метою розробки є полегшення розробки програмного забезпечення з графічним інтерфейсом користувача на мові програмування C++, за рахунок розробки бібліотеки для проектування інтерфейсів користувача з використанням зручної та інтуїтивно простої для користувача мови розмітки.

Розробка призначена для: для всіх програмістів, що займаються проектування інтерфейсів користувача на мові програмування C++.

Розробка ставить перед собою досягнення наступних цілей:

– полегшення розробки користувацьких інтерфейсів на мові прогамування C++, за рахунок розробки зручної та інтуїтивно простої мови розмітки.

– полегшення розробки ПЗ з використанням ГІК за рахунок спрощення підключення та програмування віджетів

Програмне забезпечення повинно забезпечувати виконання наступних функцій:

– розроблена бібліотека повинна дозволяти користувачу на специфічній мові розмітки спроектувати інтерфейс користувача.

– бібліотека повинна забезпечувати можливість обробки стандартного набору подій.

– за допомогою бібліотеки, а саме мови розмітки, можна налаштовувати інтерфейс користувача, розмір вікна.

– за допомогою мови розмітки можна стилізувати параметри всіх віджетів.

– за допомогою бібліотеки можна створити програмне забезпечення, з своєю логікою обробки подій.

– розробка є бібліотекою, тому не передбачає графічного інтерфейсу користувача

Висновки до розділу

В першому розділі курсової роботи було детально проаналізовано предметну область, що стосується ідеї курсової роботи, а також проаналізовано уже наявні IT-проекти, що вирішують схожі задачі. Головним аналогом, що лідирує на ринку проектування графічних інтерфейсів є фреймфорк Qt. Але, незважаючи на популярність даної бібліотеки, її освоєння для новачків може бути ненайлегшим завданням, оскільки це рішення покриває не тільки створення візуального представлення програми, а й широкий спектр потреб розробників для реалізації складних застосунків. Як висновок, використання цього фреймворку не завжди є найкращим рішенням через складність його освоєння. Нашим завданням є створення бібліотеки для проектування графічних інтерфейсів на основі власної мови розмітки, що дозволить пришвидшити розробку візуальних застосунків на мові програмування C++. Крім того, в першому розділі представлено діаграму варіантів використання, що містить в собі основні аспекти створення інтерфейсів за допомогою розроблених технологій. Також проаналізовано функціональні вимоги, і показано їх взаємозв’язок з варіантами використання у вигляді матриці трасування.

# 2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Моделювання та аналіз програмного забезпечення

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 2.1).

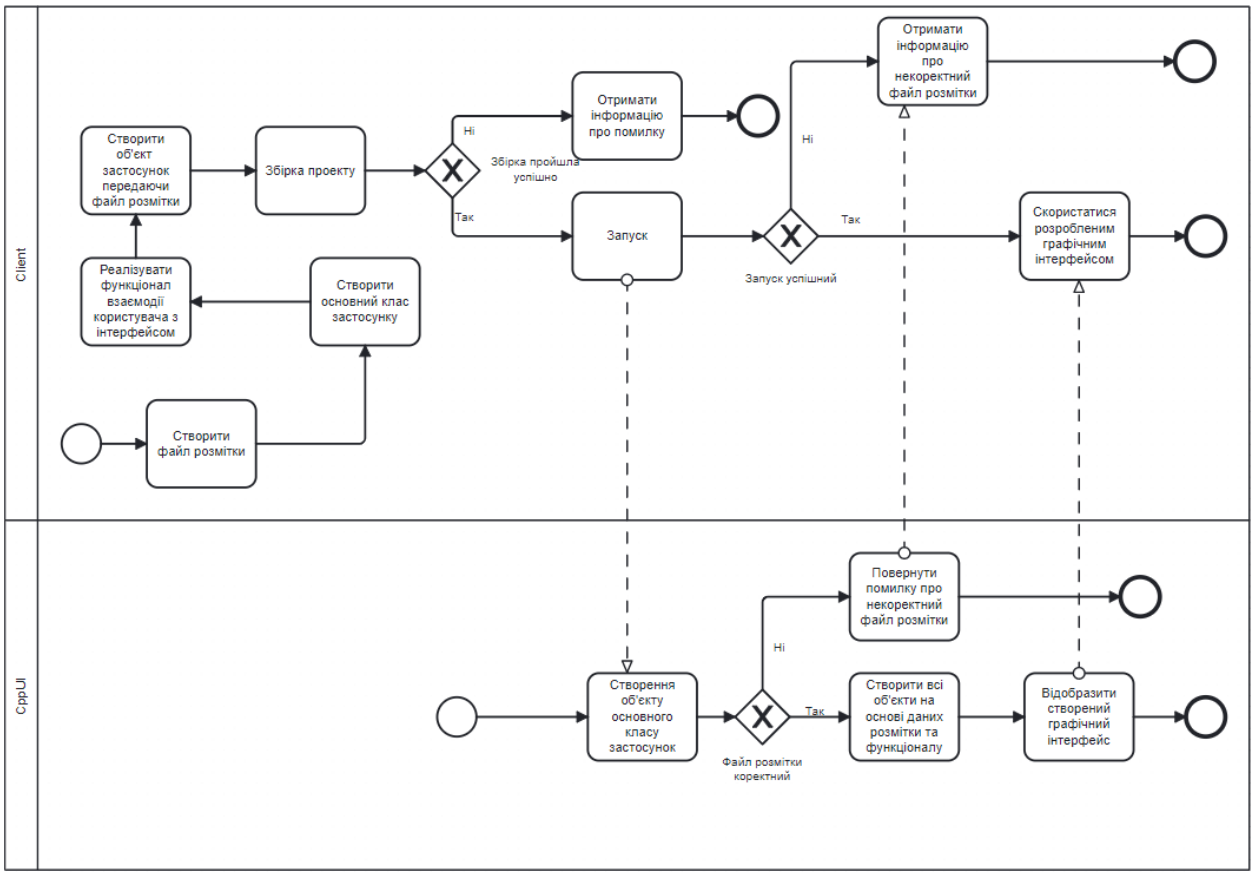



Рисунок 2.1 – Схема бізнес-процесу використання бібліотеки CppUI для створення графічного інтерфейсу користувача

У наступних абзацах представлений опис послідовності створення додатку з графічним інтерфейсом користувача.

Користувач повинен створити файл розмітки головного вікна застосунку. Файл розмітки повинен чітко визначити основні параметри вікна (заголовок, розміри, віджети, що повинні відображатись всередині вікна, іконку застосунку, меню вікна). Для кожного віджета (кнопка, поле вводу) користувач повинен також визначити основні параметри (розмір, позицію, здатність реагувати на взаємодії) у файлі розмітки.

Далі користувач повинен створити свій клас (наприклад, Application). Цей клас повинен бути дочірнім від класу BaseApplication, що є частиною CppUI. Дана вимога пояснюється тим, що клас BaseApplication реалізує необхідний інтерфейс для створення, відображення та взаємодії з віджетами, параметри яких описані в файлі розмітки.

В межах класу Application користувач повинен реалізувати необхідні методи, що відображають логіку поведінки застосунку при взаємодії з певними віджетами (наприклад, при натисканні на кнопку). Крім цього, користувач повинен коректно поєднати ці методи із сигналами (спеціальними методами всередині конкретних віджетів, що викликаються при тій чи іншій взаємодії користувача з об'єктом віджета).

Наступним кроком для користувача бібліотекою CppUI буде створення точки входу (функції main()). В тілі даної функції користувач повинен створити об'єкт класу Application та передати йому у конструктор шлях до файлу розмітки. Якщо користувач правильно реалізував клас Application, то функціонал батьківського класу (BaseApplication) створить необхідні віджети на основі даних з файлу розмітки під час запуску;

Під час збірки проекту, що використовує CppUI для створення графічного інтерфейсу користувача, відбудеться конфігурація та компіляція застосунку. Якщо користувач допустив певні помилки у налаштуванні параметрів проекту чи у написанні коду, він отримує інформацію про помилки. Для коректного налаштування проекту користувачу варто почитати інструкцію з розгортання бібліотеки, що буде описана в наступних розділах;

Після успішної збірки проекту, користувач може запустити свій застосунок, розроблений з використанням CppUI. На цьому етапі відбувається перевірка коректності написання файлу розмітки. Якщо користувач допустив помилки у формуванні файла розмітки, які роблять неможливим створення вікна, він отримає про це повідомлення на екран. Після цього розроблена користувачем програма закінчить свою роботу. У випадку, коли допущені помилки не критичні, і дозволяють створити вікно програми, користувач

отримує попередження, але робота його програми на цьому не завершиться, і користувач зможе нею скористатися та перевірити її роботу. Якщо ж файл розмітки заданий повністю коректно, застосунок з графічним інтерфейсом успішно відобразиться, функціонал буде повністю відповідати поведінці, яку визначив користувач. В такому випадку користувач зможе скористатися своїм застосунком чи представити його фінальному клієнту.

## 2.2 Архітектура програмного забезпечення

За основу архітектури програмного забезпечення був взятий шаблон монолітної архітектури. Діаграма архітектури бібліотеки зображена на рисунку 2.2.

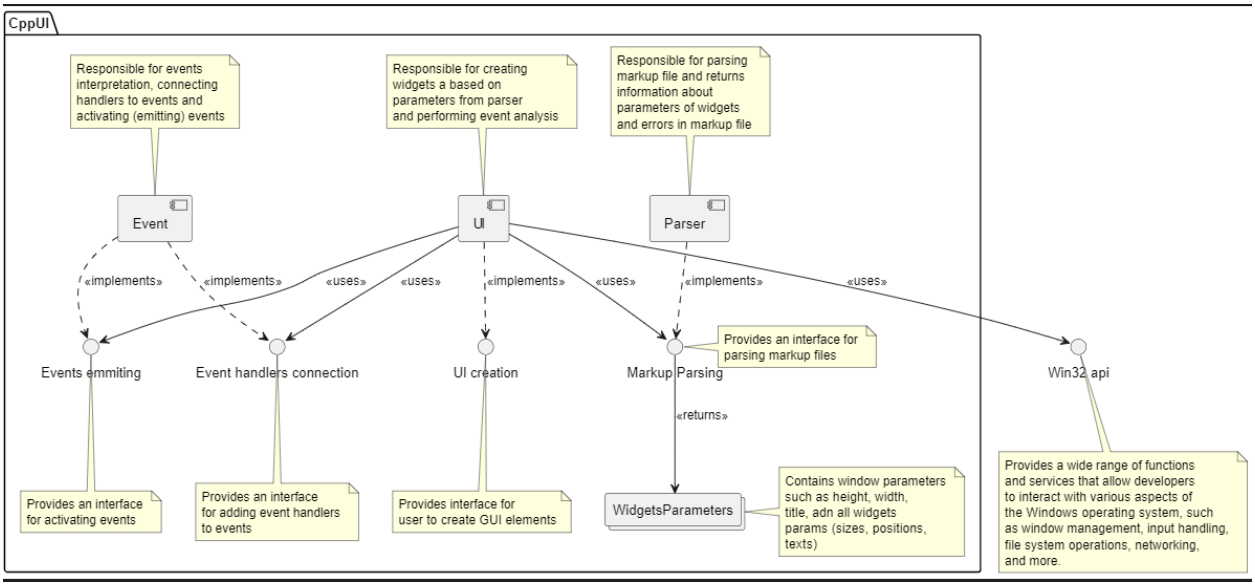



Рисунок 2.2 – Архітектура програмного забезпечення

За парсинг файлу розмітки відповідає компонент Markup. Ключовим завданням цього компонента є отримання на вхід шляху до файлу, де користувач описав параметри віджетів, які він очікує побачити на екрані застосунку, який він розробляє, та зчитування звідти всіх тих параметрів. Крім того, компонент Markup передбачає ймовірність отримання некоректних даних від користувача, тому такий варіант подій теж має очікувану поведінку. В такому випадку, компонент Markup поверне код помилки, а також відповідне повідомлення. Єдиним публічним інтерфейсом компонету Markup

є метод parsing(). Як результат роботи цього методу, компонент UI отримає необхідні параметри для створення вікна, віджетів.

Ще однією важливою частиною розробки є компонент Events, оскільки саме взаємодія користувача з графічним інтерфейсом (виникнення певних подій) і призводить до того, що користувач отримує користь від роботи з графічним інтерфейсом. Використовуючи базові принципи подійно-орієнтованої архітектури, а також власні допрацювання, було спроектовано даний компонент, що дає можливість реалізувати симуляцію виникнення певної події та викликати необхідну реакцію (функцію) у відповідь на подію (взаємодію користувача з інтерфейсом).

Компонент UI повинен представити користувачу інтерфейс, використовуючи який він зможе швидко і просто додати до свого застосунку графічний інтерфейс користувача. Тому, основними частинами BaseApplication є інтерфейс для створення віджетів (контролерів), адже саме на інформації з цих об’єктів буде будуватись логіка та складна поведінка кінцевого застосунку, інтерфейс для створення вікна, яке об’єднує в собі всі віджети, та надає доступ до управління їх параметрами, інтерфейс для створення основного класу застосунку, що потрібен для контролю за ресурсами застосунку та налаштування всіх необхідних параметрів з файлу розмітки.

Для реалізації вище описаного функціоналу UI компонент використовує Win32Api, Events, Markup компоненти.


Рисунок 2.3 – Діаграма класів компоненту UI

На рисунку 2.3 представлена діаграма класів частини програмного забезпечення, що відображає структуру сутностей для забезпечення можливості користування різними контролерами та обробки їх подій. Контролери представлені класами Button, RadioButton, SpinBox, Label, FieldEdit, CheckBox. Класи, об’єкти яких є членами інших класів: Text, Font, Size, Position. Event – клас, що надає користувачу інтерфейс для прив’язки

певних методів до певних подій. BaseAppication – базовий головний клас застосунку.

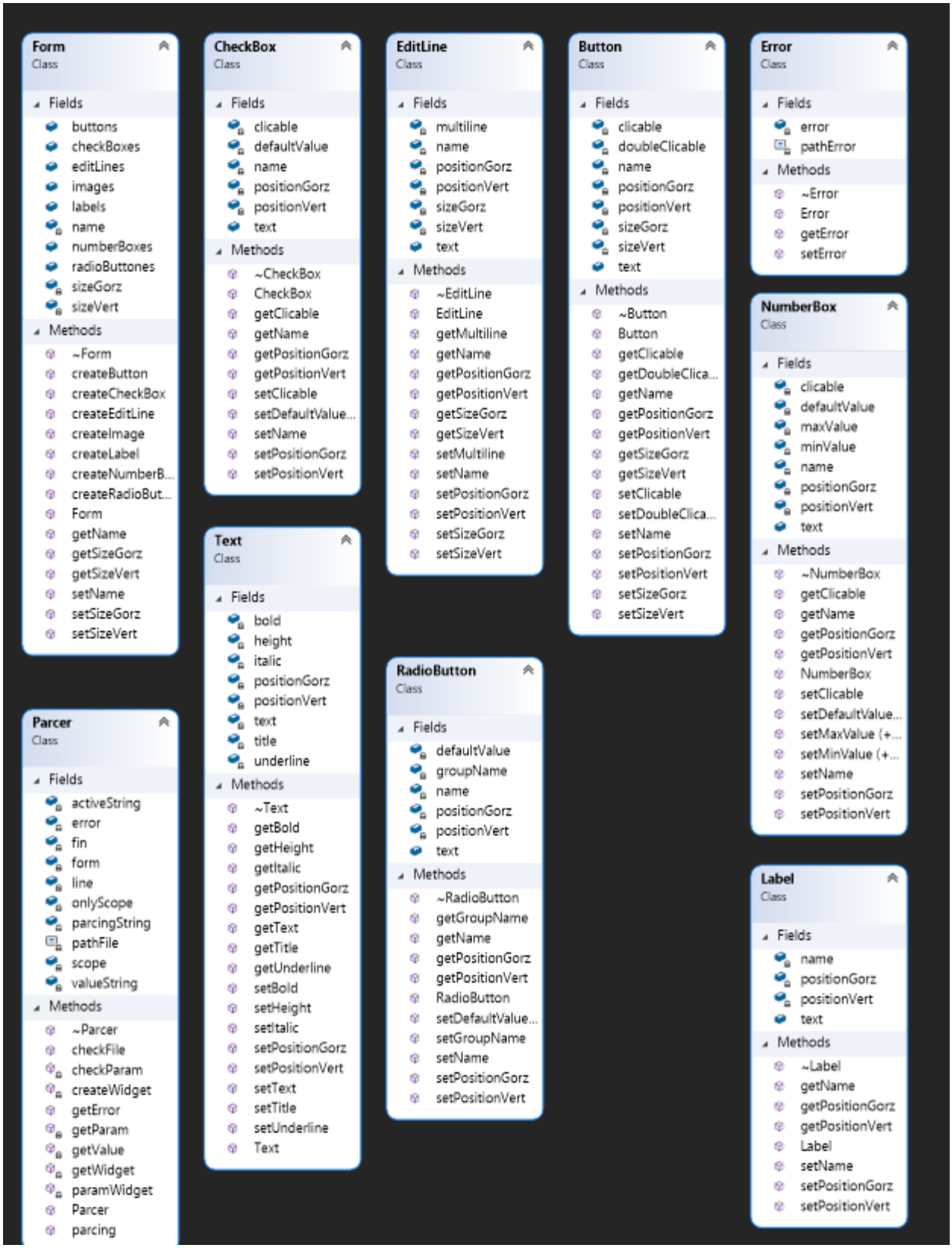



Рисунок 2.4 – Діаграма класів компоненту Markup

На рисунку 2.4 представлена діаграма класів компоненту Markup. Цей компонент дозволяє парсити файл розмітки та сторювати віджети, задавати цим віджетам параметри на основі отриманої інформації. Параметри кожного віджета мають свій клас, а саме: “Label”, “NumberBox”, “Button”, “RadioButton”, “Text”, “CheckBox”, “Editline”. Існують параметри, що спільні для всіх віджетів, такі як розміщення, та унікальні для конкретного віджета, такі як початкове значення, клікабельність та інші. Всі об’єкти параметрів віджетів є членами класу “Form”, який і є об’єктом представлення параметрів головного вікна програмного застосунку. Клас “Form” є членом класу “Parcer”, який у свою чергу є алгоритмом парсингу файлу розмітки, та надсилає команди для створення віджетів чи зміни їх параметрів класу “Form”.

## 2.3 Конструювання програмного забезпечення

Компонент Markup приймає на вхід шлях до файлу розмітки. Після зчитування файлу, компонент приступає до розшифрування команд, що описані в ньому. Розшифрування команд одна за одною допомагає покроково формувати вихідний набір параметрів. У випадку певних некоректних даних, компонент може або зупинити парсинг і повернути повідомлення про некоректні дані, або тільки додати повідомлення до рядка статусу і повернути отримані дані. Наступним кроком набір параметрів отримує клас Application, а саме конструктор його базового класу BaseApplication. Набір параметрів далі використовується класом Window для створення всіх необхідних віджетів. Далі бібліотека використовує функціонал WinApi для створення графічного інтерфейсу користувача.

На рисунку 2.3 представлена діаграма класів компоненту UI, що об’єдню в собі функціонал для відображення віджетів на основі параметрів з файлу розмітки, а також функціонал для реалізації подійно-орієнтованого механізму та інтерфейс для створення програмних застосунків.

За реалізацію подійно-орієнтованого механізму відповідає клас Event, що створений для ідентифікації та диспетчиразації подій. Ідентифікатором

події є об’єкт цього класу. Такий механізм дозволяє робити події членами класів Button, RadioButton і тд, і надалі зручно поєднювати події з хендлерами для них. У якості хендлера подій виступає клас std::function, а конкретно std::function<void()>. Це функція, що не приймає ніяких параметрів і не повертає значень. Перевагою такого рішення є створена можливість для користувача використовувати функції та методи з будь-якою кількістю параметрів для їх виклику при виникненні подій завдяки обгортанню їх в лямбда вирази. Первинна ідентифікація події, що сталася, відбувається за допомогою функціоналу WinApi, а саме трансляції та диспетчирезації низькорівневих повідомлень між віджетами та операційною системою, що відбувається в межах методу runMessageLoop класу BaseApplication, що реалізує цикл опрацювання подій в системі. Далі інформація про подію передається до класу Window, а саме опрацьовується у методі WindowProc(), де відбувається визначення віджета, якого стосується подія, а далі конкретизація типу події. Після цього буде викликано метод _emit у відповідного об’єкту класу Event, що активує всі хендлери, налаштовані користувачем. Таким чином, створений клас Event разом з класами Window та BaseApplication повністю реалізує подійну орієнтованість бібліотеки та дозволяє користувачу як і налаштовувати реакції на взаємодію користувача з віджетами, так і створювати власні події та опрацьовувати їх.

Компонент UI також відповідає за надання користувачу інтерфейсу для створення застосунків з графічним інтерфейсом користувача. Головною складовою цього інтерфейсу є клас BaseApplication, що зберігає в собі клас головного вікна. Конструктор цього класу використовує компонент парсингу розмітки і отримує параметри, описані в файлі CppUI.txt. На основі цих параметрів за допомогою необхідних приватних методій BaseApplication створює об’єкт головного вікна та заповнює його віджетами. Користувачу для створення графічного інтерфейсу потрібно або створити об’єкт класу BaseApplication, або унаслідуватись від нього. Головним членом цього класу є об’єкт класу Window, головне вікно застосунку. Створення віджетів на основі

параметрів відбувається за допомогою шаблонного методу addWidget, що належить до класу Window. Цей метод використовується спеціальними методами класу BaseApplication для кожного віджета.

Клас Window створений для створення вікна та віджетів, що розташовані на ньому. Конструктор цього класу приймає заголовок вікна у вигляді рядкової змінної, а також розміри вікна та спеціальну змінну типу HINSTANCE, що є хендлером застосунку в операційній системі Windows. В даному класі є метод render, що відповідає за створення вікна, його відображення та виклик методів render у всіх необхідних віджетів. Віджети зберігаються у форматі мапінгу спеціального ідентифікатора, що створений системою для опрацювання системних взаємодій з віджетами, на об'єкти класів віджетів. Для зручності користувача створений додатковий мапінг користувацьких ідентифікаторів (рядкова змінна) на внутрішній ідентифікатор. Таким чином, користувач може отримати посилання на об'єкт віджета за допомогою методу findWidgetByName. Важливою частиною класу Window є метод WindowProc, що створений для відловлювання повідомлень про взаємодію користувача з певними віджетами і подального виклику відповідних подій.

Клас IWidget є базовим класом для всіх віджетів. Цей клас реалізує метод render, що відповідає за створення віджета на вікні. Параметрами класу IWidget є ім'я (користувацький ідентифікатор віджета, рядкова змінна), розмір (об'єкт класу Size, що містить інформацію про висоту та ширину віджета), позицію (об'єкт класу Position, що зберігає вертикальну та горизонтальну координату лівого верхнього кута віджета відносно лівого верхнього кута вікна), та текст (об'єкт класу Text, що зберігає в собі рядкове значення тексту, а також об'єкт Font, що відображає параметри шрифта такі, як розмір, назву, жирність, нахил). Також цей клас містить віртуальний метод getStyles, що використовується методом рендер і представляє собою низькорівневі стилі WinApi для створення віджета. Метод render використовує збережені параметри класу IWidget та результат функції getStyles для створення віджетів за допомогою функції CreateWindowEx[10], що є частиною WinApi.

Кожен з класів Button, Label, FieldEdit, CheckBox, RadioButton, SpinBox наслідується від класу IWidget. Ці класи реалізують в собі метод getStyles, а також зберігають об’єкти подій, що можуть виникати при взаємодії користувача з ними. Клас Button містить події _click та _doubleClick, що відповідають за натискання та подвійне натискання на кнопку. Клас FieldEdit містить подію _textChanged, що відповідає за зміну тексту у полі для вводу. Класи RadioButton та CheckBox містять подію _clicked, що теж відповідають за натискання клієнта на цей віджет.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.1.

Таблиця 2.1 – Опис утиліт

| № п/п | Назва утиліти | Опис застосування |
|---|---|---|
| 1 | Visual Studio Code | Головне середовище розробки програмного забезпечення серверної частини курсової роботи. |
| 2 | CMake | Набір засобів та інструментів для управління та автоматизації процесу збірки, інсталяції, запуску тестування програмного забезпечення. |
| 3 | WinApi | Набір програмних інтерфейсів, що дозволяє створювати різні застосунки з використанням функціоналу Windows |

## 2.4 Аналіз безпеки даних

Розроблене програмне забезпечення є бібліотекою, тому не передбачає збереження особистих даних користувача у тому чи іншому вигляді. Необхідними даними для роботи бібліотеки є файл розмітки. Функціоналом

бібліотеки не передбачено наявності певних чутливих даних, що використовуються в розмітці. Користувач може використовувати CppUI для розробки інших застосунків, що будуть містити в собі необхідність роботи з чутливими даними. В такому випадку відповідальність на управління безпекою цих даних лежить на плечах користувача. Всі функції бібліотеки була протестовані за умови їх цільового використання. При їх використанні у непередбачених для цього випадках, всі ризики є відповідальністю користувача.

Висновки до розділу

У підрозділі 2.1 було проведено моделювання та аналіз програмного забезпечення. Модель бізнес процесу використання розробленого програмного забезпечення представлено за допомогою BPMN діаграми. Наведено опис процесу створення застосунку з графічним інтерфейсом користувача з використанням бібліотеки та мови позмітки.

У підрозділі 2.2 було побудовано діаграму архітектури програмного бібліотеки. Для реалізації бібліотеки було обрано монолітну архітектуру, що логічно розділяється на два компоненти – UI та Markup. Компонент Markup відповідає за зчитування параметрів з файлу розмітки. Компонент UI відповідає за створення графічного інтерфейсу на основі отриманих параметрів розмітки, контроль за ресурсами бібліотеки, обробку взаємодій користувача з віджетами. Для деталізації архітектури програмного забезпечення наведено діаграми класів, описано призначення всіх сутностей.

У підрозділі 2.3 було описано конструювання програмного забезпечення, алгоритм отримання параметрів з файлу розмітки, процес створення графічного інтерфейсу користувача на основі зчитаних параметрів, а також детально розглянуто всі класи, проаналізовано принципи їх роботи їх роботи, у табличному форматі наведено опис утиліт, що використовувались.

У підрозділі 2.4 було проведено аналіз безпеки ПЗ і визначено, що програмне забезпечення не використовує чутливі дані користувача. Якщо

користувач хоче реалізувати застосунок, що використовує дану бібліотеку, і передбачає зберігання особистих даних, про їх безпеку користувач повинен подбати самостійно.

# 3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Аналіз якості ПЗ

Для аналізу коду використовується аналізатор коду Embold[17]. Перевагою використання такого сервісу є можливість апаратної оцінки коду за великою кількістю метрик базуючись на статичному аналізі коду. Даний сервіс повертає загальну оцінку коду за п’ятибальною шкалою. Найбільш важливими метриками для розробленої бібліотеки є швидкодія (при відмальовуванні графічного інтерфейсу, парсингу розмітки, опрацюванню подій швидкодія відіграє важливу роль), надійність (бібліотека буде використовуватись для розробки інших програмних продуктів, тому надійність є ключовим фактором), зручність використання, ефективність та стійкість. На рисунку 2.3 представлено результат сканування коду бібліотеки сервісом Embold. Отриманий результат – 4,83, що є досить хорошим показником. За обраними метриками оцінки, надані сервісом, теж високі.

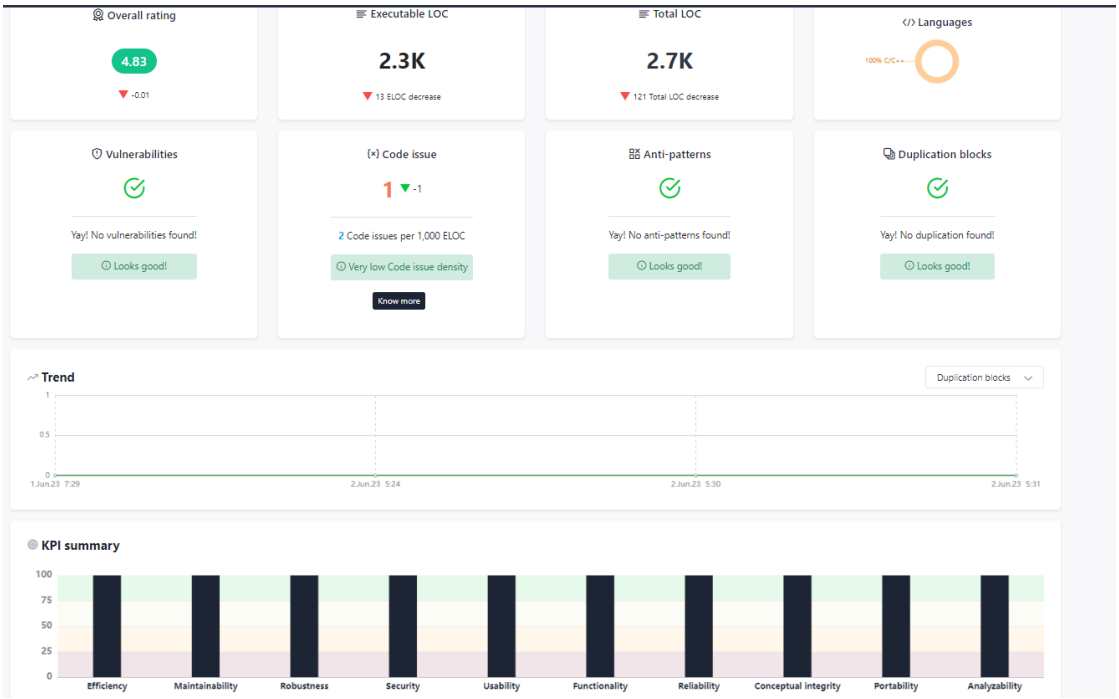



Рисунок 3.1 – Результат сканування сервісом Embold

Розроблена мова розмітки є інтуїтивно зрозумілою та простою у використанні. На рисунку 3.2 представлено приклад файлу розмітки. Читати

інформацію з цього файлу можна навіть без детальних знань про синтаксис мови розмітки, легко здогадатись, що описує той чи інший рядок. Розроблений синтаксис дозволяє користувачу описувати параметри вікна та віджетів у чіткому, лаконічному, зрозумілому форматі. При створенні файлу розмітки користувач повинен дотримуватись чітких правил оформлення опису параметрів у файлі розмітки. У випадку порушення певних правил опису даних у файлі розмітки, компонент, що відповідає за парсинг параметрів, поверне користувачу зрозумілу відповідь про місце помилки. Файлом розмітки є текстовий файл з розширенням .txt, що дозволяє редагувати вміст даного файлу в будь-якому, зручному для користувача, текстовому редакторі.

```
1   / name : Test Project
2   / sizeVert : 400
3   / sizeGorz : 360
4   | Button [
5   /         name : Button_Calculate
6   /         sizeVert : 50
7   /         sizeGorz : 325
8   /         positionVert : 60
9   /         positionGorz : 10
10  |     Text [
11  /          text: Calculate]
12  /]
13  | Label [
14  /        name: Label_X_Square
15  /         sizeVert : 25
16  /         sizeGorz : 50
17  /         positionVert : 20
18  /         positionGorz : 65
19  |     Text [
20  /          text: x^2 + ]
21  /]
```

Рисунок 3.2 – Приклад файлу розмітки

Розроблена бібліотека, для проектування графічного інтерфейсу користувача є інтуїтивно зрозумілою, зручною та простою у використанні. Дана бібліотека надає користувачу інтерфейс (базовий клас, BaseApplication) для головного класу застосунку, що включає в себе реалізовані параметри створення вікна та віджетів на основі параметрів з файлу розмітки, а також реалізований метод runMessageLoop, що реалізує запуск опрацювання повідомлень між складовими графічного інтерфейсу. Для створення графічного інтерфейсу користувачу достатньо описати параметри необхідних

віджетів у файлі розмітки та створити свій клас, що наслідується від базового класу. Створений клас не вимагає від користувача реалізації будь-яких методів поки користувач не хоче додати функціонал опрацювання певних події. Для реалізації логіки, що спрацьовує при взаємодії користувача з віджетами, достатньо реалізувати спеціальний метод з необхідним функціоналом та підключити його до події за допомогою методу Event::connect.

## 3.2 Опис процесів тестування

Після розробки подійно-орієнтованої бібліотеки, було виконано мануальне тестування з метою перевірки основних можливостей бібліотеки та юніт тестування з метою перевірки правильності роботи програмного застосунку.

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.1 – 3.28.

Таблиця 3.1 – Тест 1.1

| | |
|---|---|
| Тест | Створення параметрів вікна без віджетів |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.1 |
| Початковий стан системи | Було коректно підключено бібліотеку |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Створюється об’єкт параметри форми з стандартними значеннями параметрів(size 1200x720, name “Form”) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними значеннями параметрів(розмір 1200x720) |

Таблиця 3.2 – Тест 1.2

| | |
|---|---|
| Тест | Створення параметрів віджету “Button” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.2 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення віджета “Button” |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Button” з стандартними параметрами(size 120x72, position(0,0)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Button” з стандартними параметрами(size 120x72, position(0,0)) |

Таблиця 3.3 – Тест 1.3

| | |
|---|---|
| Тест | Створення параметрів віджету “CheckBox” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.3 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення віджета “CheckBox” |

Продовження таблиці 3.3

| | |
|---|---|
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється о’єкт параметрів “CheckBox” з стандартними параметрами(position(0,0)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “CheckBox” з стандартними параметрами(position(0,0)) |

Таблиця 3.4 – Тест 1.4

| | |
|---|---|
| Тест | Створення параметрів віджету “EditLine” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.4 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення віджета “EditLine” |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “EditLine” з стандартними параметрами size 120x72, (position(0,0)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “EditLine” з стандартними параметрами(size 120x72, position(0,0)) |

Таблиця 3.5 – Тест 1.5

| | |
|---|---|
| Тест | Створення параметрів віджету “Label” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.5 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення віджета “Label” |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Label” з стандартними параметрами(position(0,0)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Label” з стандартними параметрами(position(0,0)) |

Таблиця 3.6 – Тест 1.6

| | |
|---|---|
| Тест | Створення параметрів віджету “RadioButton” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.6 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення віджета “RadioButton” |

Продовження таблиці 3.6

| | |
|---|---|
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “RadioButton” з стандартними параметрами(position(0,0)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “RadioButton” з стандартними параметрами(position(0,0)) |

Таблиця 3.7 – Тест 1.7

| | |
|---|---|
| Тест | Створення параметрів віджету “NumberBox” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.7 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення віджета “NumberBox” |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “NumberBox” з стандартними параметрами(position(0,0)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “NumberBox” з стандартними параметрами(position(0,0)) |

Таблиця 3.8 – Тест 1.8

| | |
|---|---|
| Тест | Створення параметрів віджету “Text” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.8 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення “Text” у іншому віжеті |
| Очікуваний результат | Створюється об’єкт параметів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Text” у вказаному віджеті з стандартними параметрами (position(0,0), text “Text”, title “Arial”, italic false, bold false, underline false, height 12) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Text” у вказаному віджеті з стандартними параметрами (position(0,0), text “Text”, title “Arial”, italic false, bold false, underline false, height 12) |

Таблиця 3.9 – Тест 1.9

| | |
|---|---|
| Тест | Задання параметрів для віджета “Form” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.9 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |

Продовження таблиці 3.9

| | |
|---|---|
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав параметри віджета “Form” (size 100x100, name “Test form”) |
| Очікуваний результат | Створюється об’єкт параметрів форми з вказаними параметрами з (size 100x100, name “Test form”) |
| Фактичний результат | Створюється об’єкт параметрів форми з вказаними параметрами з (size 100x100, name “Test form”) |

Таблиця 3.10 – Тест 1.10

| | |
|---|---|
| Тест | Задання параметрів для віджета “Button” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.10 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав параметри віджета “Button” (size 200x70, position(100,100)) |
| Очікуваний результат | Створюється об’єкт параметів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Button” з заданими параметрами (size 200x70, position(100,100)) |

Продовження таблиці 3.10

| | |
|---|---|
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Button” з заданими параметрами (size 200x70, position(100,100)) |

Таблиця 3.11 – Тест 1.11

| | |
|---|---|
| Тест | Задання параметрів для віджета “CheckBox” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.11 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав параметри віджета “CheckBox” (position(50,50)) |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “CheckBox” з стандартними параметрами(position(50,50)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “CheckBox” з стандартними параметрами(position(50,50)) |

Таблиця 3.12 – Тест 1.12

| | |
|---|---|
| Тест | Задання параметрів для віджета “Editline” |
| Модуль | Зчитування параметрів з файлу розмітки |

Продовження таблиці 3.12

| | |
|---|---|
| Номер тесту | 1.12 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав параметри віджета “Editline” (position(50,50)) |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Editline” з вказаними параметрами(position(50,50)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Editline” з вказаними параметрами(position(50,50)) |

Таблиця 3.13 – Тест 1.13

| | |
|---|---|
| Тест | Задання параметрів для віджета “Label” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.13 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав параметри віджета “Label” (position(50,50)) |

Продовження таблиці 3.13

| | |
|---|---|
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Label” з вказаними параметрами(position(50,50)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Label” з вказаними параметрами(position(50,50)) |

Таблиця 3.14 – Тест 1.14

| | |
|---|---|
| Тест | Задання параметрів для віджета “NumberBox” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.14 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав параметри віджета “NumberBox” (position(50,50)) |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “NumberBox” з вказаними параметрами(position(50,50)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “NumberBox” з вказаними параметрами(position(50,50)) |

Таблиця 3.15 – Тест 1.15

| | |
|---|---|
| Тест | Задання параметрів для віджета “RadioButton” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.15 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав параметри віджета “RadioButton” (position(50,50)) |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “RadioButton” з вказаними параметрами (position(50,50)) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “RadioButton” з вказаними параметрами (position(50,50)) |

Таблиця 3.16 – Тест 1.16

| | |
|---|---|
| Тест | Задання параметрів для віджета “Text” |
| Модуль | Зчитування параметрів з файлу розмітки |
| Номер тесту | 1.16 |
| Початковий стан системи | Було коректно підключено бібліотеку та заповнено файл розмітки |
| Вхідні данні | Коректно заповнений файл розмітки “CppUI.txt” |

Продовження таблиці 3.16

| | |
|---|---|
| Опис проведення тесту | Користувач підключив бібліотеку, у файлі розмітки правильно, без порушень правил синтаксису та граматики, описав створення “Text” з параметрами (text “Text”, title “Arial”, italic false, bold false, underline false, height 12) у іншому віжеті |
| Очікуваний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Text” у вказаному віджеті з вказаними параметрами параметрами (text “Text”, title “Arial”, italic false, bold false, underline false, height 12) |
| Фактичний результат | Створюється об’єкт параметрів форми з стандартними параметрами(size 1200x720, name “Forms”) та створюється об’єкт параметрів “Text” у вказаному віджеті з вказаними параметрами параметрами (text “Text”, title “Arial”, italic false, bold false, underline false, height 12) |

Таблиця 3.17 – Тест 2.1

| | |
|---|---|
| Тест | Створення графічного відображення вікна без віджетів |
| Модуль | Відображення віджетів на основі параметрів з мови розмітки |
| Номер тесту | 2.1 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено базовий варіант файлу розмітки, що містить тільки параметри вікна |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |

Продовження таблиці 3.17

| | |
|---|---|
| Очікуваний результат | Відображення вікна з параметрами, що описані в файлі розмітки, що не містить віджетів |
| Фактичний результат | Відображене вікно з параметрами, що описані в файлі розмітки |

Таблиця 3.18 – Тест 2.2

| | |
|---|---|
| Тест | Створення графічного відображення вікна з кнопкою |
| Модуль | Відображення віджетів на основі параметрів з файлу розмітки |
| Номер тесту | 2.2 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та кнопки з параметрами, що описані в файлі розмітки |
| Фактичний результат | Відображено вікно та кнопку з параметрами, що описані в файлі розмітки |

Таблиця 3.19 – Тест 2.3

| | |
|---|---|
| Тест | Створення графічного відображення вікна з текстовим написом |
| Модуль | Відображення віджетів на основі параметрів з файлу розмітки |
| Номер тесту | 2.3 |

Продовження таблиці 3.19

| | |
|---|---|
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри текстового напису |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та текстового напису з параметрами, що описані в файлі розмітки |
| Фактичний результат | Відображено вікно та текстовий напис з параметрами, що описані в файлі розмітки |

Таблиця 3.20 – Тест 2.4

| | |
|---|---|
| Тест | Створення графічного відображення вікна з полем для вводу |
| Модуль | Відображення віджетів на основі параметрів з файлу розмітки |
| Номер тесту | 2.4 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри поля для вводу |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та текстового напису з параметрами, що описані в файлі розмітки |

Продовження таблиці 3.20

| | |
|---|---|
| Фактичний результат | Відображено вікно та текстовий напис з параметрами, що описані в файлі розмітки |

Таблиця 3.21 – Тест 2.5

| | |
|---|---|
| Тест | Створення графічного відображення вікна з кнопкою-прапорцем |
| Модуль | Відображення віджетів на основі параметрів з файлу розмітки |
| Номер тесту | 2.5 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки-прапорця |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та кнопки-прапорця з параметрами, що описані в файлі розмітки |
| Фактичний результат | Відображено вікно та кнопку-прапорець з параметрами, що описані в файлі розмітки |

Таблиця 3.22 – Тест 2.6

| | |
|---|---|
| Тест | Створення графічного відображення вікна з кнопкою вибору |
| Модуль | Відображення віджетів на основі параметрів з файлу розмітки |
| Номер тесту | 2.6 |

Продовження таблиці 3.22

| | |
|---|---|
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки вибору |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та кнопки вибору з параметрами, що описані в файлі розмітки |
| Файтичний результат | Відображено вікно та кнопку вибору з параметрами, що описані в файлі розмітки |

Таблиця 3.23 – Тест 2.7

| | |
|---|---|
| Тест | Створення графічного відображення вікна з спінбоксом |
| Модуль | Відображення віджетів на основі параметрів з файлу розмітки |
| Номер тесту | 2.7 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та спінбокса з параметрами, що описані в файлі розмітки |
| Фактичний результат | Відображено вікно та спінбокс з параметрами, що описані в файлі розмітки |

Таблиця 3.24 – Тест 3.1

| | |
|---|---|
| Тест | Реалізація методів взаємодії користувача з кнопкою |
| Модуль | Реалізація методів взаємодії користувача з віджетами |
| Номер тесту | 3.1 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки. Реалізовано метод, який відображає повідомлення на екрані. Підключення методу до події _click на кнопці |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та кнопки з параметрами, що описані в файлі розмітки. При натисканні на кнопку з’являється повідомлення на екрані. |
| Фактичний результат | Відображено вікно та кнопку з параметрами, що описані в файлі розмітки. При натисканні на кнопку з’явилось повідомлення на екрані |

Таблиця 3.25 – Тест 3.2

| | |
|---|---|
| Тест | Реалізація методів взаємодії користувача з кнопкою (подвійне натискання) |
| Модуль | Реалізація методів взаємодії користувача з віджетами |
| Номер тесту | 3.2 |

Продовження таблиці 3.25

| | |
|---|---|
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки. Реалізовано метод, який відображає повідомлення на екрані. Підключення методу до події _doubleClick на кнопці |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та кнопки з параметрами, що описані в файлі розмітки. При подвійному натисканні на кнопку з’являється повідомлення на екрані. |
| Фактичний результат | Відображено вікно та кнопку з параметрами, що описані в файлі розмітки. При подвійному натисканні на кнопку з’явилось повідомлення на екрані |

Таблиця 3.26 – Тест 3.3

| | |
|---|---|
| Тест | Реалізація методів взаємодії користувача з полем для вводу |
| Модуль | Реалізація методів взаємодії користувача з віджетами |
| Номер тесту | 3.3 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри поля для вводу. Реалізовано метод, який відображає повідомлення на екрані. Підключення методу до події _textChanged на полі для вводу |
| Вхідні данні | - |

Продовження таблиці 3.26

| | |
|---|---|
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та поле для вводу з параметрами, що описані в файлі розмітки. При зміні введеного тексту у поле для вводу з’являється повідомлення на екрані. |
| Фактичний результат | Відображено вікно та текстовий напис з параметрами, що описані в файлі розмітки. При зміні введеного тексту у полі для вводу з’явилось повідомлення на екрані |

Таблиця 3.27 – Тест 3.4

| | |
|---|---|
| Тест | Реалізація методів взаємодії користувача з кнопкою-прапорцем |
| Модуль | Реалізація методів взаємодії користувача з віджетами |
| Номер тесту | 3.4 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки-прапорця. Реалізовано метод, який відображає повідомлення на екрані. Підключення методу до події _click на кнопці-прапорці |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та кнопки-прапорця з параметрами, що описані в файлі розмітки. При натисканні на кнопку з’являється повідомлення на екрані. |

Продовження таблиці 3.27

| | |
|---|---|
| Фактичний результат | Відображено вікно та кнопку-прапорець з параметрами, що описані в файлі розмітки. При натисканні на кнопку з’явилось повідомлення на екрані |

Таблиця 3.28 – Тест 3.5

| | |
|---|---|
| Тест | Реалізація методів взаємодії користувача з кнопкою вибору |
| Модуль | Реалізація методів взаємодії користувача з віджетами |
| Номер тесту | 3.5 |
| Початковий стан системи | Було коректно підключено бібліотеку, створено файл розмітки, що містить тільки параметри вікна та параметри кнопки вибору. Реалізовано метод, який відображає повідомлення на екрані. Підключення методу до події _click на кнопці |
| Вхідні данні | - |
| Опис проведення тесту | Користувач запускає програмний застосунок |
| Очікуваний результат | Відображення вікна та кнопки вибору з параметрами, що описані в файлі розмітки. При натисканні на кнопку вибору з’являється повідомлення на екрані. |
| Фактичний результат | Відображено вікно та кнопку вибору з параметрами, що описані в файлі розмітки. При натисканні на кнопку вибору з’явилось повідомлення на екрані |

## 3.3 Опис контрольного прикладу

В якості опису контрольного прикладу опишемо створення графічного інтерфейсу користувача для вирішення задачі обчислення розв’язків

квадратного рівніння. Прототип вікна даного застосунку зображено на рисунку 3.2. Для створення такого застосунку першим кроком користувач має підключити бібліотеку для свого проекту (алгоритм підключення описано в розділі 4).

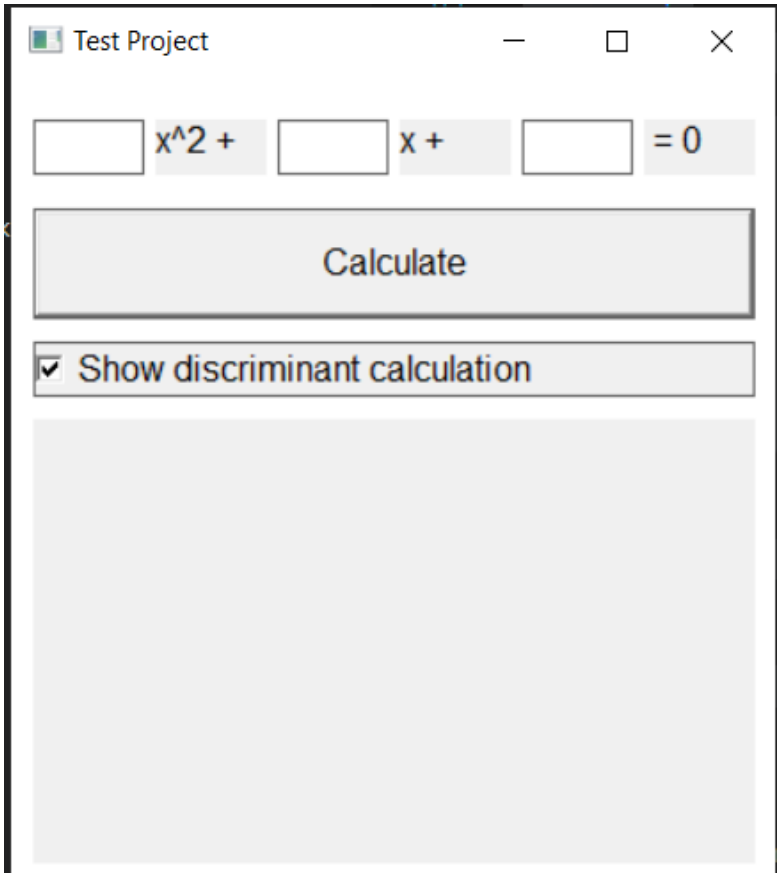
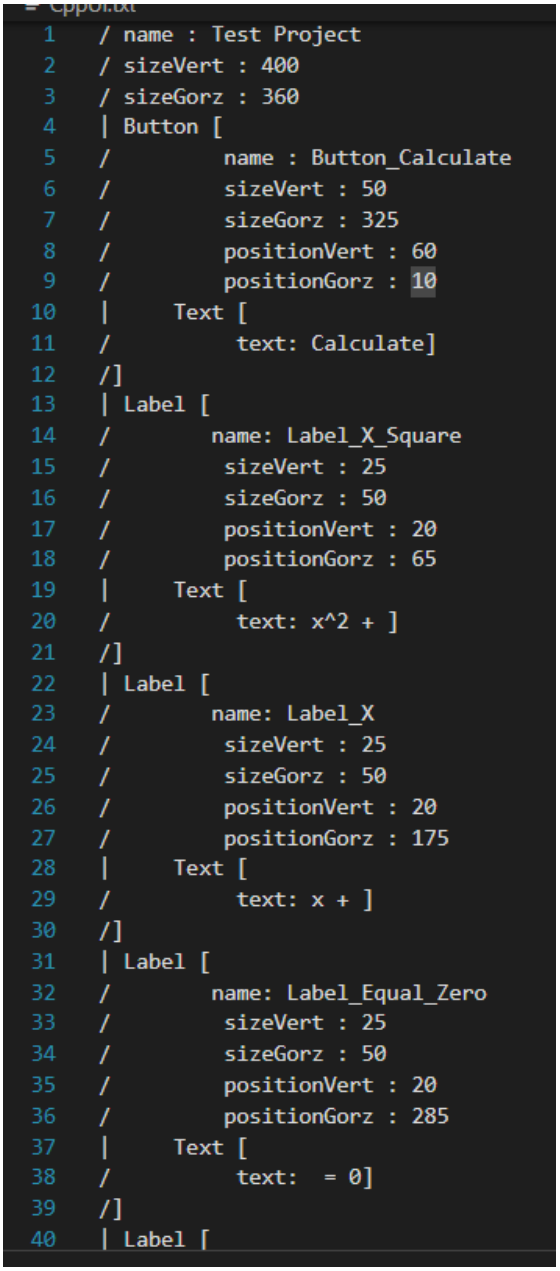



Рисунок 3.3 – Прототип екранної форми контрольного прикладу

Далі користувачу потрібно описати параметри віджетів в файлі розмітки. Для створення екранної форми, зображеної на рисунку 3.3, користувачу потрібно описати параметри трьох полей для вводу (для введення коефіцієнтів квадратного рівняння), параметри кнопки для обчислення коренів квадратного рівняння, параметри кнопки-прапорця, що активує вивід значення дискримінанта на екран разом з коренями, а також параметри 4 написві для відображення статичного тексту та результату. На рисунку 3.4 показано приклад файлу розмікти з усіма необхідними параметрами для створення віконної форми, що показана на рисунку 3.3.



```
/ name : Test Project
/ sizeVert : 400
/ sizeGorz : 360
| Button [
/         name : Button_Calculate
/         sizeVert : 50
/         sizeGorz : 325
/         positionVert : 60
/         positionGorz : 10
|     Text [
/          text: Calculate]
/]
| Label [
/        name: Label_X_Square
/         sizeVert : 25
/         sizeGorz : 50
/         positionVert : 20
/         positionGorz : 65
|     Text [
/          text: x^2 + ]
/]
| Label [
/        name: Label_X
/         sizeVert : 25
/         sizeGorz : 50
/         positionVert : 20
/         positionGorz : 175
|     Text [
/          text: x + ]
/]
| Label [
/        name: Label_Equal_Zero
/         sizeVert : 25
/         sizeGorz : 50
/         positionVert : 20
/         positionGorz : 285
|     Text [
/          text:  = 0]
/]
| Label [
```

Рисунок 3.4 – Приклад файлу розмітки

Наступним кроком користувач повинен створити основний клас застосунку, що наслідується від класу BaseApplication, що є частиною інтерфейсу бібліотеки. Для створення тільки відображення (без логіки взаємодії з користувачем) достатньо просто створити об’єкт створеного класу. На рисунку 3.5 представлено код, що задачу створення графічного інтерфейсу без логіки обробки подій.

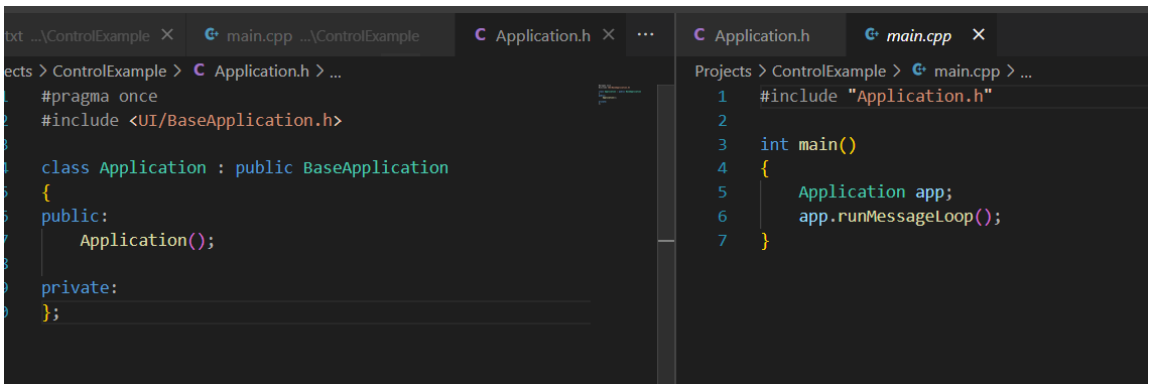



Рисунок 3.5 – Код для створення графічного відображення форми

Далі користувачу потрібно реалізувати функцію (метод), що буде викликатись при натисканні на кнопку. Для зручності розв'зку квадратного рівняння користувач може ввести абстракцію Quadratic Equation (код представлено на рисунку 3.6). Даний клас у конструкторі прийматиме коефіцієнти квадратного рівняння, а метод getRootsString повертатиме рядкове представлення коренів цього рівняння.

```
/**
 * @brief _ax^2 + _bx + _c = 0
 */
class QuadraticEquation
{
public:
    QuadraticEquation(double a, double b, double c) : _a{a}, _b{b}, _c{c} {}
    std::string getRootsString(bool printDiscriminant) const

private:
    double _a;
    double _b;
    double _c;
};
```

Рисунок 3.6 – Клас для представлення розв'язків рівняння у вигляді текстового рядка

Отже, метод, який спрацьовуватиме при натисканні кнопки, повинен зчитати з полів для вводу коефіцієни квадратного рівняння, перевірити, що всі вони числа, створити об'єкт класу QuadraticEquation, отримати результат методу getRootsString та відобразити його у спеціальному полі для виводу результатів на екрані. У випадку, коли у певне поле введення буде введено нечислове значення, передбачимо очищення всіх полів вводу та вивід

повідомлення про некоректнійцсть значення. На рисунку 3.7 представлено код методу calculate, що має викликатись при натисканні на кнопку.

```
void calculate()
{
    try
    {
        auto *labelA = _mainWindow.findWidgetByName<FieldEdit>("Coefficient_X_Squared");
        auto *labelB = _mainWindow.findWidgetByName<FieldEdit>("Coefficient_X");
        auto *labelC = _mainWindow.findWidgetByName<FieldEdit>("Coefficient_Free_Member");
        QuadraticEquation eq(
            std::stod(labelA->getText().c_str()),
            std::stod(labelB->getText().c_str()),
            std::stod(labelC->getText().c_str()));
        auto *checkBoxShowDisc = _mainWindow.findWidgetByName<CheckBox>("CheckBox_Show_Discriminant");
        auto *label = _mainWindow.findWidgetByName<Label>("Label_Answer");
        label->setText(eq.getRootsString(checkBoxShowDisc->isChecked()));
    }
    catch (...)
    {
        auto *label = _mainWindow.findWidgetByName<Label>("Label_Answer");
        label->setText("Invalid arguments");
    }
}
```

Рисунок 3.7 – Метод calculate

На рисунку 3.8 показано підключення методу до події натискання на кнопку.

```
auto *button = _mainWindow.findWidgetByName<Button>("Button_Calculate");
button->_click.connect([this]()
                       { this->calculate(); });
```

Рисунок 3.8 – Підключення методу до кнопки

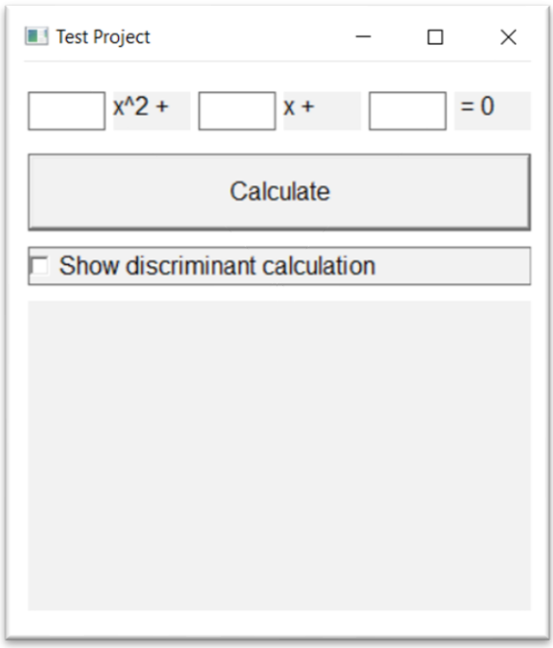



Рисунок 3.9 – Початковий стан застосунку

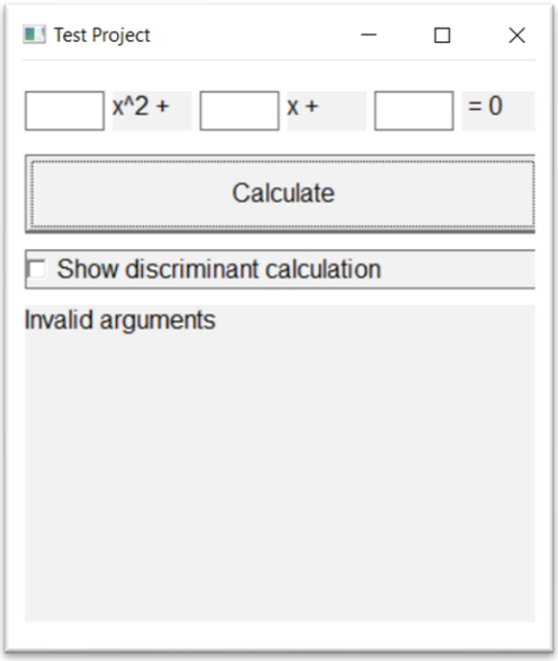



Рисунок 3.10 – Результат застосунку при невведених даних

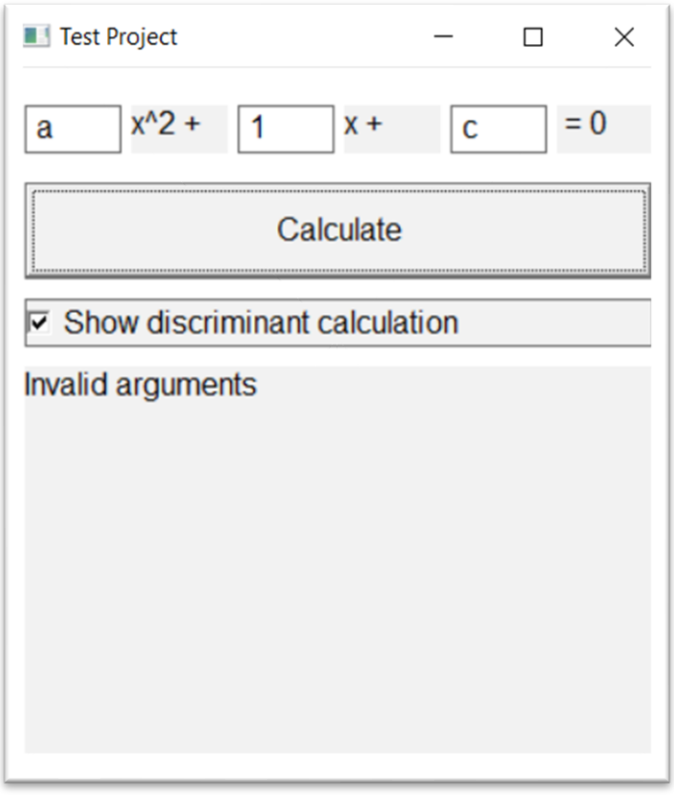



Рисунок 3.11 – Результат при введенні нечислових коефіцієнтів

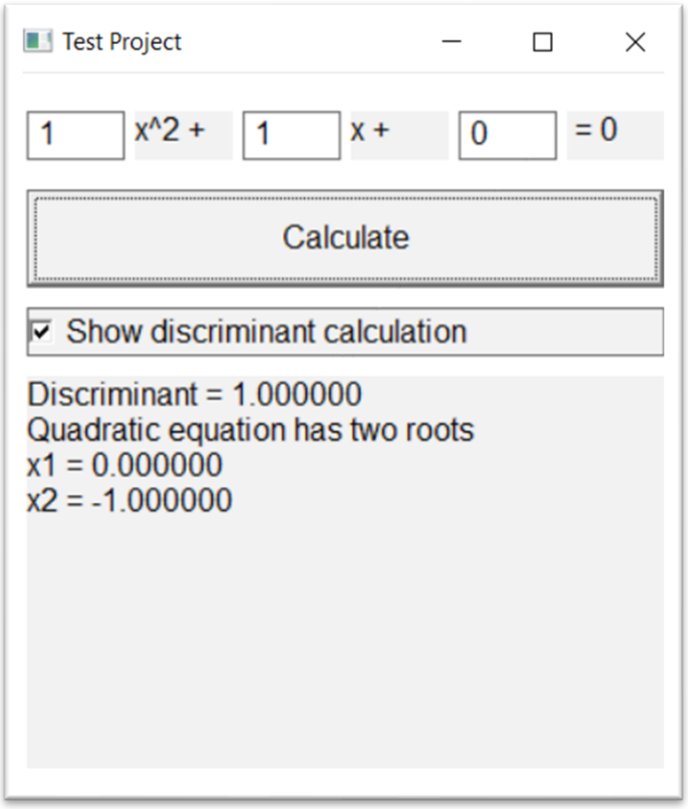



Рисунок 3.12 – Результат системи – два корені

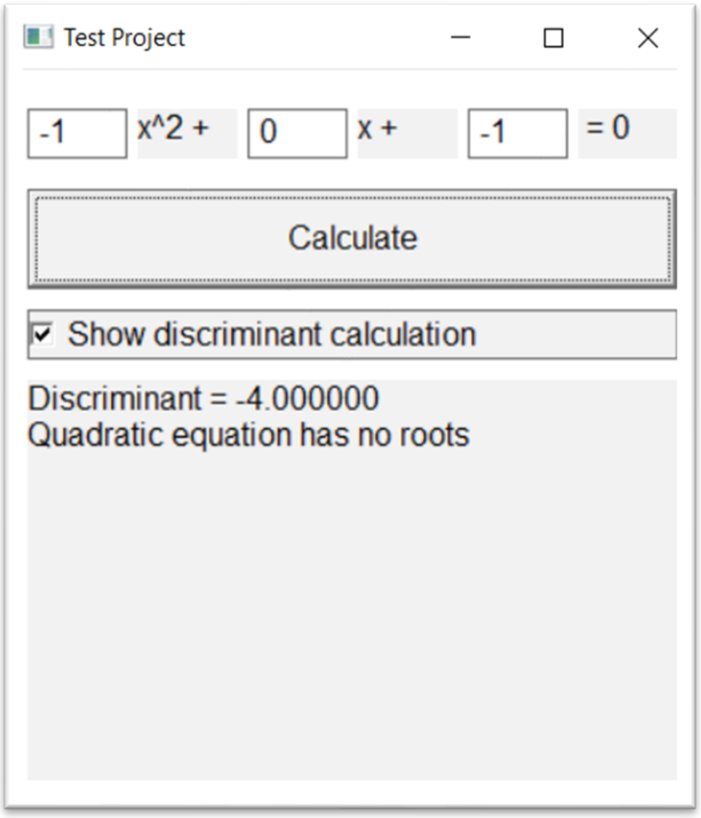



Рисунок 3.13 – Результат системи – коренів немає

На рисунках 3.10 – 3.13 представлені приклади роботи контрольного прикладу для різних варіантів введених даних від кристувача.

## 3.4 Висновки до розділу

У третьому розділі було проведено аналіз якості та тестування програмного забезпечення.

У підрозділі 3.1 за допомогою сервісу Embold було проведено статичний аналіз коду за обраними метриками, а саме швидкодія, надійність, ефективність, стійкість та зручність використання. За всіма метриками отримано високу оцінку. Загальна оцінка коду проекту надана сервісом Embold дорівнює 4,83 за п’ятибальною шкалою, що є хорошим результатом. Також у даному розділі показано, що бібліотека відповідає всім нефункціональним вимогам.

У підрозділі 3.2 було описано процеси тестування розробленої бібліотеки. Для тестування використовувались мануальні тести, оскільки вони найбільше підходять для тестування такого програмного забезпечення. Опис тестів наведено у таблиці. Тестування пройшло успішно, всі тести пройдені.

У підрозділі 3.3 було наведено контрольний приклад використання роозробленого програмного забезпечення бібліотеки. За допомогою бібліотеки створено застосунок з графічним інтерфейсом користувача для обчислення коренів квадратного рівняння.

# 4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Розгортання програмного забезпечення

Розгортання бібліотеки полягає у її підключенні до будь-якого проекту. Рекомендованим середовищем для розробки застосунків з використанням бібліотеки є редактор коду Visual Studio Code. Для налаштування збірки, тестування, запуску застосунку з графічним інтерфейсом рекомендується використовувати CMake. Саме ці технологій використовувались для розробки бібліотеки і для них буде показана інструкція з розгортання. Користувач має право використовувати для розробки своїх застосунків (з використанням розробленої бібліотеки) і інші технології, якщо має достатню експертизу там.

Вихідний код бібліотеки розміщений на платформі GitHub. Стартовою точкою інтеграції бібліотеки в конкретний проект буде створення нового CMake проекту (користувач може використовувати будь-який проект, куди йому потрібно підключити бібліотеку). На рисунку 4.1 можна побачити приклад CMakeLists.txt файлу перед початком інтеграції бібліотеки. На рисунку 4.2 можна побачити приклад початкового main.cpp файлу. Приклад роботи застосунку до інтеграції бібліотеки можна побачити на рисунку 4.3.

Першим етапом інтеграції бібліотеки до описаного вище проетку буде клонування репозиторію з бібліотекою до папки проекту. Це можна зробити за допомогою команди “git clone https://github.com/vovahrytsiuk/Diploma.git ./CppUI”. В результаті в папці проекту ми отримаємо повний вихідний код бібліотеки. На рисунку 4.4 можна побачити структуру проекту після клонування бібліотеки. Наступним кроком додамо статичну бібліотеку до проекту за допомогою команди add_subdirectory(CppUI). Дана команда допоможе налаштувати шлях до файлів бібліотеки в нашому застосунку. Надалі потрібно залінкувати бібліотеку за допомогою команди target_link_libraries. Обов’язковим кроком такоє є встановлення стандарту

C++17. На рисунку 4.5 пожна побачити приклад CMakeLists.txt з підключеною бібліотекою.

main.cpp CMakeLists.txt

CMakeLists.txt

```
1   cmake_minimum_required(VERSION 3.0.0)
2   project(testProject VERSION 0.1.0)
3
4   include(CTest)
5   enable_testing()
6
7   add_executable(testProject main.cpp)
8
9   set(CPACK_PROJECT_NAME ${PROJECT_NAME})
10  set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
11  include(CPack)
12
```

Рисунок 4.1 – “CmakeLists.txt” до інтеграції бібліотеки

main.cpp CMakeLists.txt

main.cpp > ...

```
1   #include <iostream>
2
3   int main() {
4       std::cout << "Hello, world!\n";
5   }
6
```

Рисунок 4.2 – “main.cpp” до інтеграції бібліотеки

```
PS D:\testApp> D:\testApp\build\Debug\testProject.exe
Hello, world!
PS D:\testApp>
```

Рисунок 4.3 – Приклад роботи застосунку до інтеграції бібліотеки

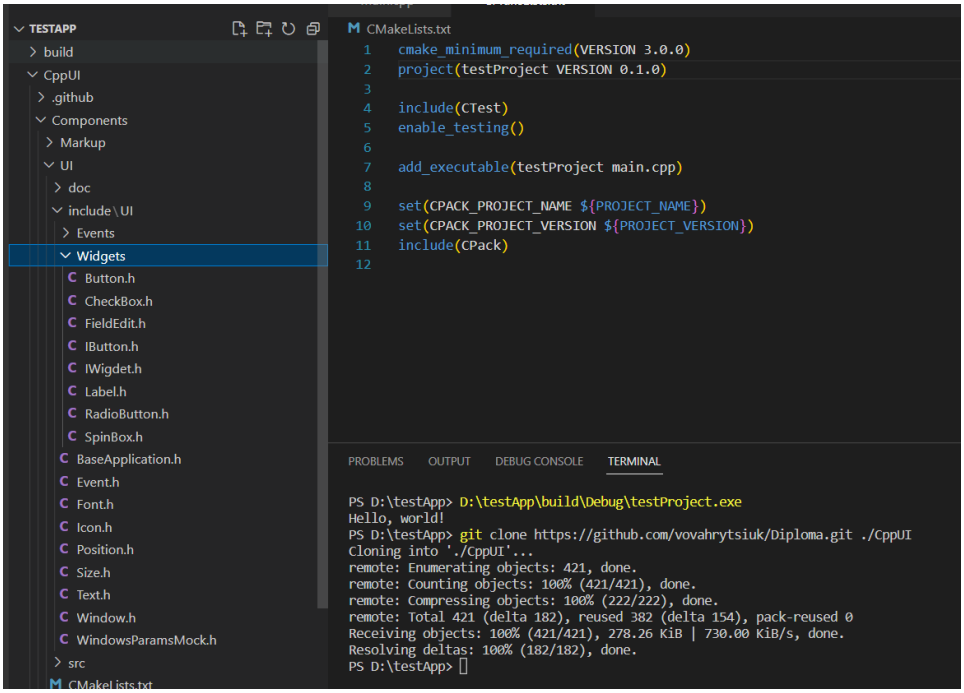



Рисунок 4.4 – Структура проекту після клонування бібліотеки

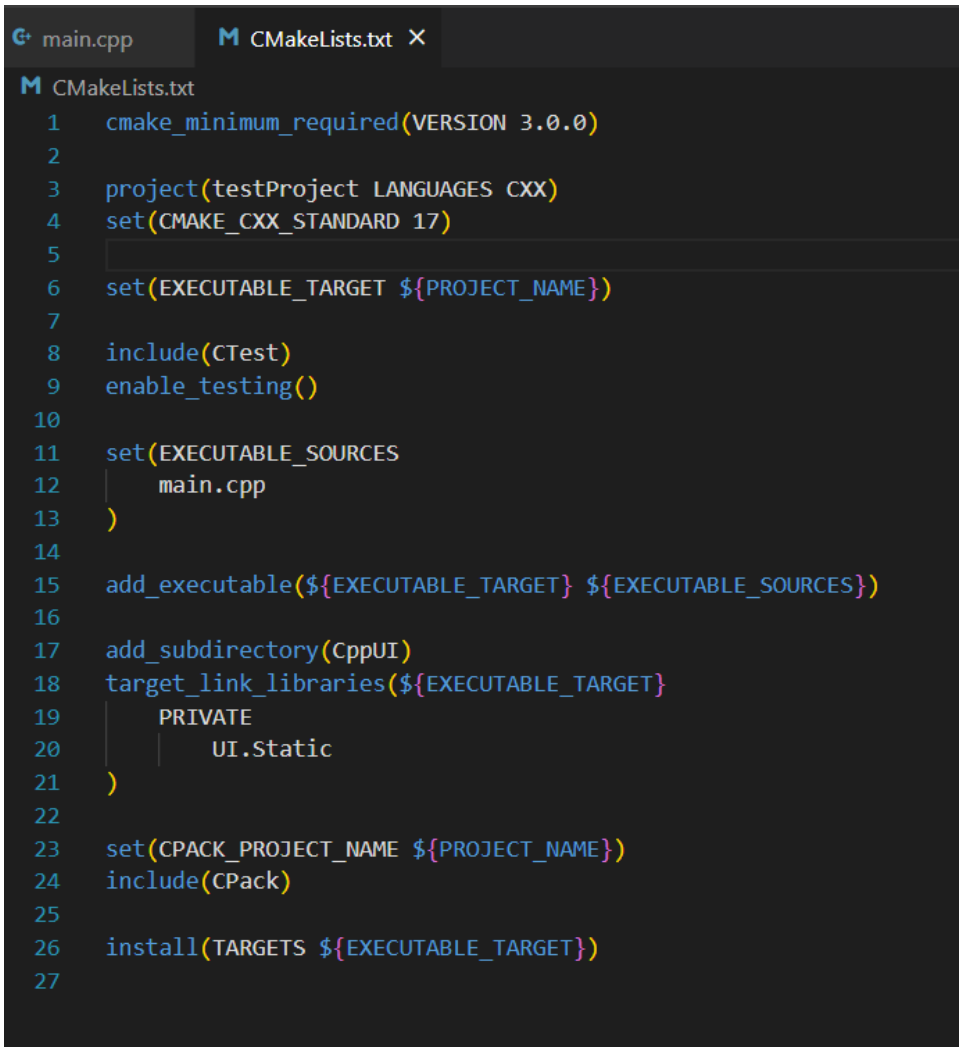



Рисунок 4.5 – “CmakeLists.txt” після інтеграції бібліотеки

Щоб створити простий графічний інтерфейс, що скаладається з одного вікна, розмірами 800 на 1200 та з заголовком “Test Project” потрібно створити клас Application та наслідується від класу BaseApplication, що є частиною бібліотеки. Для налаштування параметрів вікна потрібно створити файл розмітки з назвою “CppUI.txt”. Приклад такого файлу з описаними вище параметрами можна побачити на рисунку 4.6.

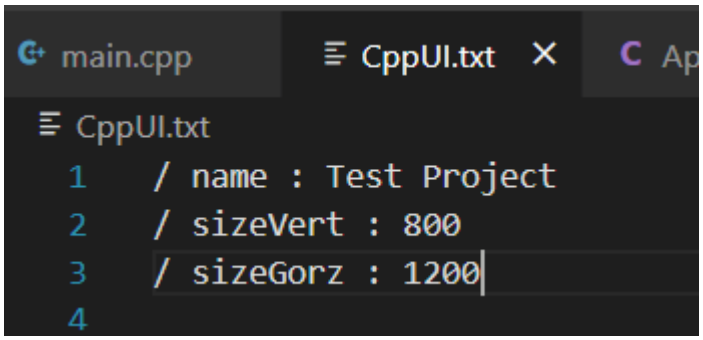



Рисунок 4.6 – Приклад файлу розмітки

Спроектований графічний інтерфейс представлений на рисунку 4.7


Рисунок 4.7 – Спроектований графічний інтерфейс

Щоб розмістити на вікні віджети, потрібно описати їх параметри в файлі розмітки. Для опрацювання взаємодії з віджетами (наприклад, натискання на

кнопку) потрібно у вигляді методу класу Application реалізувати очікувану поведінку та прив’язати її до відповідної події. Наприклад, на рисунку 4.8 можна побачити файл розмітки, що включає в себе опис параметрів вікна, кнопки та текстового напису. На рисунках 4.9 – 4.11 показано приклад програмування інтерфейсу з реалізацією поведінки, що спрацює при натисканні кнопки.

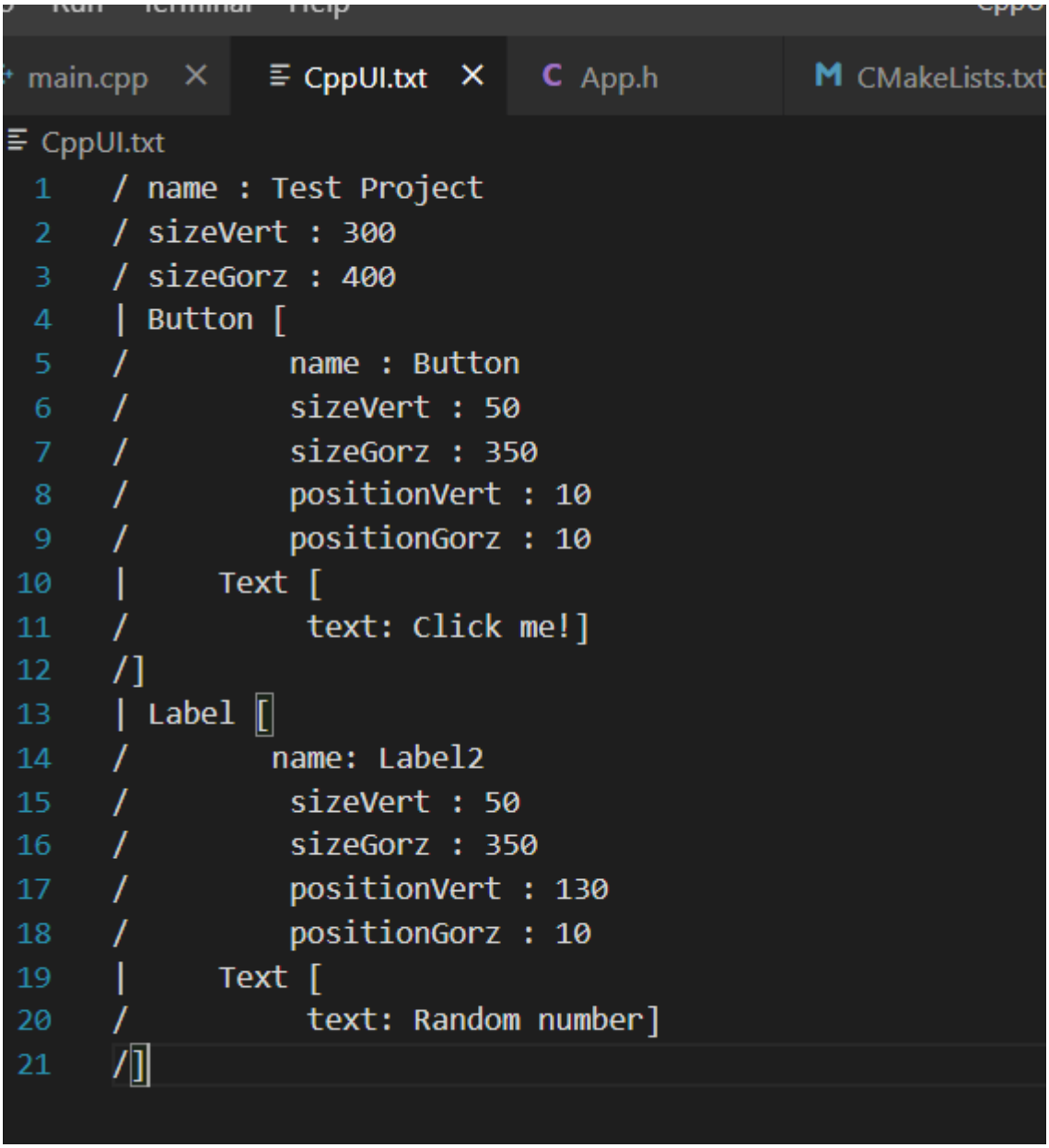



Рисунок 4.8 – Приклад файлу розмітки з кнопкою та текстовим написом

main.cpp | App.h | CMakeLists.txt | App.cp

App.h > Application > Application()

```
1   #pragma once
2
3   #include <UI/BaseApplication.h>
4
5   class Application : public BaseApplication
6   {
7   public:
8       Application();
9   private:
10      void setRandomNumberToLabel1();
11  };
```

Рисунок 4.9 – Файл “App.h”

main.cpp | App.cpp

App.cpp > Application()

```
1   #include "App.h"
2   #include <ctime>
3
4   void Application::setRandomNumberToLabel1()
5   {
6       auto* label = _mainWindow.findWidgetByName<Label>("Label1");
7       label->setText(std::to_string(std::rand()));
8   }
9
10  Application::Application()
11  {
12      auto* button = _mainWindow.findWidgetByName<Button>("Button1");
13      button->_click.connect([this](){this->setRandomNumberToLabel1();});
14  }
```

Рисунок 4.10 – Файл “App.cpp”

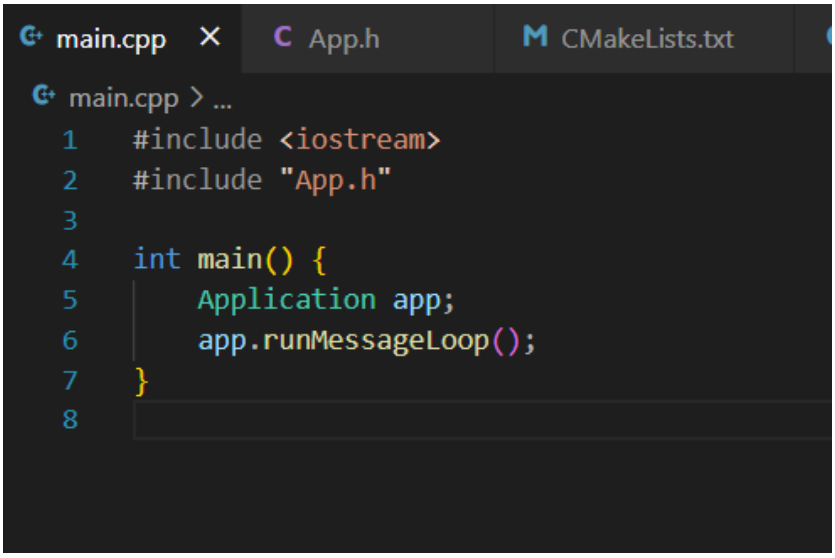



Рисунок 4.11 – Файл “main.cpp”

На рисунку 4.12 можна побачити спроектований інтерфейс.

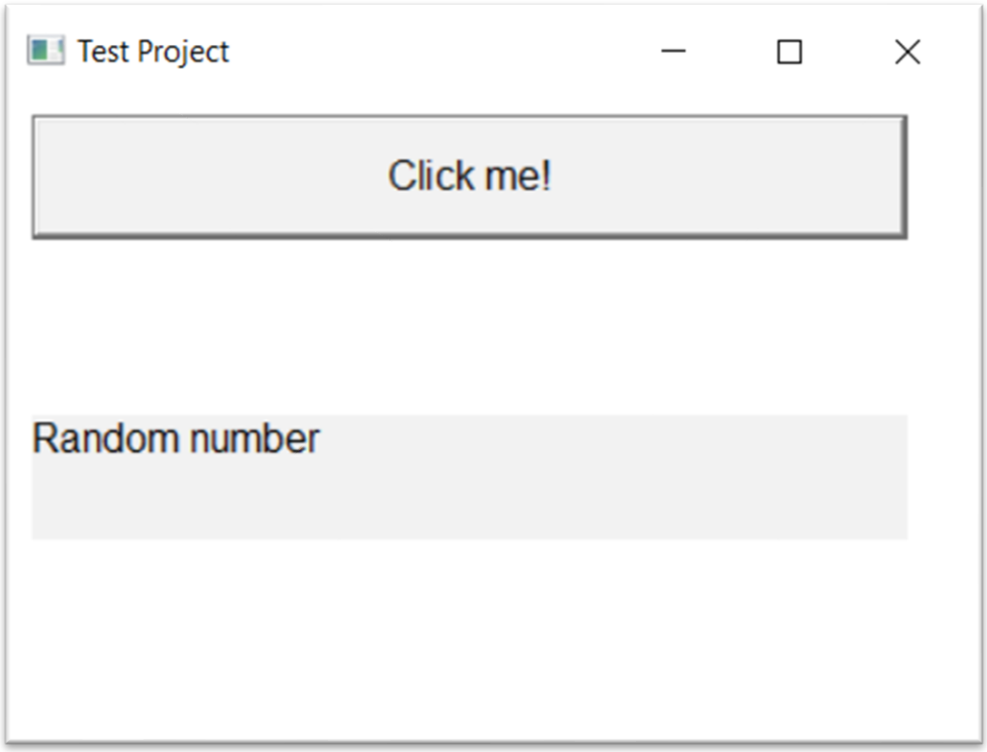



Рисунок 4.12 – Спроектований інтерфейс

На рисунку 4.13 можна побачити стан графічний інтерфейс після натискання кнопки.

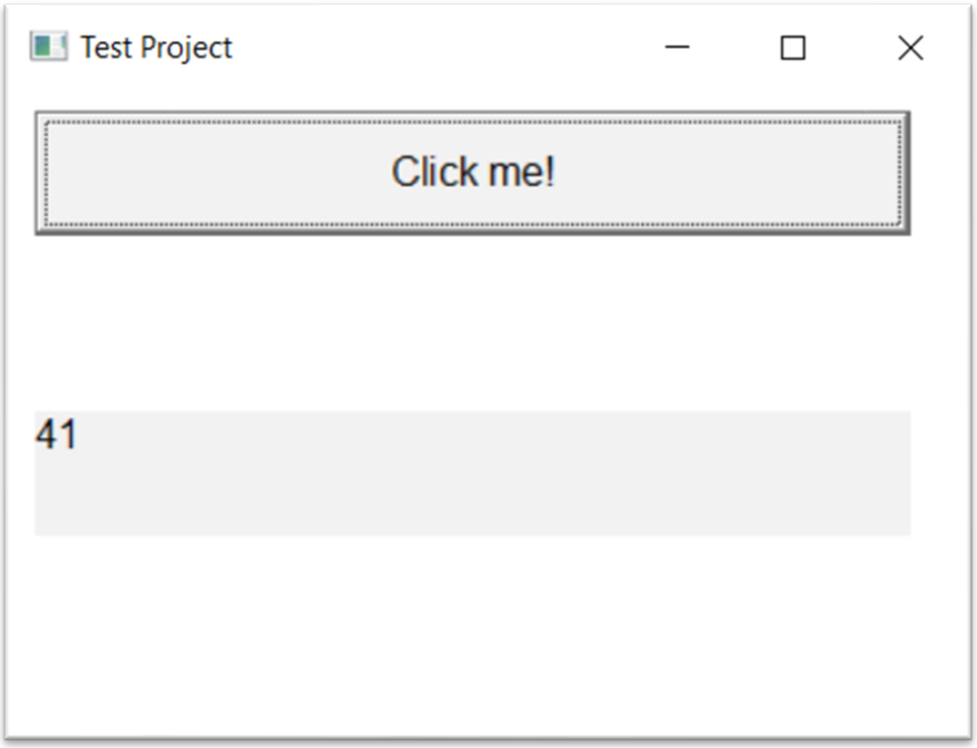



Рисунок 4.13 – Стан інтерфейсу після натискання на кнопку

## 4.2 Підтримка програмного забезпечення

Для отримання нової версії бібліотеки користувачу потрібно оновити завантажений вихідний код програмного забезпечення. Враховуючи вищеописаний алгоритм інтеграції CppUI, користувач може використати команду git pull для завантаження останніх змін з репозиторію. На рисунку 4.14 можна побачити приклад оновлення локальної копії бібліотеки.

```
PS D:\testApp> cd .\CppUI\
PS D:\testApp\CppUI> git pull
Already up to date.
PS D:\testApp\CppUI>
```

Рисунок 4.14 – Оновлення локальної копії бібліотеки

Після виконання описаних кроків, користувач отримає оновлену версію бібліотеки.

## 4.3 Висновки до розділу

В підрозділі 4.1 описано процес розгортання бібліотеки. Для розробленої програмного забезпечення цей процес складається з двох етапів, а саме лінкування за допомогою CMake та проектування інтерфейсу користувача, використовуючи функціонал бібліотеки.

Вихідний код зберігається у GitHub репозиторії. Звідти всі файли бібліотеки можуть бути вивантажені (клоновані) користувачем до кореневої директорії проекту, для якого потрібно розробити інтерфейс користувача.

Після успішного лінкування бібліотеки, використання зводиться до створення файлу розмітки, де задаються всі параметри віджетів, а також створення користувацького класу, що наслідується від класу BaseApplication і відповідає за контроль та управління всіма віджетами. Додатково, користувач може у вигляді методів користувацького класу визначити поведінку при виникненні тих чи інших подій взаємодії користувача з віджетами.

В підрозділі 4.2 описано отримання нової версії бібліотеки, що зводиться до вивантаження актуальної версії вихідного коду з GitHub репозиторію.

# ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано подійно-орієнтовану бібліотеку для проектування графічного інтерфейсу користувача на основі власної мови розмітки.

У першому розділі було проведено аналіз предметної області та відомих програмних продуктів, серед яких Qt, WinForms, GIMP Toolkit та wxWidgets. Кожен зі згаданих програмних продуктів має свої недоліки, що полягають у складності інтеграції інструменту до застосунку, оскільки ці засоби є технологіями та включають в себе набагато ширший функціонал, ніж потрібно для створення графічного інтерфейсу. Багатофункціональність призводить до суттєвого збільшення складності освоєння цих засобів і, в результаті, проектування графічного інтерфейсу користувача стає абсолютно нетривіальною задачею. Також у першому розділі було розроблено функціональні та нефункціональні вимоги.

У другому розділі було проведено моделювання та конструювання програмного забезпечення. Наведено модель бізнес процесу використання розробленого програмного забезпечення за допомогою BPMN діаграми. Для реалізації бібліотеки було обрано монолітну архітектуру, що логічно розділяється на два компоненти – UI та Markup. Компонент Markup відповідає за зчитування параметрів з файлу розмітки. Компонент UI відповідає за створення графічного інтерфейсу на основі отриманих параметрів розмітки, контроль за ресурсами бібліотеки, обробку взаємодій користувача з віджетами. Архітектуру деталізовано за допомогою діаграми класів, надано опис призначення та реалізації всіх сутностей. ПЗ не використовує чутливі дані користувача. Якщо користувач хоче реалізувати застосунок, що використовує дану бібліотеку, і передбачає зберігання особистих даних, про їх безпеку користувач повинен подбати самостійно.

У третьому розділі було проведено аналіз якості та тестування програмного забезпечення. За допомогою сервісу Embold було проведено

статичний аналіз коду за обраними метриками, а саме швидкодія, надійність, ефективність, стійкість та зручність використання. За всіма метриками отримано високу оцінку. Загальна оцінка коду проекту, що надана сервісом Embold, дорівнює 4,83 за п’ятибальною шкалою, що є хорошим результатом. Також у даному розділі показано, що бібліотека відповідає всім нефункціональним вимогам. Далі було описано процеси тестування розробленої бібліотеки. Тестування пройшло успішно, всі тести пройдені. У якості контрольного прикладу використання розробленого програмного забезпечення бібліотеки створено застосунок з графічним інтерфейсом користувача для обчислення коренів квадратного рівняння.

В 4 розділі описано процес розгортання бібліотеки, що складається з двох етапів, а саме лінкування за допомогою CMake та проектування інтерфейсу користувача, використовуючи функціонал бібліотеки та мову розмітки.

Практична значимість розробки полягає у спрощенні інтеграції засобів проектування графічного інтерфейсу користувача та полегшення процесу розробки інтерфейсу за допомогою зручної та інтуїтивно простої мови розмітки. Розроблена бібліотека може бути інтегрована до будь якого консольного застосунку, написаного на C++ під управлінням операційної системи Windows, всього лише за декілька кроків, що робить суттєвий прорив у засобах проектування графічного інтерфейсу користувача на мові програмування C++. Розроблена мова розмітки дозволяє швидко та зручно описати параметри віджетів та вікна у файлі розмітки за допомогою простого, чіткого та зрозумілого синтаксису. Розроблений механізм обробки подій в системі дозволяє швидко визначити методи чи функції з логікою, що відповідає очікуванням користувача, та підключити їх до відповідних подій.

Створення графічного інтерфейсу користувача – неймовірно широка область програмування,що включає в себе безліч можливостей для розвитку розробленої бібліотеки. Розробка може бути вдосконалена за рахунок додавання нестандартних віджетів, реалізація кросплатформенності, автоматичного

розміщення віджетів, автоматичного визначення розмірів віджетів, прив’язки розміщення та вирівнювання віджетів до певних контрольних точок. В результаті розроблена бібліотека стане незамінним інструментом користувача, який займається проектуванням графічного інтерфейсу на мові програмування C++.

В якості середовища розробки було обрано редактор коду Visual Studio Code та інструмент для збірки CMake. Поєднання цих засобів враховуючи всі їхні можливості, створю оптимальне оточення для розробки такого програмного забезпечення.

Після реалізації бібліотека була протестована мануальними тестами, що включали в себе тестування всіх компонентів з різними параметрами у файлах розмітки, щоб переконатися у правильності роботи бібліотеки. Всі тести пройдені успішно. З використанням бібліотеки було розроблено кілька програмних застосунків з графічним інтерфейсом користувача. Приклад такого застосунку – програмне забезпечення для вирішення квадратних рівнянь, що представлене у вигляді контрольного прикладу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Ферг С. Подійно-орієнтоване програмування: вступ, навчальний посібник, історія. / Стефан Ферг. – Лондон, 2006;
2. The Real History of the GUI. URL: https://www.sitepoint.com/real-history-gui/ (дата звернення: 03.05.2023);
3. James H. Coombs. Markup Systems and the Future of Scholarly Text Processing. URL: http://xml.coverpages.org/coombs.html (дата зверення: 7.05.2023);
4. Офіційна документація Qt. URL: https://doc.qt.io/ (дата звернення: 31.04.2024);
5. Офіційна документація Windows Forms. URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-7.0 (дата звернення: 01.05.2023);
6. Офіційна документація GIMP. URL: https://docs.gimp.org/ (дата звернення : 7.05.2023);
7. Офіційна документація WxWidgets. URL: https://docs.wxwidgets.org/3.0/ (дата звернення: 8.05.2023);
8. Bass L. Software Architecture in Practice / L. Bass, P. Clements, R. Kazman., 2021. – 464 с. – (4).
9. Патерн проектування: фабричний метод. URL: https://refactoring.guru/design-patterns/factory-method (дата звернення 11.05.2023);
10. Патерн проектування: спостерігач. URL: https://refactoring.guru/design-patterns/observer (дата звернення: 11.05.2023);

11. Офіційна документація Win32 Api. URL: https://learn.microsoft.com/en-us/windows/win32/api/ (дата звернення: 13.05.2023);

12. Офіційна документація Visual Studio. URL: https://learn.microsoft.com/en-us/visualstudio/?view=vs-2022 (дата звернення: 15.05.2023);

13. Офіційна документація Visual Studio Code: https://code.visualstudio.com/docs (дата звернення: 15.05.2023);

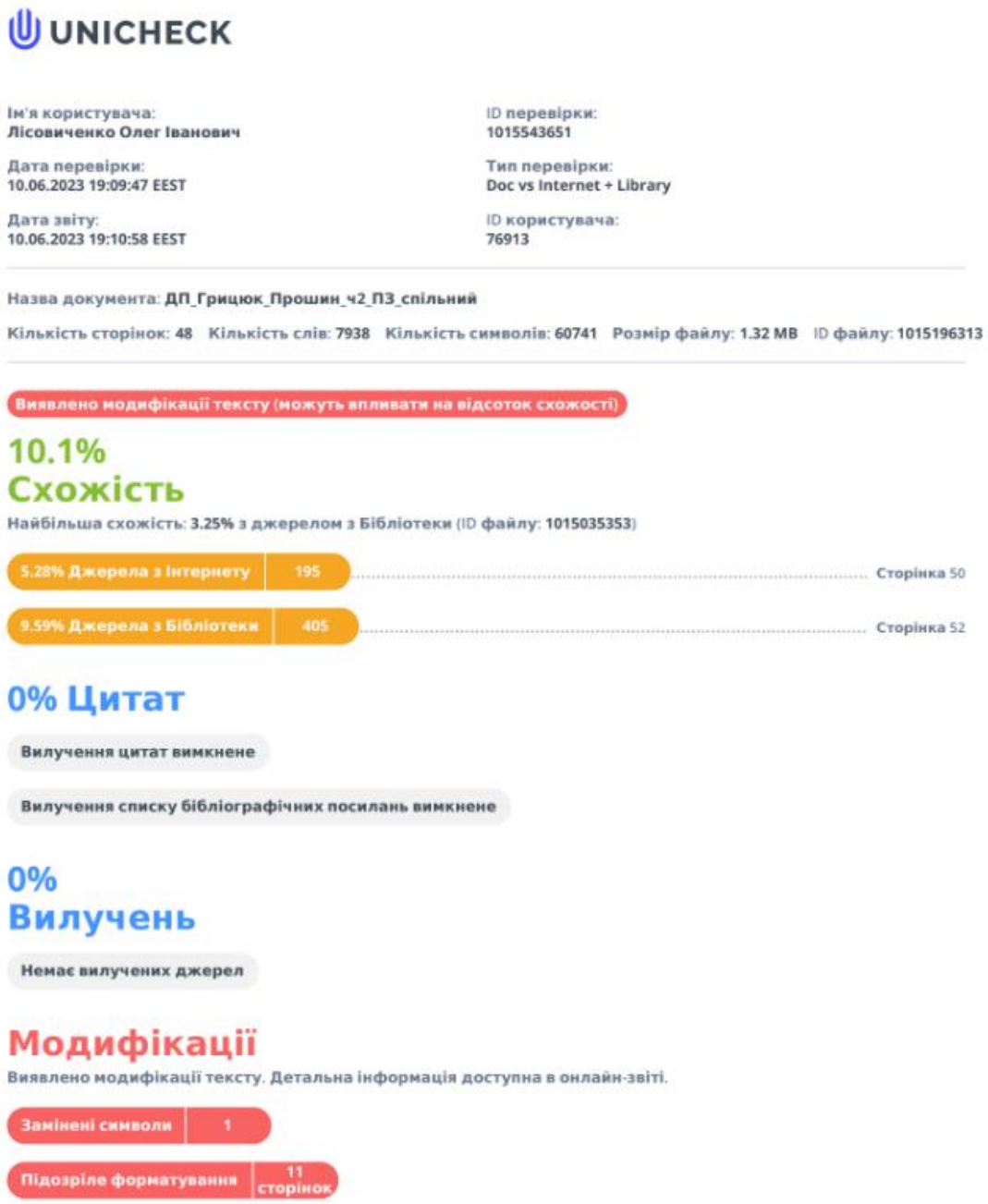
14. Офіційна документація Code::Blocks. URL: https://www.codeblocks.org/user-manual/ (дата звернення: 15.05.2023);

15. Офіційна документація Eclipse. URL: https://www.eclipse.org/documentation/ (дата звернення: 15.05.2023);

16. Офіційна документація Cmake. URL: https://cmake.org/documentation/ (дата звернення: 19.05.2023);

17. Офіційна документація Embold. URL: https://docs.embold.io/ (дата звернення: 23.05.2023).

# ДОДАТОК А ЗВІТ ПОДІБНОСТІ



UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015543651

Дата перевірки:
10.06.2023 19:09:47 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
10.06.2023 19:10:58 EEST

ID користувача:
76913

Назва документа: ДП_Грицюк_Прошин_ч2_ПЗ_спільний

Кількість сторінок: 48 Кількість слів: 7938 Кількість символів: 60741 Розмір файлу: 1.32 MB ID файлу: 1015196313

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

10.1%
Схожість

Найбільша схожість: 3.25% з джерелом з Бібліотеки (ID файлу: 1015035353)

5.28% Джерела з Інтернету 195 ..... Сторінка 50

9.59% Джерела з Бібліотеки 405 ..... Сторінка 52

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 1

Підозріле форматування 11 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

# ПОДІЙНО-ОРІЄНТОВАНА БІБЛІОТЕКА ДЛЯ СТВОРЕННЯ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ НА ОСНОВІ ВЛАСНОЇ МОВИ РОЗМІТКИ (комплексна тема)

**Текст програми**

КПІ.ІП-9209.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавці:
_____ Володимир ГИЦЮК
_____ Назарій ПРОШИН

Київ – 2023

## Файл Events.h

```
#pragma once


#include <functional>
#include <vector>

class Event
{
public:
    using EventHandler = std::function<void()>;

    void connect(EventHandler eventHandler)
    {
        _handlers.push_back(eventHandler);
    }

    void emit()
    {
        for (const auto &eventHandler : _handlers)
        {
            eventHandler();
        }
    }

private:
    std::vector<EventHandler> _handlers;
};
```

## Файл Button.h

```
#pragma once

#include "IButton.h"
#include "../Event.h"
#include <string>

class Button : public IButton
{
public:
    Button(WORD id, const std::string &name, const Size &size, const Position
&position, const Text &text, bool clickable, bool doubleClickable);

    ButtonType getButtonType() override
```

```
    {
        return ButtonType::PushButton;
    }

    bool getDoubleClickable() const
    {
        return _doubleClickable;
    }

private:
    bool _doubleClickable;

protected:
    int getStyles() override;

public:
    Event _doubleClick;
};
```

## Файл CheckBox.h

```
#pragma once

#include "IButton.h"
#include "../Event.h"
#include <string>

class CheckBox : public IButton
{
public:
    CheckBox(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text, bool clickable);

    bool isChecked() const
    {
        return (SendMessage(_hwnd, BM_GETCHECK, 0, 0) == BST_CHECKED);
    }

    ButtonType getButtonType() override
    {
        return ButtonType::CheckBox;
    }

protected:
```

```
    int getStyles() override;

private:
    void changeState()
    {
        SendMessage(_hwnd, BM_SETCHECK, (isChecked() ? BST_UNCHECKED :
BST_CHECKED), 0);
    }

private:
public:
};
```

## Файл FieldEdit.h

```
#pragma once

#include "IWigdet.h"
#include "../Event.h"
#include <string>

class FieldEdit : public IWidget
{
public:
    FieldEdit(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text = Text(), bool multiline = false);

protected:
    std::string getClassName() override
    {
        return "EDIT";
    }

    int getStyles() override;

public:
    bool _multiline;
    Event _textChanged;
};
```

## Файл IButton.h

```
#pragma once

#include "IWigdet.h"
```

```
#include "../Event.h"
#include <string>

enum class ButtonType
{
    PushButton,
    CheckBox,
    RadioButton
};

class IButton : public IWidget
{
public:
    IButton(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text, bool clickable) : IWidget(id, name,
size, position, text), _clickable{clickable} {}
    virtual ButtonType getButtonType() = 0;

    bool getClickable() const
    {
        return _clickable;
    }

    Event _click;

protected:
    std::string getClassName() override
    {
        return "Button";
    }

private:
    bool _clickable;
};
```

## Файл IWidget.h

```
#pragma once

#include "../Text.h"
#include "../Size.h"
#include "../Position.h"

#include <windows.h>
```

```
class Window;

class IWidget
{
protected:
    HWND _hwnd;
    WORD _id;
    std::string _name;
    Size _size;
    Position _position;
    Text _text;

    virtual std::string getClassName() = 0;
    virtual int getStyles() = 0;
    virtual int getExStyle();

public:
    virtual bool render(Window &parent);

    std::string getText()
    {
        int length = GetWindowTextLengthW(_hwnd);
        if (length == 0)
            return "";

        // Allocate memory for the text
        char *buffer = new char[length + 1];

        // Retrieve the text of the static control
        GetWindowText(_hwnd, buffer, length + 1);

        // Convert to wstring
        std::string text(buffer);

        // Clean up and return the text
        delete[] buffer;
        return text;
    }

    int getId()
    {
        return _id;
```

```
    }

    std::string getName() const
    {
        return _name;
    }

    void setText(const std::string &text)
    {
        SetWindowText(_hwnd, text.c_str());
    }

    IWidget(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text) : _hwnd(NULL), _id{id}, _name{name},
_size{size}, _position{position}, _text{text} {}
    virtual ~IWidget() = default;
    HWND get_hwnd() const { return _hwnd; }
};
```

## Файл Label.h

```
#pragma once

#include "IWigdet.h"
#include <string>

class Label : public IWidget
{
public:
    Label(WORD id, const std::string &name, const Size &size, const Position
&position, const Text &text);

protected:
    std::string getClassName() override
    {
        return "STATIC";
    }

    int getStyles() override;

private:
    std::string _className = "Static";
};
```

## Файл RadioButton.h

```
#pragma once

#include "IButton.h"
#include "../Event.h"
#include <string>

class RadioButton : public IButton
{
public:
    RadioButton(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text, bool clickable, bool newGroup);

    ButtonType getButtonType() override
    {
        return ButtonType::RadioButton;
    }

protected:
    int getStyles() override;

private:
private:
    bool _newGroup;
};
```

## Файл SpinBox.h

```
#pragma once

#include "IWigdet.h"
#include <string>
#include <CommCtrl.h>

class Edit : public IWidget
{

public:
    Edit(WORD id, const std::string &name, const Size &size, const Position
&position);

    bool render(Window &parent) override;

protected:
```

```
    int getExStyle() override;

    std::string getClassName() override;

    int getStyles() override;

public:
    // Event _textChanged;
};

// clickable
class SpinBox : public IWidget
{
public:
    SpinBox(WORD id, const std::string &name, int upperLimit, int lowerLimit,
int defaultPosition, WORD editFieldId);

    bool render(Window &parent) override;

protected:
    std::string getClassName() override;

    int getStyles() override;

private:
    int _upperLimit;
    int _lowerLimit;
    int _defaultPos;
    WORD _editFieldId;
    HWND _edit;
};
```

## Файл BaseApplication.h

```
#pragma once

#include "Window.h"
#include <string>
#include <CommCtrl.h>
#include "Widgets/Button.h"
#include "Widgets/Label.h"
#include "Widgets/CheckBox.h"
```

```
#include "WindowsParamsMock.h"
#include "Widgets/FieldEdit.h"
#include "Widgets/SpinBox.h"
#include "Widgets/RadioButton.h"

class BaseApplication
{
public:
    BaseApplication(const Window &params)
        : _hInstance{GetModuleHandle(NULL)}, _mainWindow{params._title,
Size(params._height, params._width), _hInstance}

    {
        createMainWindowWidgets(params);
        _mainWindow.render();
    }

    void runMessageLoop()
    {
        MSG msg = {};
        while (GetMessage(&msg, NULL, 0, 0) > 0)
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    Window &GetMainWindow()
    {
        return _mainWindow;
    }

private:
    template <class ParamsTypeT, typename CreatorFunc>
    void createWidgets(const std::vector<ParamsTypeT> &params, CreatorFunc
creatorFunc)
    {
        for (const auto &param : params)
        {
            (this->*creatorFunc)(param);
        }
    }
```

```
    void createMainWindowWidgets(const Params::Window &params);
    void createButton(const Params::Button &params);
    void createLabel(const Params::Label &params);
    void createFieldEdit(const Params::FieldEdit &params);
    void createSpinBox(const Params::SpinBox &params);
    void createRadioButton(const Params::RadioButton &params);
    void createCkeckBox(const Params::CheckBox &params);

protected:
    HINSTANCE _hInstance; // Application handle
    Window _mainWindow;   // Main window of the application
};
```

## Файл Font.h

```
#pragma once

#include <Windows.h>
#include <string>

class Font
{
public:
    Font(int height = 18, bool italic = false, bool underline = false, bool
strikeOut = false, const std::string &faceName = "Calibri")
        : _height{height}, _italic{italic}, _underline{underline},
_strikeOut{strikeOut}, _handle{NULL}, _faceName{faceName}
    {
    }

    Font(const Font &font)
    {
        _height = font._height;
        _italic = font._italic;
        _underline = font._underline;
        _strikeOut = font._strikeOut;
        _handle = NULL;
    }

    ~Font()
    {
        if (_handle != NULL)
        {
            DeleteObject(_handle);
        }
```

```
    }

    auto getHandle()
    {
        if (_handle == NULL)
        {
            _handle = CreateFont(_height, 0, 0, 0, FW_NORMAL, _italic,
_underline, _strikeOut, DEFAULT_CHARSET, OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_DONTCARE,
_faceName.c_str());
        }
        return _handle;
    }

private:
    int _height;
    bool _italic;
    bool _underline;
    bool _strikeOut;
    HFONT _handle;
    std::string _faceName;
};
```

## Файл Position.h

```
#pragma once

class Position final
{
public:
    Position(int x, int y)
        : _x{x}, _y{y} {}

    int getY() const
    {
        return _y;
    }

    int getX() const
    {
        return _x;
    }

private:
    int _x;
```

```
    int _y;
};
```

**Файл Size.h**

```
#pragma once

class Size final
{
public:
    Size(int height, int width)
        : _height{height}, _width{width} {}

    int getWidth() const
    {
        return _width;
    }

    int getHeight() const
    {
        return _height;
    }

private:
    int _height;
    int _width;
};
```

**Файл Text.h**

```
#pragma once

#include "Font.h"
#include <string>

class Text final
{
public:
    Text(const std::string &text = "", const Font &font = Font())
        : _text{text}, _font(font)
    {
    }

    auto &getText()
    {
        return _text;
    }
```

```
    auto &getFont()
    {
        return _font;
    }

private:
    std::string _text;
    Font _font;
};
```

## Файл Window.h

```
#pragma once

#include "Widgets/IWigdet.h"
#include <memory>
#include "Widgets/Button.h"
#include "Icon.h"

#include <windows.h>
#include <string>
#include <vector>
#include <unordered_map>
#include <stdexcept>

class Window
{
public:
    Window(const std::string &title, const Size &size, HINSTANCE hInstance)
        : _title{title}, _size{size}, _hInstance{hInstance}, _isShown{true}
    {
        WNDCLASS wc = {};

        wc.lpfnWndProc = Window::WindowProc;
        wc.hInstance = hInstance;
        wc.lpszClassName = _className.c_str();

        RegisterClass(&wc);
    }

    template <class Widget, typename... Args>
    WORD addWidget(const std::string &name, Args... args)
    {
        WORD id = static_cast<WORD>(_widgets.size());
```

```
        _widgets[id] = (std::make_unique<Widget>(id, name, args...));
        _widgetNameToId[name] = id;
        return id;
    }

    template <class Widget>
    Widget *findWidgetById(WORD id)
    {
        if (_widgets.count(id))
        {
            return dynamic_cast<Widget *>(_widgets.at(id).get());
        }
        return nullptr;
    }

    template <class Widget>
    Widget *findWidgetByName(const std::string &name)
    {
        if (_widgetNameToId.count(name))
        {
            return findWidgetById<Widget>(_widgetNameToId[name]);
        }
        return nullptr;
    }

    void render()
    {
        // Render window
        create();
        // Show Window
        show();
        // Render Widgets
        renderWidgets();
    }

    void renderWidgets()
    {
        for (const auto &[key, value] : _widgets)
        {
            if (!value->render(*this))
            {
                throw std::runtime_error("Widget with name =\"" +
value.get()->getName() + "\" not rendered");
```

```
        }
    }
}

bool create()
{
    _hwnd = CreateWindowEx(
        0,                    // Optional window styles.
        _className.c_str(),   // Window class
        _title.c_str(),       // Window text
        WS_OVERLAPPEDWINDOW, // Window style

        // Size and position
        CW_USEDEFAULT, CW_USEDEFAULT,
        _size.getWidth(), _size.getHeight(),

        NULL,       // Parent window
        NULL,       // Menu
        _hInstance, // Instance handle
        NULL        // Additional application data
    );

    SetWindowLongPtr(_hwnd, GWLP_USERDATA, (LONG_PTR)this);

    SetClassLongPtr(_hwnd,      // window handle
                    GCLP_HICON, // changes icon
                    (LONG_PTR)_icon.getHandle());

    return _hwnd != NULL;
}

void show()
{
    ShowWindow(_hwnd, _isShown);
}

void SetTitle(const std::string &title)
{
    _title = title;
    SetWindowText(_hwnd, _title.c_str());
}

HWND getHandle()
```

```
    {
        return _hwnd;
    }

    auto getHinstance()
    {
        return _hInstance;
    }

private:
    template <class Widget>
    Widget *defineWidget(WORD id)
    {
        auto &widget = _widgets[id];
        return dynamic_cast<Widget *>(widget.get());
    }

private:
    std::string _className = "Window";

    std::string _title;
    Size _size;
    bool _isShown;
    HINSTANCE _hInstance;
    HWND _hwnd;
    Icon _icon;

    std::unordered_map<WORD, std::unique_ptr<IWidget>> _widgets;
    std::unordered_map<std::string, WORD> _widgetNameToId;

    static LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam);
};
```

**Файл Button.cpp**

```
#include "UI/Widgets/Button.h"

Button::Button(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text, bool clickable, bool doubleClickable)
    : IButton(id, name, size, position, text, clickable),
_doubleClickable{doubleClickable}
{
}
```

```
int Button::getStyles()
{
    return WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON | BS_NOTIFY;
}
```

## Файл CheckBox.cpp

```
#include "UI/Widgets/CheckBox.h"

CheckBox::CheckBox(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text, bool clickable)
    : IButton(id, name, size, position, text, clickable)
{
    _click.connect([this]()
                   { this->changeState(); });
}

int CheckBox::getStyles()
{
    return WS_CHILD | WS_VISIBLE | BS_CHECKBOX | WS_BORDER;
}
```

## Файл FieldEdit.cpp

```
#include "UI/Widgets/FieldEdit.h"

FieldEdit::FieldEdit(WORD id, const std::string &name, const Size &size,
const Position &position, const Text &text, bool multiline)
    : IWidget(id, name, size, position, text), _multiline{multiline}
{
}

int FieldEdit::getStyles()
{
    int styles = WS_CHILD | WS_VISIBLE | WS_BORDER | ES_AUTOHSCROLL;
    if (_multiline)
    {
        styles |= WS_VSCROLL |
                  ES_LEFT | ES_MULTILINE | ES_AUTOVSCROLL;
    }
    else
    {
        styles |= ES_AUTOHSCROLL;
    }
    return styles;
}
```

## Файл IWidget.cpp

```
#include <UI/Widgets/IWigdet.h>
#include <UI/Window.h>

bool IWidget::render(Window &parent)
{
    _hwnd = CreateWindowEx(
        getExStyle(),
        getClassName().c_str(),  // Predefined class; Unicode assumed
        _text.getText().c_str(), // Text
        getStyles(),             // Styles
        _position.getX(),        // x position
        _position.getY(),        // y position
        _size.getWidth(),        // Width
        _size.getHeight(),       // Height
        parent.getHandle(),      // Parent window
        (HMENU)_id,              // No menu.
        (HINSTANCE)GetWindowLongPtr(parent.getHandle(), GWLP_HINSTANCE),
        NULL); // Pointer not needed.

    SendMessage(_hwnd, WM_SETFONT, (WPARAM)_text.getFont().getHandle(),
MAKELPARAM(TRUE, 0));

    return _hwnd != NULL;
}

int IWidget::getExStyle()
{
    return 0;
}
```

## Файл Label.cpp

```
#include "UI/Widgets/Label.h"

Label::Label(WORD id, const std::string &name, const Size &size, const
Position &position, const Text &text)
    : IWidget(id, name, size, position, text)
{
}

int Label::getStyles()
{
    return WS_TABSTOP | WS_VISIBLE | WS_CHILD | SS_LEFT;
```

```
}
```

## Файл RadioButton.cpp

```
#include "UI/Widgets/RadioButton.h"

RadioButton::RadioButton(WORD id, const std::string &name, const Size &size,
const Position &position, const Text &text, bool clickable, bool newGroup)
    : IButton(id, name, size, position, text, clickable), _newGroup{newGroup}
{
}

int RadioButton::getStyles()
{
    int styles = WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON;
    if (_newGroup)
    {
        styles |= WS_GROUP;
    }
    return styles;
}
```

## Файл SpinBox.cpp

```
#include "UI/Widgets/SpinBox.h"
#include <Commctrl.h>
#include <UI/Window.h>
#include <strsafe.h>

Edit::Edit(WORD id, const std::string &name, const Size &size, const Position
&position)
    : IWidget(id, name, size, position, Text())
{
}

std::string Edit::getClassName()
{
    return WC_EDIT;
}

int Edit::getStyles()
{
    return WS_CHILD | WS_VISIBLE | ES_RIGHT | ES_READONLY;
}

int Edit::getExStyle()
```

```
{
    return WS_EX_CLIENTEDGE;
}

bool Edit::render(Window &parent)
{
    _hwnd = CreateWindowEx(getExStyle(), getClassName().c_str(), NULL,
getStyles(), _position.getX(), _position.getY(), _size.getWidth(),
_size.getHeight(), parent.getHandle(),
                           (HMENU)_id, NULL, NULL);
    return _hwnd != NULL;
}

SpinBox::SpinBox(WORD id, const std::string &name, int upperLimit, int
lowerLimit, int defaultPosition, WORD editFieldId)
    : IWidget(id, name, Size{0, 0}, Position{0, 0}, Text()),
_upperLimit{upperLimit}, _lowerLimit{lowerLimit},
_defaultPos{defaultPosition}, _editFieldId{editFieldId}
{
}

std::string SpinBox::getClassName()
{
    return UPDOWN_CLASS;
}

int SpinBox::getStyles()
{
    return WS_CHILDWINDOW | WS_VISIBLE | UDS_AUTOBUDDY | UDS_SETBUDDYINT |
UDS_ALIGNRIGHT | UDS_ARROWKEYS | UDS_HOTTRACK;
}

bool SpinBox::render(Window &parent)
{
    auto *edit = parent.findWidgetById<Edit>(_editFieldId);
    if (edit->get_hwnd())
    {
        INITCOMMONCONTROLSEX icex;

        icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
        icex.dwICC = ICC_UPDOWN_CLASS;
        InitCommonControlsEx(&icex);
```

```
        _hwnd = CreateWindow(getClassName().c_str(), NULL, getStyles(),
                             _position.getX(),
                             _position.getY(),
                             _size.getWidth(),
                             _size.getHeight(),
                             parent.getHandle(), (HMENU)_id, NULL, NULL);

        SendMessageW(_hwnd, UDM_SETBUDDY, (WPARAM)(edit->get_hwnd()), 0);
        SendMessageW(_hwnd, UDM_SETRANGE, 0, MAKELPARAM(_upperLimit,
_lowerLimit));
        SendMessageW(_hwnd, UDM_SETPOS32, 0, _defaultPos);
    }

    return _hwnd != NULL;
}
```

## Файл BaseApplication.cpp

```
#include <UI/BaseApplication.h>

void BaseApplication::createMainWindowWidgets(const Params::Window &params)
{
    createWidgets(params._buttons, &BaseApplication::createButton);
    createWidgets(params._checkBoxes, &BaseApplication::createCkeckBox);
    createWidgets(params._labels, &BaseApplication::createLabel);
    createWidgets(params._spinBoxes, &BaseApplication::createSpinBox);
    createWidgets(params._lineEdits, &BaseApplication::createFieldEdit);
    createWidgets(params._radioButtons, &BaseApplication::createRadioButton);
}

void BaseApplication::createButton(const Params::Button &params)
{
    _mainWindow.addWidget<Button>(params._name, Size{params._height,
params._width}, Position{params._x, params._y}, Text{params._text},
params._clickable, params._doubleClickable);
}

void BaseApplication::createLabel(const Params::Label &params)
{
    _mainWindow.addWidget<Label>(params._name, Size{params._height,
params._width}, Position{params._x, params._y}, Text{params._text});
}

void BaseApplication::createFieldEdit(const Params::FieldEdit &params)
```

```
{
    _mainWindow.addWidget<FieldEdit>(params._name, Size(params._height,
params._width), Position(params._x, params._y), Text(""));
}

void BaseApplication::createSpinBox(const Params::SpinBox &params)
{
    auto id = _mainWindow.addWidget<Edit>(params._name + "DefaultSystemEdit",
Size(params._height, params._width), Position(params._x, params._y));
    _mainWindow.addWidget<SpinBox>(params._name, params._upper,
params._lower, params._default, id);
}

void BaseApplication::createRadioButton(const Params::RadioButton &params)
{
    _mainWindow.addWidget<RadioButton>(params._name, Size{params._height,
params._width}, Position{params._x, params._y}, Text{params._text},
params._clickable, params._newGroup);
}

void BaseApplication::createCkeckBox(const Params::CheckBox &params)
{
    _mainWindow.addWidget<CheckBox>(params._name, Size{params._height,
params._width}, Position{params._x, params._y}, Text{params._text},
params._clickable);
}
```

## Файл Window.cpp

```
#include <UI/Window.h>
#include <UI/Widgets/Button.h>
#include <UI/Widgets/CheckBox.h>
#include <UI/Widgets/FieldEdit.h>

LRESULT CALLBACK Window::WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
    Window *pThis = NULL;
    if (uMsg == WM_NCCREATE)
    {
    }
    else
    {
        pThis = (Window *)GetWindowLongPtr(hwnd, GWLP_USERDATA);
```

```
    }

    if (pThis)
    {
        switch (uMsg)
        {
        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd, &ps);

            // All painting occurs here, between BeginPaint and EndPaint.

            FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR_WINDOW + 1));

            EndPaint(hwnd, &ps);
            return 0;
        }
        case WM_COMMAND:
            switch (HIWORD(wParam))
            {
            case BN_CLICKED:
            {
                auto *button = pThis->defineWidget<IButton>(LOWORD(wParam));
                if (button && button->getClickable())
                {
                    button->_click.emit();
                }
                break;
            }
            case BN_DBLCLK:
            {
                auto *button = pThis->defineWidget<Button>(LOWORD(wParam));
                if (button && button->getDoubleClickable())
                {
                    button->_doubleClick.emit();
                }
                break;
            }
```

```
            case EN_CHANGE:
            {
                auto *lineEdit = pThis-
>defineWidget<FieldEdit>(LOWORD(wParam));
                if (lineEdit)
                {
                    lineEdit->_textChanged.emit();
                }
                break;
            }
            }
            return 0;
        }
    }

    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```

## Файл Application.h

```
#pragma once
#include <UI/BaseApplication.h>
#include <string>

/**
 * @brief _ax^2 + _bx + _c = 0
 */
class QuadraticEquation
{
public:
    QuadraticEquation(double a, double b, double c) : _a{a}, _b{b}, _c{c} {}
    std::string getRootsString(bool printDiscriminant) const
    {
        std::string answer = "";
        double d = _b * _b - 4 * _a * _c;
        if (printDiscriminant)
        {
            answer.append("Discriminant = " + std::to_string(d) + "\n");
        }
        if (d < 0)
        {
            answer.append("Quadratic equation has no roots\n");
        }
        else if (d == 0)
```

```
        {
            answer.append("Quadratic equation has one root\n");
            answer.append("x = " + std::to_string(-1 * _b / 2 / _a));
        }
        else
        {
            answer.append("Quadratic equation has two roots\n");
            answer.append("x1 = " + std::to_string((-1 * _b + sqrt(d)) / 2 /
_a));
            answer.append("\nx2 = " + std::to_string((-1 * _b - sqrt(d)) / 2
/ _a));
        }
        return answer;
    }

private:
    double _a;
    double _b;
    double _c;
};

class Application : public BaseApplication
{
public:
    Application();

    void calculate()
    {
        try
        {
            auto *labelA =
_mainWindow.findWidgetByName<FieldEdit>("Coefficient_X_Squared");
            auto *labelB =
_mainWindow.findWidgetByName<FieldEdit>("Coefficient_X");
            auto *labelC =
_mainWindow.findWidgetByName<FieldEdit>("Coefficient_Free_Member");
            QuadraticEquation eq(
                std::stod(labelA->getText().c_str()),
                std::stod(labelB->getText().c_str()),
                std::stod(labelC->getText().c_str()));
            auto *checkBoxShowDisc =
_mainWindow.findWidgetByName<CheckBox>("CheckBox_Show_Discriminant");
```

```
            auto *label =
_mainWindow.findWidgetByName<Label>("Label_Answer");
            label->setText(eq.getRootsString(checkBoxShowDisc->isChecked()));
        }
        catch (...)
        {
            auto *label =
_mainWindow.findWidgetByName<Label>("Label_Answer");
            label->setText("Invalid arguments");
        }
    }

private:
};
```

## Файл main.cpp

```
#include "Application.h"

int main()
{
    Application app;
    app.runMessageLoop();
}
```

## Файл Button.cpp

```
#include "Button.h"
#include <stdlib.h>
#include <time.h>
#include <string>

Button::Button()
{
      srand(time(NULL));
      name = std::to_string(rand());
}

void Button::setSizeVert(int info)
{
      sizeVert = info;
      std::cout << "Set sizeVert: " << info << " for " << name << '\n';
}

int Button::getSizeVert()
{
```

```
	return sizeVert;

}

void Button::setSizeGorz(int info)
{
	sizeGorz = info;
	std::cout << "Set sizeGorz: " << info << " for " << name << '\n';
}

int Button::getSizeGorz()
{
	return sizeGorz;
}

void Button::setPositionVert(int info)
{
	positionVert = info;
	std::cout << "Set positionVert: " << info << " for " << name << '\n';
}

int Button::getPositionVert()
{
	return positionVert;
}

void Button::setPositionGorz(int info)
{
	positionGorz = info;
	std::cout << "Set positionGorz: " << info << " for " << name << '\n';
}

int Button::getPositionGorz()
{
	return positionGorz;
}

void Button::setName(std::string info)
{
	name = info;
	std::cout << "Set name: " << info << " for " << name << '\n';
}
```

```
std::string Button::getName()
{
	return name;
}

void Button::setClicable(bool info)
{
	clicable = info;
	std::cout << "Set clicable " << info << "for " << name << '\n';
}

bool Button::getClicable()
{
	return clicable;
}

void Button::setDoubleClicable(bool info)
{
	doubleClicable = info;
	std::cout << "Set doubleClicable " << info << "for " << name << '\n';
}

bool Button::getDoubleClicable()
{
	return doubleClicable;
}
```

## Файл Button.h

```
#pragma once
#include <iostream>
#include "Text.h"
class Button
{
public:
	Button();
	~Button() = default;


	void setSizeVert(int info);
	int  getSizeVert();
	void setSizeGorz(int info);
	int  getSizeGorz();
```

```
	void setPositionVert(int info);
	int  getPositionVert();
	void setPositionGorz(int info);
	int  getPositionGorz();
	void setName(std::string info);
	std::string getName();
	void setClicable(bool info);
	bool getClicable();
	void setDoubleClicable(bool info);
	bool getDoubleClicable();


	Text text;
private:
	std::string name;
	int sizeVert = 72;
	int sizeGorz = 120;
	int positionVert = 0;
	int positionGorz = 0;

	bool clicable = false;
	bool doubleClicable = false;
};
```

## Файл CheckBox.cpp

```
#include "CheckBox.h"
#include <stdlib.h>
#include <time.h>
#include <string>

CheckBox::CheckBox()
{
	srand(time(NULL));
	name = std::to_string(rand());
}

void CheckBox::setPositionVert(int info)
{
	positionVert = info;
	std::cout << "Set positionVert: " << info << " for " << name << '\n';
}
```

```
int CheckBox::getPositionVert()
{
	return positionVert;
}

void CheckBox::setPositionGorz(int info)
{
	positionGorz = info;
	std::cout << "Set positionGorz: " << info << " for " << name << '\n';
}

int CheckBox::getPositionGorz()
{
	return positionGorz;
}

void CheckBox::setName(std::string info)
{
	name = info;
	std::cout << "Set name: " << info << " for " << name << '\n';
}

std::string CheckBox::getName()
{
	return name;
}

void CheckBox::setDefaultValue(bool info)
{
	defaultValue = info;
	std::cout << "Set defaultValue: " << info << " for " << name << '\n';
}

bool CheckBox::setDefaultValue()
{
	return defaultValue;
}

void CheckBox::setClicable(bool info)
{
	clicable = info;
	std::cout << "Set clicable " << info << "for " << name << '\n';
}
```

```
bool CheckBox::getClicable()
{
	return clicable;
}
```

## Файл CheckBox.h

```
#pragma once
#include <iostream>
#include "Text.h"
class CheckBox
{
public:
	CheckBox();
	~CheckBox() = default;


	void setPositionVert(int info);
	int  getPositionVert();
	void setPositionGorz(int info);
	int  getPositionGorz();
	void setName(std::string info);
	std::string getName();
	void setDefaultValue(bool info);
	bool setDefaultValue();
	void setClicable(bool info);
	bool getClicable();

	Text text;

private:
	std::string name;
	int positionVert = 0;
	int positionGorz = 0;
	bool defaultValue = false;
	bool clicable = false;

};
```

## Файл diploma.cpp

```
#include "Parcer.h"
```

```
int main()
{
	Parcer parcer;
	bool parcingStatus = true;

	if (parcer.checkFile())
	{
		if (parcer.parcing())
		{

		}
		else
		{
			parcingStatus = false;
		}
	}
	else
	{
		parcingStatus = false;
		std::cout << parcer.getError();
	}

	//std::cout << parcer.getWARN();
}
```

## Файл EditLine.cpp

```
#include "EditLine.h"
#include <stdlib.h>
#include <time.h>
#include <string>

EditLine::EditLine()
{
	srand(time(NULL));
	name = std::to_string(rand());
}

void EditLine::setSizeVert(int info)
{
	sizeVert = info;
	std::cout << "Set sizeVert: " << info << " for " << name << '\n';
}
```

```
int EditLine::getSizeVert()
{
	return sizeVert;

}

void EditLine::setSizeGorz(int info)
{
	sizeGorz = info;
	std::cout << "Set sizeGorz: " << info << " for " << name << '\n';
}

int EditLine::getSizeGorz()
{
	return sizeGorz;
}

void EditLine::setPositionVert(int info)
{
	positionVert = info;
	std::cout << "Set positionVert: " << info << " for " << name << '\n';
}

int EditLine::getPositionVert()
{
	return positionVert;
}

void EditLine::setPositionGorz(int info)
{
	positionGorz = info;
	std::cout << "Set positionGorz: " << info << " for " << name << '\n';
}

int EditLine::getPositionGorz()
{
	return positionGorz;
}

void EditLine::setName(std::string info)
{
	name = info;
```

```
	std::cout << "Set name: " << info << " for " << name << '\n';
}

std::string EditLine::getName()
{
	return name;
}

void EditLine::setMultiline(bool info)
{
	multiline = info;
	std::cout << "Set name: " << info << " for " << name << '\n';
}

bool EditLine::getMultiline()
{
	return multiline;
}
```

## Файл EditLine.h.cpp

```
#pragma once
#include <iostream>
#include "Text.h"
class EditLine
{
public:
	EditLine();
	~EditLine() = default;


	void setSizeVert(int info);
	int  getSizeVert();
	void setSizeGorz(int info);
	int  getSizeGorz();
	void setPositionVert(int info);
	int  getPositionVert();
	void setPositionGorz(int info);
	int  getPositionGorz();
	void setName(std::string info);
	std::string getName();
	void setMultiline(bool info);
	bool getMultiline();
```

```
	Text text;
private:
	std::string name;
	int sizeVert = 72;
	int sizeGorz = 120;
	int positionVert = 0;
	int positionGorz = 0;

	bool multiline = false;
};
```

## Файл Error.cpp

```
#include "Error.h"
#include <string>

void Error::setError(std::string info, int line)
{
	if (line != 0)
	{
		error = "line " + std::to_string(line) + ": ";
	}
	error += info;

	std::cout << '\n' << error << '\n';
	std::ofstream fout;
	fout.open(pathError);
	fout << error;
}
```

## Файл Error.h

```
#pragma once
#include <iostream>
#include<fstream>

class Error
{
public:
	Error() = default;
	~Error() = default;
	void setError(std::string info,int line);
	std::string getError() { return error; }
```

```
private:
	const std::string pathError{ "Error.txt" };

	std::string error{""};

};
```

## Файл Form.cpp

```
#include "Form.h"

void Form::setSizeVert(int info)
{
	sizeVert = info;
	std::cout << "Set sizeVert: " << info << " for " << name << '\n';
}

int Form::getSizeVert()
{
	return sizeVert;
}

void Form::setSizeGorz(int info)
{
	sizeGorz = info;
	std::cout << "Set sizeGorz: " << info << " for " << name << '\n';
}

int Form::getSizeGorz()
{
	return sizeGorz;

}

void Form::createButton()
{
	Button button;
	buttons.push_back(button);
	std::cout << "Create button in " << name << '\n';
}

void Form::createLabel()
{
```

```
	Label label;
	labels.push_back(label);
	std::cout << "Create label in " << name << '\n';
}

void Form::createNumberBox()
{
	NumberBox numberBox;
	numberBoxes.push_back(numberBox);
	std::cout << "Create numberBox in " << name << '\n';
}

void Form::createCheckBox()
{
	CheckBox checkBox;
	checkBoxes.push_back(checkBox);
	std::cout << "Create checkBox in " << name << '\n';
}

void Form::createRadioButton()
{
	RadioButton radioButton;
	radioButtones.push_back(radioButton);
	std::cout << "Create rdioButton in " << name << '\n';
}

void Form::createEditLine()
{
	EditLine editLine;
	editLines.push_back(editLine);
	std::cout << "Create editLine in " << name << '\n';
}

void Form::createImage()
{
	Image image;
	images.push_back(image);
	std::cout << "Create image in " << name << '\n';
}

void Form::setName(std::string info)
{
	name = info;
```

```
	std::cout << "Set name: " << info << " for " << name << '\n';
}

std::string Form::getName()
{
	return name;
}
```

## Файл Form.h

```
#pragma once
#include <iostream>
#include <vector>

#include "Button.h"
#include "Label.h"
#include "NumberBox.h"
#include "CheckBox.h"
#include "RadioButton.h"
#include "EditLine.h"
#include "Image.h"
class Form
{
public:
	Form() = default;
	~Form() = default;


	void setSizeVert(int info);
	int  getSizeVert();
	void setSizeGorz(int info);
	int  getSizeGorz();
	void setName(std::string info);
	std::string getName();


	void createButton();
	void createLabel();
	void createNumberBox();
	void createCheckBox();
	void createRadioButton();
	void createEditLine();
	void createImage();
```

```
	std::vector<Button> buttons;
	std::vector<Label>  labels;
	std::vector<NumberBox> numberBoxes;
	std::vector<CheckBox> checkBoxes;
	std::vector<RadioButton> radioButtones;
	std::vector<EditLine> editLines;
	std::vector<Image> images;
private:
	int sizeVert = 720;
	int sizeGorz = 1200;
	std::string name = "Form";
};
```

## Файл Label.cpp

```
#include "Label.h"
#include <stdlib.h>
#include <time.h>
#include <string>

Label::Label()
{
	srand(time(NULL));
	name = std::to_string(rand());
}

void Label::setPositionVert(int info)
{
	positionVert = info;
	std::cout << "Set positionVert: " << info << " for " << name << '\n';
}

int Label::getPositionVert()
{
	return positionVert;
}

void Label::setPositionGorz(int info)
{
	positionGorz = info;
	std::cout << "Set positionGorz: " << info << " for " << name << '\n';
}
```

```
int Label::getPositionGorz()
{
	return positionGorz;
}

void Label::setName(std::string info)
{
	name = info;
	std::cout << "Set name: " << info << " for " << name << '\n';
}

std::string Label::getName()
{
	return name;
}
```

## Файл Label.cpp

```
#pragma once
#include <iostream>
#include "Text.h"
class Label
{
public:
    Label();
    ~Label() = default;


    void setPositionVert(int info);
    int  getPositionVert();
    void setPositionGorz(int info);
    int  getPositionGorz();
    void setName(std::string info);
    std::string getName();

    Text text;

private:
    std::string name;
    int positionVert = 0;
    int positionGorz = 0;

};
```

## Файл NumberBox.cpp

```
#include "NumberBox.h"
#include <stdlib.h>
#include <time.h>
```

```
#include <string>

NumberBox::NumberBox()
{
	srand(time(NULL));
	name = std::to_string(rand());
}

void NumberBox::setPositionVert(int info)
{
	positionVert = info;
	std::cout << "Set positionVert: " << info << " for " << name << '\n';
}

int NumberBox::getPositionVert()
{
	return positionVert;
}

void NumberBox::setPositionGorz(int info)
{
	positionGorz = info;
	std::cout << "Set positionGorz: " << info << " for " << name << '\n';
}

int NumberBox::getPositionGorz()
{
	return positionGorz;
}

void NumberBox::setName(std::string info)
{
	name = info;
	std::cout << "Set name: " << info << " for " << name << '\n';
}

std::string NumberBox::getName()
{
	return name;
}

void NumberBox::setMinValue(int info)
{
```

```
	minValue = info;
	std::cout << "Set minValue: " << info << " for " << name << '\n';
}

int NumberBox::setMinValue()
{
	return minValue;
}

void NumberBox::setMaxValue(int info)
{
	maxValue = info;
	std::cout << "Set maxValue: " << info << " for " << name << '\n';
}

int NumberBox::setMaxValue()
{
	return maxValue;
}

void NumberBox::setDefaultValue(int info)
{
	defaultValue = info;
	std::cout << "Set defaultValue: " << info << " for " << name << '\n';
}

int NumberBox::setDefaultValue()
{
	return defaultValue;
}

void NumberBox::setClicable(bool info)
{
	clicable = info;
	std::cout << "Set clicable " << info << "for " << name << '\n';
}

bool NumberBox::getClicable()
{
	return clicable;
}
```

**Файл NumberBox.h**

```
#pragma once
#pragma once
#include <iostream>
#include "Text.h"
class NumberBox
{
public:
	NumberBox();
	~NumberBox() = default;


	void setPositionVert(int info);
	int  getPositionVert();
	void setPositionGorz(int info);
	int  getPositionGorz();
	void setName(std::string info);
	std::string getName();
	void setMinValue(int info);
	int setMinValue();
	void setMaxValue(int info);
	int setMaxValue();
	void setDefaultValue(int info);
	int setDefaultValue();
	void setClicable(bool info);
	bool getClicable();

	Text text;

private:
	std::string name;
	int positionVert = 0;
	int positionGorz = 0;
	int minValue = 0;
	int maxValue = 12;
	int defaultValue = 0;
	bool clicable = false;

};
```

**Файл Parser.cpp**

```
#include "Parcer.h"

#include <iostream>
#include <string>
```

```
namespace NMMessages
{
	const std::string notOpenFile = "can not open file: ";
	const std::string unknCommand = "Unknown command";
	const std::string unknWidget = "Unknown widget or not in scope";
	const std::string createWidgetError = "This widget can't creating in
this scope";
	const std::string outOfScope = "Out of any scopes";
	const std::string noParam = "Must be ':' in this line";
	const std::string noValue = "Must be value in this line";
	const std::string badParam = "This widget havent this param";
	const std::string badValue = "This line has invalid value or symbol
after scope sign";
}

namespace NMCaommands
{
	const char create  = '|';
	const char param   = '/';
	const char comment = '#';
}

namespace NMWidgets
{
	const std::string form   = "Form";
	const std::string button = "Button";
	const std::string label  = "Label";
	const std::string text   = "Text";
	const std::string numberBox = "NumberBox";
	const std::string checkBox = "CheckBox";
	const std::string radioButton = "RadioButton";
	const std::string editLine = "EditLine";
}

bool checkInt(std::string info)
{
	for (size_t i = 0; i < info.size(); ++i)
	{
		if (info[i] != '0' &&
			info[i] != '1' &&
			info[i] != '2' &&
			info[i] != '3' &&
```

```
                info[i] != '4' &&
                info[i] != '5' &&
                info[i] != '6' &&
                info[i] != '7' &&
                info[i] != '8' &&
                info[i] != '9' &&
                info[i] != ' ')
            {
                return false;
            }
        }
        return true;
}

bool checkBool(std::string info)
{
        std::string temp;
        for (size_t i = 0; i < info.size(); ++i)
        {
            if (info[i] != ' ')
            {
                temp += info[i];
            }
        }
        if (temp != "true" && temp != "false")
        {
            return false;
        }
        return true;
}

bool getBool(std::string info)
{
        std::string temp;
        for (size_t i = 0; i < info.size(); ++i)
        {
            if (info[i] != ' ')
            {
                temp += info[i];
            }
        }
        if (temp == "true")
        {
```

```
			return true;
		}
		if (temp == "false")
		{
			return false;
		}
	}

	bool checkTitle(std::string info)
	{
		if (info != "Calibri" && info != "Arial"
			&& info != "Times New Roman" && info != "Impact"
			&& info != "Corbel")
		{
			return false;
		}
		return true;
	}

	Parcer::Parcer()
	{
		fin.open(pathFile);
	}

	Parcer::~Parcer()
	{
		fin.close();
	}

	bool Parcer::checkFile()
	{
		if (fin.is_open())
		{
			return true;
		}
		else
		{
			error.setError(NMMessages::notOpenFile + pathFile, line);
			return false;
		}
	}

	bool Parcer::parcing()
```

```
{
	scope.push(NMWidgets::form);

	while (!fin.eof())
	{
		if (scope.size() == 0)
		{
			error.setError(NMMessages::outOfScope, line);
			return false;
		}
		++line;
		std::getline(fin, parcingString);

		if (parcingString.size() > 0)
		{//////////////////////////////////////////
			if (parcingString[0] == NMCaommands::create)
			{
				if (getWidget())
				{
					if (!createWidget())
					{

	error.setError(NMMessages::createWidgetError, line);
						return false;
					}
				}
				else
				{
					error.setError(NMMessages::unknWidget, line);
					return false;
				}
			}//////////////////////////////////////////
			else if (parcingString[0] == NMCaommands::param)
			{
				if (getParam())
				{
					if (getValue())
					{
						if (checkParam())
						{
							if (!paramWidget())
							{
```

```
		error.setError(NMMessages::badValue, line);
                                                return false;
                                          }
                                    }
                                    else
                                    {
                                          error.setError(NMMessages::badParam,
line);
                                          return false;
                                    }
                              }
                              else
                              {
                                    error.setError(NMMessages::noValue, line);
                                    return false;
                              }
                        }
                        else
                        {
                              error.setError(NMMessages::noParam, line);
                              return false;
                        }
                  }//////////////////////////////////////////////////
                  else if (parcingString[0] == NMCaommands::comment)
                  {

                  }
                  else
                  {//////////////////////////////////////////////////
                        error.setError(NMMessages::unknCommand, line);
                        return false;
                  }
            }
      }
      std::cout << '\n' << "Parcing done!" << '\n';
      return true;
}

bool Parcer::createWidget()
{
      if (scope.top() == NMWidgets::form)
      {
```

```
        if(activeString == NMWidgets::text)
        {
              return false;
        }
        if (activeString == NMWidgets::button)
        {
              scope.push(NMWidgets::button);
              form.createButton();
              //std::cout << '\n' << "Create button" << '\n';
        }
        if (activeString == NMWidgets::label)
        {
              scope.push(NMWidgets::label);
              form.createLabel();
              //std::cout << '\n' << "Create label" << '\n';
        }
        if (activeString == NMWidgets::numberBox)
        {
              scope.push(NMWidgets::numberBox);
              form.createNumberBox();
              //std::cout << '\n' << "Create numberBox" << '\n';
        }
        if (activeString == NMWidgets::checkBox)
        {
              scope.push(NMWidgets::checkBox);
              form.createCheckBox();
              //std::cout << '\n' << "Create checkBox" << '\n';
        }
        if (activeString == NMWidgets::radioButton)
        {
              scope.push(NMWidgets::radioButton);
              form.createRadioButton();
              //std::cout << '\n' << "Create radioButton" << '\n';
        }
        if (activeString == NMWidgets::editLine)
        {
              scope.push(NMWidgets::editLine);
              form.createEditLine();
              //std::cout << '\n' << "Create editLine" << '\n';
        }
  }
  else
  {
```

```
            if (activeString != NMWidgets::text)
            {
                  return false;
            }
            scope.push(NMWidgets::text);
            std::cout << '\n' << "Create Text" << '\n';
      }

      return true;
}



bool Parcer::paramWidget()
{
      //valueString
      //parcingString
      //activeString
      if (scope.top() == NMWidgets::text)
      {
            scope.pop();

            if (scope.top() == NMWidgets::button)
            {
                  if (activeString == "text")
                  {
                        form.buttons[form.buttons.size() -
1].text.setText(valueString);
                  }
                  if (activeString == "positionVert")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.buttons[form.buttons.size() -
1].text.setPositionVert(std::stoi(valueString));
                        }
                  }
                  if (activeString == "positionGorz")
                  {
```

```
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.buttons[form.buttons.size() -
1].text.setPositionGorz(std::stoi(valueString));
                        }
                  }
                  if (activeString == "height")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.buttons[form.buttons.size() -
1].text.setHeight(std::stoi(valueString));
                        }
                  }
                  if (activeString == "italic")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.buttons[form.buttons.size() -
1].text.setItalic(true);
                              }
                              else
                              {
                                    form.buttons[form.buttons.size() -
1].text.setItalic(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "underline")
```

```
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.buttons[form.buttons.size() -
1].text.setUnderline(true);
                              }
                              else
                              {
                                    form.buttons[form.buttons.size() -
1].text.setUnderline(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "bold")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.buttons[form.buttons.size() -
1].text.setBold(true);
                              }
                              else
                              {
                                    form.buttons[form.buttons.size() -
1].text.setBold(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "title")
                  {
                        if(checkTitle(valueString))
                        {
```

```
                              form.buttons[form.buttons.size() -
1].text.setTitle(valueString);
                        }
                        else
                        {
                              return false;
                        }
                  }

            }

            if (scope.top() == NMWidgets::label)
            {
                  if (activeString == "text")
                  {
                        form.labels[form.labels.size() -
1].text.setText(valueString);
                  }
                  if (activeString == "positionVert")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.labels[form.labels.size() -
1].text.setPositionVert(std::stoi(valueString));
                        }
                  }
                  if (activeString == "positionGorz")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.labels[form.labels.size() -
1].text.setPositionGorz(std::stoi(valueString));
                        }
                  }
                  if (activeString == "height")
```

```
                {
                        if (!checkInt(valueString))
                        {
                                return false;
                        }
                        else
                        {
                                form.labels[form.labels.size() -
1].text.setHeight(std::stoi(valueString));
                        }
                }
                if (activeString == "italic")
                {
                        if (checkBool(valueString))
                        {
                                if (getBool(valueString))
                                {
                                        form.labels[form.labels.size() -
1].text.setItalic(true);
                                }
                                else
                                {
                                        form.labels[form.labels.size() -
1].text.setItalic(false);
                                }
                        }
                        else
                        {
                                return false;
                        }
                }
                if (activeString == "underline")
                {
                        if (checkBool(valueString))
                        {
                                if (getBool(valueString))
                                {
                                        form.labels[form.labels.size() -
1].text.setUnderline(true);
                                }
                                else
                                {
```

```
                                    form.labels[form.labels.size() -
1].text.setUnderline(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "bold")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.labels[form.labels.size() -
1].text.setBold(true);
                              }
                              else
                              {
                                    form.labels[form.labels.size() -
1].text.setBold(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "title")
                  {
                        if (checkTitle(valueString))
                        {
                              form.labels[form.labels.size() -
1].text.setTitle(valueString);
                        }
                        else
                        {
                              return false;
                        }
                  }
            }
```

```
            if (scope.top() == NMWidgets::numberBox)
            {
                  if (activeString == "text")
                  {
                        form.numberBoxes[form.numberBoxes.size() -
1].text.setText(valueString);
                  }
                  if (activeString == "positionVert")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.numberBoxes[form.numberBoxes.size() -
1].text.setPositionVert(std::stoi(valueString));
                        }
                  }
                  if (activeString == "positionGorz")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.numberBoxes[form.numberBoxes.size() -
1].text.setPositionGorz(std::stoi(valueString));
                        }
                  }
                  if (activeString == "height")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.numberBoxes[form.numberBoxes.size() -
1].text.setHeight(std::stoi(valueString));
                        }
```

```
                  }
                  if (activeString == "italic")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.numberBoxes[form.numberBoxes.size() -
1].text.setItalic(true);
                              }
                              else
                              {
                                    form.numberBoxes[form.numberBoxes.size() -
1].text.setItalic(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "underline")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.numberBoxes[form.numberBoxes.size() -
1].text.setUnderline(true);
                              }
                              else
                              {
                                    form.numberBoxes[form.numberBoxes.size() -
1].text.setUnderline(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "bold")
                  {
```

```
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.numberBoxes[form.numberBoxes.size() -
1].text.setBold(true);
                              }
                              else
                              {
                                    form.numberBoxes[form.numberBoxes.size() -
1].text.setBold(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "title")
                  {
                        if (checkTitle(valueString))
                        {
                              form.numberBoxes[form.numberBoxes.size() -
1].text.setTitle(valueString);
                        }
                        else
                        {
                              return false;
                        }
                  }
            }




            if (scope.top() == NMWidgets::checkBox)
            {
                  if (activeString == "text")
                  {
                        form.checkBoxes[form.checkBoxes.size() -
1].text.setText(valueString);
                  }
                  if (activeString == "positionVert")
```

```
                {
                        if (!checkInt(valueString))
                        {
                                return false;
                        }
                        else
                        {
                                form.checkBoxes[form.checkBoxes.size() -
1].text.setPositionVert(std::stoi(valueString));
                        }
                }
                if (activeString == "positionGorz")
                {
                        if (!checkInt(valueString))
                        {
                                return false;
                        }
                        else
                        {
                                form.checkBoxes[form.checkBoxes.size() -
1].text.setPositionGorz(std::stoi(valueString));
                        }
                }
                if (activeString == "height")
                {
                        if (!checkInt(valueString))
                        {
                                return false;
                        }
                        else
                        {
                                form.checkBoxes[form.checkBoxes.size() -
1].text.setHeight(std::stoi(valueString));
                        }
                }
                if (activeString == "italic")
                {
                        if (checkBool(valueString))
                        {
                                if (getBool(valueString))
                                {
                                        form.checkBoxes[form.checkBoxes.size() -
1].text.setItalic(true);
```

```
                        }
                        else
                        {
                              form.checkBoxes[form.checkBoxes.size() -
1].text.setItalic(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "underline")
            {
                  if (checkBool(valueString))
                  {
                        if (getBool(valueString))
                        {
                              form.checkBoxes[form.checkBoxes.size() -
1].text.setUnderline(true);
                        }
                        else
                        {
                              form.checkBoxes[form.checkBoxes.size() -
1].text.setUnderline(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "bold")
            {
                  if (checkBool(valueString))
                  {
                        if (getBool(valueString))
                        {
                              form.checkBoxes[form.checkBoxes.size() -
1].text.setBold(true);
                        }
                        else
                        {
```

```
                                    form.checkBoxes[form.checkBoxes.size() -
1].text.setBold(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "title")
                  {
                        if (checkTitle(valueString))
                        {
                              form.checkBoxes[form.checkBoxes.size() -
1].text.setTitle(valueString);
                        }
                        else
                        {
                              return false;
                        }
                  }
            }




            if (scope.top() == NMWidgets::radioButton)
            {
                  if (activeString == "text")
                  {
                        form.radioButtones[form.radioButtones.size() -
1].text.setText(valueString);
                  }
                  if (activeString == "positionVert")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.radioButtones[form.radioButtones.size() -
1].text.setPositionVert(std::stoi(valueString));
```

```
                        }
                  }
                  if (activeString == "positionGorz")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.radioButtones[form.radioButtones.size() -
1].text.setPositionGorz(std::stoi(valueString));
                        }
                  }
                  if (activeString == "height")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.radioButtones[form.radioButtones.size() -
1].text.setHeight(std::stoi(valueString));
                        }
                  }
                  if (activeString == "italic")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {

      form.radioButtones[form.radioButtones.size() - 1].text.setItalic(true);
                              }
                              else
                              {

      form.radioButtones[form.radioButtones.size() - 1].text.setItalic(false);
                              }
                        }
                        else
                        {
```

```
                        return false;
                    }
                }
                if (activeString == "underline")
                {
                    if (checkBool(valueString))
                    {
                        if (getBool(valueString))
                        {

    form.radioButtones[form.radioButtones.size() -
1].text.setUnderline(true);
                        }
                        else
                        {

    form.radioButtones[form.radioButtones.size() -
1].text.setUnderline(false);
                        }
                    }
                    else
                    {
                        return false;
                    }
                }
                if (activeString == "bold")
                {
                    if (checkBool(valueString))
                    {
                        if (getBool(valueString))
                        {

    form.radioButtones[form.radioButtones.size() - 1].text.setBold(true);
                        }
                        else
                        {

    form.radioButtones[form.radioButtones.size() - 1].text.setBold(false);
                        }
                    }
                    else
                    {
                        return false;
```

```
                        }
                  }
                  if (activeString == "title")
                  {
                        if (checkTitle(valueString))
                        {
                              form.radioButtones[form.radioButtones.size() -
1].text.setTitle(valueString);
                        }
                        else
                        {
                              return false;
                        }
                  }
            }



            if (scope.top() == NMWidgets::editLine)
            {
                  if (activeString == "text")
                  {
                        form.editLines[form.editLines.size() -
1].text.setText(valueString);
                  }
                  if (activeString == "positionVert")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.editLines[form.editLines.size() -
1].text.setPositionVert(std::stoi(valueString));
                        }
                  }
                  if (activeString == "positionGorz")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
```

```
                        else
                        {
                              form.editLines[form.editLines.size() -
1].text.setPositionGorz(std::stoi(valueString));
                        }
                  }
                  if (activeString == "height")
                  {
                        if (!checkInt(valueString))
                        {
                              return false;
                        }
                        else
                        {
                              form.editLines[form.editLines.size() -
1].text.setHeight(std::stoi(valueString));
                        }
                  }
                  if (activeString == "italic")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
                              {
                                    form.editLines[form.editLines.size() -
1].text.setItalic(true);
                              }
                              else
                              {
                                    form.editLines[form.editLines.size() -
1].text.setItalic(false);
                              }
                        }
                        else
                        {
                              return false;
                        }
                  }
                  if (activeString == "underline")
                  {
                        if (checkBool(valueString))
                        {
                              if (getBool(valueString))
```

```
                        {
                              form.editLines[form.editLines.size() -
1].text.setUnderline(true);
                        }
                        else
                        {
                              form.editLines[form.editLines.size() -
1].text.setUnderline(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "bold")
            {
                  if (checkBool(valueString))
                  {
                        if (getBool(valueString))
                        {
                              form.editLines[form.editLines.size() -
1].text.setBold(true);
                        }
                        else
                        {
                              form.editLines[form.editLines.size() -
1].text.setBold(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "title")
            {
                  if (checkTitle(valueString))
                  {
                        form.editLines[form.editLines.size() -
1].text.setTitle(valueString);
                  }
                  else
```

```
                        {
                              return false;
                        }
                  }

            }
            scope.push(NMWidgets::text);
      }




      else if (scope.top() == NMWidgets::button)
      {
            if (activeString == "sizeVert")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.buttons[form.buttons.size() -
1].setSizeVert(std::stoi(valueString));
                  }
            }
            if (activeString == "sizeGorz")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.buttons[form.buttons.size() -
1].setSizeGorz(std::stoi(valueString));
                  }
            }
            if (activeString == "positionVert")
            {
                  if (!checkInt(valueString))
                  {
```

```
                        return false;
                  }
                  else
                  {
                        form.buttons[form.buttons.size() -
1].setPositionVert(std::stoi(valueString));
                  }
            }
            if (activeString == "positionGorz")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.buttons[form.buttons.size() -
1].setPositionGorz(std::stoi(valueString));
                  }
            }
            if (activeString == "clicable")
            {
                  if (checkBool(valueString))
                  {
                        if (getBool(valueString))
                        {
                              form.buttons[form.buttons.size() -
1].setClicable(true);
                        }
                        else
                        {
                              form.buttons[form.buttons.size() -
1].setClicable(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "doubleClicable")
            {
                  if (checkBool(valueString))
```

```
                  {
                        if (getBool(valueString))
                        {
                              form.buttons[form.buttons.size() -
1].setDoubleClicable(true);
                        }
                        else
                        {
                              form.buttons[form.buttons.size() -
1].setDoubleClicable(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "name")
            {
                  form.buttons[form.buttons.size() - 1].setName(valueString);
            }
      }




      else if (scope.top() == NMWidgets::label)
      {
            if (activeString == "positionVert")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.labels[form.labels.size() -
1].setPositionVert(std::stoi(valueString));
                  }
            }
            if (activeString == "positionGorz")
```

```
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.labels[form.labels.size() -
1].setPositionGorz(std::stoi(valueString));
                  }
            }
            if (activeString == "name")
            {
                  form.labels[form.labels.size() - 1].setName(valueString);
            }
      }




      else if (scope.top() == NMWidgets::numberBox)
      {
            if (activeString == "positionVert")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.numberBoxes[form.numberBoxes.size() -
1].setPositionVert(std::stoi(valueString));
                  }
            }
            if (activeString == "positionGorz")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
```

```
                        form.numberBoxes[form.numberBoxes.size() -
1].setPositionGorz(std::stoi(valueString));
                  }
            }
            if (activeString == "name")
            {
                  form.numberBoxes[form.numberBoxes.size() -
1].setName(valueString);
            }
            if (activeString == "minValue")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.numberBoxes[form.numberBoxes.size() -
1].setMinValue(std::stoi(valueString));
                  }
            }
            if (activeString == "maxValue")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.numberBoxes[form.numberBoxes.size() -
1].setMaxValue(std::stoi(valueString));
                  }
            }
            if (activeString == "defaultValue")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.numberBoxes[form.numberBoxes.size() -
1].setDefaultValue(std::stoi(valueString));
```

```
                  }
            }
            if (activeString == "clicable")
            {
                  if (checkBool(valueString))
                  {
                        if (getBool(valueString))
                        {
                              form.numberBoxes[form.numberBoxes.size() -
1].setClicable(true);
                        }
                        else
                        {
                              form.numberBoxes[form.numberBoxes.size() -
1].setClicable(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
      }



      else if (scope.top() == NMWidgets::checkBox)
      {
            if (activeString == "positionVert")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.checkBoxes[form.checkBoxes.size() -
1].setPositionVert(std::stoi(valueString));
                  }
            }
            if (activeString == "positionGorz")
            {
                  if (!checkInt(valueString))
```

```
                {
                        return false;
                }
                else
                {
                        form.checkBoxes[form.checkBoxes.size() -
1].setPositionGorz(std::stoi(valueString));
                }
        }
        if (activeString == "name")
        {
                form.checkBoxes[form.checkBoxes.size() -
1].setName(valueString);
        }
        if (activeString == "defaultValue")
        {
                if (checkBool(valueString))
                {
                        if (getBool(valueString))
                        {
                                form.checkBoxes[form.checkBoxes.size() -
1].setDefaultValue(true);
                        }
                        else
                        {
                                form.checkBoxes[form.checkBoxes.size() -
1].setDefaultValue(false);
                        }
                }
        }
        if (activeString == "clicable")
        {
                if (checkBool(valueString))
                {
                        if (getBool(valueString))
                        {
                                form.checkBoxes[form.checkBoxes.size() -
1].setClicable(true);
                        }
                        else
                        {
                                form.checkBoxes[form.checkBoxes.size() -
1].setClicable(false);
```

```
                        }
                }
                else
                {
                        return false;
                }
        }
    }


    else if (scope.top() == NMWidgets::radioButton)
    {
        if (activeString == "positionVert")
        {
                if (!checkInt(valueString))
                {
                        return false;
                }
                else
                {
                        form.radioButtones[form.radioButtones.size() -
1].setPositionVert(std::stoi(valueString));
                }
        }
        if (activeString == "positionGorz")
        {
                if (!checkInt(valueString))
                {
                        return false;
                }
                else
                {
                        form.radioButtones[form.radioButtones.size() -
1].setPositionGorz(std::stoi(valueString));
                }
        }
        if (activeString == "name")
        {
                form.radioButtones[form.radioButtones.size() -
1].setName(valueString);
        }
        if (activeString == "groupName")
        {
```

```
                  form.radioButtones[form.radioButtones.size() -
1].setGroupName(valueString);
            }
            if (activeString == "defaultValue")
            {
                  if (checkBool(valueString))
                  {
                        if (getBool(valueString))
                        {
                              form.radioButtones[form.radioButtones.size() -
1].setDefaultValue(true);
                        }
                        else
                        {
                              form.radioButtones[form.radioButtones.size() -
1].setDefaultValue(false);
                        }
                  }
            }
      }




      else if (scope.top() == NMWidgets::button)
      {
            if (activeString == "sizeVert")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.buttons[form.buttons.size() -
1].setSizeVert(std::stoi(valueString));
                  }
            }
            if (activeString == "sizeGorz")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
```

```
                }
                else
                {
                        form.buttons[form.buttons.size() -
1].setSizeGorz(std::stoi(valueString));
                }
            }
            if (activeString == "positionVert")
            {
                if (!checkInt(valueString))
                {
                        return false;
                }
                else
                {
                        form.buttons[form.buttons.size() -
1].setPositionVert(std::stoi(valueString));
                }
            }
            if (activeString == "positionGorz")
            {
                if (!checkInt(valueString))
                {
                        return false;
                }
                else
                {
                        form.buttons[form.buttons.size() -
1].setPositionGorz(std::stoi(valueString));
                }
            }
            if (activeString == "clicable")
            {
                if (checkBool(valueString))
                {
                        if (getBool(valueString))
                        {
                                form.buttons[form.buttons.size() -
1].setClicable(true);
                        }
                        else
                        {
```

```
                              form.buttons[form.buttons.size() -
1].setClicable(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "doubleClicable")
            {
                  if (checkBool(valueString))
                  {
                        if (getBool(valueString))
                        {
                              form.buttons[form.buttons.size() -
1].setDoubleClicable(true);
                        }
                        else
                        {
                              form.buttons[form.buttons.size() -
1].setDoubleClicable(false);
                        }
                  }
                  else
                  {
                        return false;
                  }
            }
            if (activeString == "name")
            {
                  form.buttons[form.buttons.size() - 1].setName(valueString);
            }
      }



      else if (scope.top() == NMWidgets::form)
      {
            if (activeString == "sizeVert")
            {
                  if (!checkInt(valueString))
                  {
```

```
                        return false;
                  }
                  else
                  {
                        form.setSizeVert(std::stoi(valueString));
                  }
            }
            if (activeString == "sizeGorz")
            {
                  if (!checkInt(valueString))
                  {
                        return false;
                  }
                  else
                  {
                        form.setSizeGorz(std::stoi(valueString));
                  }
            }
            if (activeString == "name")
            {
                  form.setName(valueString);
            }
      }


      bool closeScope = false;
      for (size_t i = 0; i < parcingString.size(); ++i)
      {
            if (closeScope && (parcingString[i] != ']' || parcingString[i] !=
' '))
            {
                  return false;
            }
            if (parcingString[i] == ']')
            {
                  closeScope = true;
            }

      }
      if (closeScope)
      {
            std::cout << "\nClose scope " << scope.top() << '\n';
            scope.pop();
```

```
	}

	return true;
}

bool Parcer::getWidget()
{
	activeString.clear();
	bool inScope = false;
	for (size_t i = 1; i < parcingString.size(); ++i)
	{
		if (parcingString[i] != '[' && parcingString[i] != ' '
			&& parcingString[i] != '|' && parcingString[i] != '/'
			&& parcingString[i] != '>' && parcingString[i] != '#')
		{
			activeString += parcingString[i];
		}

		if (parcingString[i] == '[')
		{
			inScope = true;
			break;
			//std::cout << '\n' << activeString << '\n';
		}
	}
	if (inScope != true || (activeString != NMWidgets::button &&
		activeString != NMWidgets::label && activeString !=
NMWidgets::text)
		&& activeString != NMWidgets::numberBox && activeString !=
NMWidgets::checkBox
		&& activeString != NMWidgets::radioButton && activeString !=
NMWidgets::editLine)
	{
		return false;
	}

	return true;
}

bool Parcer::getParam()
{
	activeString.clear();
	bool value = false;
```

```
	bool closeScope = false;
	for (size_t i = 1; i < parcingString.size(); ++i)
	{
		if (parcingString[i] == ':')
		{
			value = true;
			break;
			//std::cout << '\n' << activeString << '\n';
		}
		if (parcingString[i] == ']')
		{
			closeScope = true;
		}
		if (parcingString[i] != '[' && parcingString[i] != ' '
			&& parcingString[i] != '|' && parcingString[i] != '/'
			&& parcingString[i] != '>' && parcingString[i] != '#')
		{
			activeString += parcingString[i];
		}
	}
	if (value != true)
	{
		if (closeScope)
		{
			onlyScope = true;
		}
		else
		{
			return false;
		}

	}

	return true;
}

bool Parcer::getValue()
{
	valueString.clear();
	bool value = false;
	bool firstspace = false;
	for (size_t i = 1; i < parcingString.size(); ++i)
	{
```

```
            if (value)
            {
                  if (parcingString[i] != '[' && parcingString[i] != ']'
                        && parcingString[i] != '|' && parcingString[i] != '/'
                        && parcingString[i] != '>' && parcingString[i] != '#')
                  {
                        if (parcingString[i] != ' ' || firstspace)
                        {
                              valueString += parcingString[i];
                        }
                        if (parcingString[i] == ' ')
                        {
                              firstspace = true;
                        }
                  }

            }
            if (parcingString[i] == ':')
            {
                  value = true;
            }
      }
      if (valueString.size() == 0)
      {
            if (onlyScope)
            {

            }
            else
            {
                  return false;
            }

      }

      return true;
}

bool Parcer::checkParam()
{
      if (onlyScope)
      {
```

```
	}
	else
	{
		if (scope.top() == NMWidgets::text)
		{
			if (activeString != "text"
				&& activeString != "positionVert" && activeString !=
"positionGorz"
				&& activeString != "height" && activeString !=
"italic"
				&& activeString != "underline" && activeString !=
"bold"
				&& activeString != "title")
			{
				return false;
			}
		}
		else if (scope.top() == NMWidgets::button)
		{
			if (activeString != "sizeVert" && activeString != "sizeGorz"
				&& activeString != "positionVert" && activeString !=
"positionGorz"
				&& activeString != "name" && activeString !=
"clicable" && activeString != "doubleClicable")
			{
				return false;
			}
		}
		else if (scope.top() == NMWidgets::label)
		{
			if (activeString != "positionVert" && activeString !=
"positionGorz"
				&& activeString != "name")
			{
				return false;
			}
		}
		else if (scope.top() == NMWidgets::numberBox)
		{
			if (activeString != "positionVert" && activeString !=
"positionGorz"
```

```
                        && activeString != "name" && activeString !=
"minValue" && activeString != "maxValue"
                        && activeString != "defaultValue" && activeString !=
"clicable")
                  {
                        return false;
                  }
            }
            else if (scope.top() == NMWidgets::checkBox)
            {
                  if (activeString != "positionVert" && activeString !=
"positionGorz"
                        && activeString != "name" && activeString !=
"defaultValue"
                        && activeString != "clicable")
                  {
                        return false;
                  }
            }
            else if (scope.top() == NMWidgets::radioButton)
            {
                  if (activeString != "positionVert" && activeString !=
"positionGorz"
                        && activeString != "name" && activeString !=
"defaultValue"
                        && activeString != "groupName")
                  {
                        return false;
                  }
            }
            else if (scope.top() == NMWidgets::editLine)
            {
                  if (activeString != "sizeVert" && activeString != "sizeGorz"
                        && activeString != "positionVert" && activeString !=
"positionGorz"
                        && activeString != "name" && activeString !=
"multiline")
                  {
                        return false;
                  }
            }
            else if (scope.top() == NMWidgets::form)
            {
```

```
                  if (activeString != "sizeVert" && activeString != "sizeGorz"
&& activeString != "name")
                  {
                        return false;
                  }
            }
            return true;
      }

}
```

## Файл Parser.h

```
#pragma once
#include <fstream>
#include <stack>

#include "Error.h"
#include "Form.h"


class Parcer
{
public:
      Parcer();
      ~Parcer();
      bool checkFile();
      bool parcing();

      std::string getError() { return error.getError(); }
private:
      const std::string pathFile{ "CppUi.txt" };

      std::ifstream fin;

      int line{ 0 };
      std::string activeString{""};
      std::string parcingString{""};
      std::string valueString{""};
      std::stack<std::string> scope;

      bool onlyScope = false;


      Error error;
```

```
	Form form;


	bool createWidget();
	bool paramWidget();

	bool getWidget();
	bool getParam();
	bool getValue();
	bool checkParam();
};
```

**Файл RadioButton.cpp**

```
#include "RadioButton.h"
#include <stdlib.h>
#include <time.h>
#include <string>

RadioButton::RadioButton()
{
	srand(time(NULL));
	name = std::to_string(rand());
}

void RadioButton::setPositionVert(int info)
{
	positionVert = info;
	std::cout << "Set positionVert: " << info << " for " << name << '\n';
}

int RadioButton::getPositionVert()
{
	return positionVert;
}

void RadioButton::setPositionGorz(int info)
{
	positionGorz = info;
	std::cout << "Set positionGorz: " << info << " for " << name << '\n';
}

int RadioButton::getPositionGorz()
{
	return positionGorz;
```

```
}

void RadioButton::setName(std::string info)
{
	name = info;
	std::cout << "Set name: " << info << " for " << name << '\n';
}

std::string RadioButton::getName()
{
	return name;
}

void RadioButton::setDefaultValue(bool info)
{
	defaultValue = info;
	std::cout << "Set defaultValue: " << info << " for " << name << '\n';
}

bool RadioButton::setDefaultValue()
{
	return defaultValue;
}

void RadioButton::setGroupName(std::string info)
{
	groupName = info;
	std::cout << "Set groupName: " << info << " for " << name << '\n';
}

std::string RadioButton::getGroupName()
{
	return groupName;
}
```

## Файл RadioButton.cpp

```
#pragma once
#include <iostream>
#include "Text.h"
class RadioButton
{
public:
	RadioButton();
```

```
	~RadioButton() = default;


	void setPositionVert(int info);
	int  getPositionVert();
	void setPositionGorz(int info);
	int  getPositionGorz();
	void setName(std::string info);
	std::string getName();
	void setDefaultValue(bool info);
	bool setDefaultValue();
	void setGroupName(std::string info);
	std::string getGroupName();

	Text text;
private:
	std::string name;
	std::string groupName = "RB";
	int positionVert = 0;
	int positionGorz = 0;
	bool defaultValue = false;
};
```

## Файл Text.cpp

```
#include "Text.h"

void Text::setPositionVert(int info)
{
	positionVert = info;
	std::cout << "Set positionVert: " << info << " for " << text << '\n';

}

int Text::getPositionVert()
{
	return positionVert;
}

void Text::setPositionGorz(int info)
{
	positionGorz = info;
	std::cout << "Set positionGorz: " << info << " for " << text << '\n';
}
```

```
int Text::getPositionGorz()
{
	return positionGorz;
}

void Text::setHeight(int info)
{
	height = info;
	std::cout << "Set height: " << info << " for " << text << '\n';
}

int Text::getHeight()
{
	return height;
}

void Text::setItalic(bool info)
{
	italic = info;
	std::cout << "Set italic " << info << "for " << text << '\n';
}

bool Text::getItalic()
{
	return italic;
}

void Text::setUnderline(bool info)
{
	underline = info;
	std::cout << "Set underline " << info << "for " << text << '\n';
}

bool Text::getUnderline()
{
	return underline;
}

void Text::setBold(bool info)
{
	bold = info;
	std::cout << "Set bold " << info << "for " << text << '\n';
```

```
}

bool Text::getBold()
{
	return bold;
}

std::string Text::getText()
{
	return text;
}

void Text::setText(std::string info)
{
	text = info;
	std::cout << "Set text: " << info << " for " << text << '\n';
}

std::string Text::getTitle()
{
	return title;
}

void Text::setTitle(std::string info)
{
	title = info;
	std::cout << "Set title: " << info << " for " << text << '\n';
}
```

**Файл Text.h**

```
#pragma once
#include <iostream>
class Text
{
public:
	Text() = default;
	~Text() = default;


	void setPositionVert(int info);
	int  getPositionVert();
	void setPositionGorz(int info);
	int  getPositionGorz();
```

```
        void setHeight(int info);
        int getHeight();
        void setItalic(bool info);
        bool getItalic();
        void setUnderline(bool info);
        bool getUnderline();
        void setBold(bool info);
        bool getBold();

        std::string getText();
        void setText(std::string info);

        std::string getTitle();
        void setTitle(std::string info);

    private:
        int positionVert = 0;
        int positionGorz = 0;
        std::string text = "Text";


        int height = 12;
        bool italic = false;
        bool underline = false;
        bool bold = false;
        std::string title = "Arial";


    };
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

# ПОДІЙНО-ОРІЄНТОВАНА БІБЛІОТЕКА ДЛЯ СТВОРЕННЯ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ НА ОСНОВІ ВЛАСНОЇ МОВИ РОЗМІТКИ (комплексна тема)

## Програма та методика тестування

КПІ.ІП-9209.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавці:
_____ Володимир ГРИЦЮК
_____ Назарій ПРОШИН

Київ – 2023

**ЗМІСТ**

# 1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є подійно-орієнтована бібліотека для створення користувацьких інтерфейсів на основі власної мови розмітки. Програмне забезпечення було розроблено для операційної системи Windows на мові програмування C++ з використанням інструментів CMake в середовищі Visual Studio Code.

# 2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка правильності роботи алгоритмів парсингу мови розмітки;

– перевірка правильності створення та відображення вікна та віджетів на основі параметрів, що можуть бути задані у файлі розмітки;

– перевірка правильності опрацювання подій взаємодії з віджетами;

– перевірка сумісності бібліотеки з операційними системами Windows 7/8/10.

– знаходження проблем, помилок і недоліків з метою їх усунення.

# 3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

# 4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення, так і в зручності користування. Тестування проводилось як і окремо для компонентів, що відповідають за парсинг, та компонентів, що відповідають за відображення віджетів та опрацювання подій взаємодії з ними, так і для бібліотеки загалом. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– тестування алгоритмів парсингу файлів файлів розмітки, що містять різні віджети, параметри яких коректно задані;

– тестування алгоритмів парсингу файлів розмітки на повернення повилок при некоректно заданих параметрах віджетів, порушенні синтаксису чи граматики мови розмітки;

– тестування створення та відображення віджетів з різними параметрами;

– тестування опрацювання подій взаємодії з віджетами;

– тестування бібліотеки на виведення помилок, коли файл розмітки некоректний;

– тестування бібліотеки на створення графічного інтерфейсу, коли файл розмітки та логіка взаємодії з віджетами визначені коректно;

– тестування на моніторах з різною роздільною здатністю екрану;

– тестування на персональних комп’ютерах з різною версією операційної системи Windows (Windows 7/8/10);

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

# ПОДІЙНО-ОРІЄНТОВАНА БІБЛІОТЕКА ДЛЯ СТВОРЕННЯ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ НА ОСНОВІ ВЛАСНОЇ МОВИ РОЗМІТКИ (комплексна тема)

**Керівництво користувача**

КПІ.ІХ-9209.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавці:
_____ Володимир ГРИЦЮК
_____ Назарій ПРОШИН

Київ – 2023

**ЗМІСТ**

# 1 ПРИЗНАЧЕННЯ ПРОГРАМИ

“CppUI” – бібліотека для створення користувацьких інтерфейсів на основі власної мови розмітки. Бібліотека дозволяє створити графічний інтерфейс користувача для застосунків, що працюють в операційній системі Windows, та написані на мові програмування C++. Використовуючи спеціальну мову розмітки, користувач може у файлі “CppUI.txt” описати параметри вікна та віджетів. Використовую функціонал бібліотеки, користувач можн створити графічний інтерфейс для свого застосунку та налаштувати логіку обробки взаємодії з різними віджетами під свої потреби. Вихідний код бібліотеки міститься в git репозиторії, що включає в себе файли коду всіх компонентів, а також приклади використання бібліотеки. Користувач може завантажити бібліотеку та використовувати її у своїх проектах.

# 2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

## 2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність персонального комп’ютера зі встановленою операційною системою Windows;

– процесор Intel Core Pentium або вище (рекомендовано Intel Core i3 або вище);

– об’єм ОЗП 2 ГБ або більше (рекомендовано 4 Гб або більше);

– C++ компілятор, що підтримує стандарт C++17.

## 2.2 Завантаження застосунку

Вихідний код бібліотеки розміщений на платформі GitHub. Першим етапом інтеграції бібліотеки до будь-якого проекту буде клонування репозиторію з бібліотекою до папки проекту. Це можна зробити за допомогою команди “git clone https://github.com/vovahrytsiuk/Diploma.git ./CppUI”. В результаті в папці проекту ми отримаємо повний вихідний код бібліотеки. Наступним кроком користувач повинен налаштувати підключення бібліотеки до свого проекту. Для прикладу таке налаштування за допомогою CMake. Користувач додає статичну бібліотеку до проекту за допомогою команди add_subdirectory(CppUI). Дана команда допоможе налаштувати шлях до файлів бібліотеки в нашому застосунку. Надалі потрібно залінкувати бібліотеку за допомогою команди target_link_libraries. Обов’язковим кроком також є встановлення стандарту C++17. Після виконання цих кроків користувач зможе використовувати функціонал бібліотеки при створенні графічного інтерфейсу для свого застосунку.

## 2.3 Перевірка коректної роботи

Після завершення підключення бібліотеки, користувач може перевірити правильність підключення та роботоздатність, створивши за допомогою бібліотеки вікно, параметри якого потрібно задати у файлі розмітки. Для цього користувачу потрібно створити файл “CppUI.txt” в кореневій папці проекту та описати в ньому розмір та заголовок вікна. Приклад такого файлу розмітки наведено на рисунку 2.1.

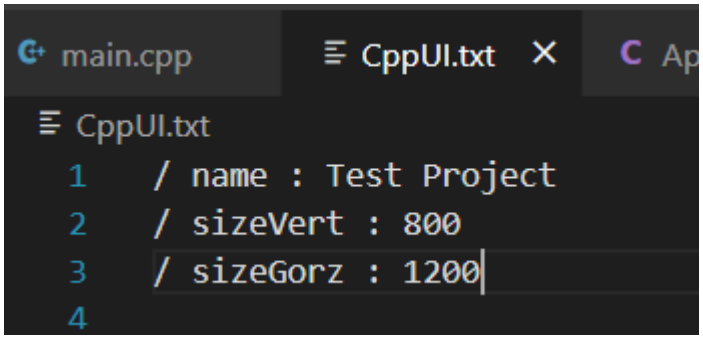



Рисунок 2.1 – Приклад файлу розмітки

Далі користувачу потрібно у функції main() створити об’єкт BaseApplication (є частиною бібліотеки, потрібно включити відповідний файл заголовку) та викликати його метод runMessageLoop. Після успішної компіляції та запуску з’явиться вікно, яке показане на рисунку 2.2.

Рисунок 2.2 – Створене вікно на основі параметрів у файлі розмітки

# 3 ВИКОНАННЯ ПРОГРАМИ

Надалі, для створення графічного інтерфейсу користувач повинен визначити параметри необхідних віджетів у файлі розмітки. Приклад файлу розмітки, що містить опис параметрів вікна, текстового напису та кнопки, показано на рисунку 3.1.

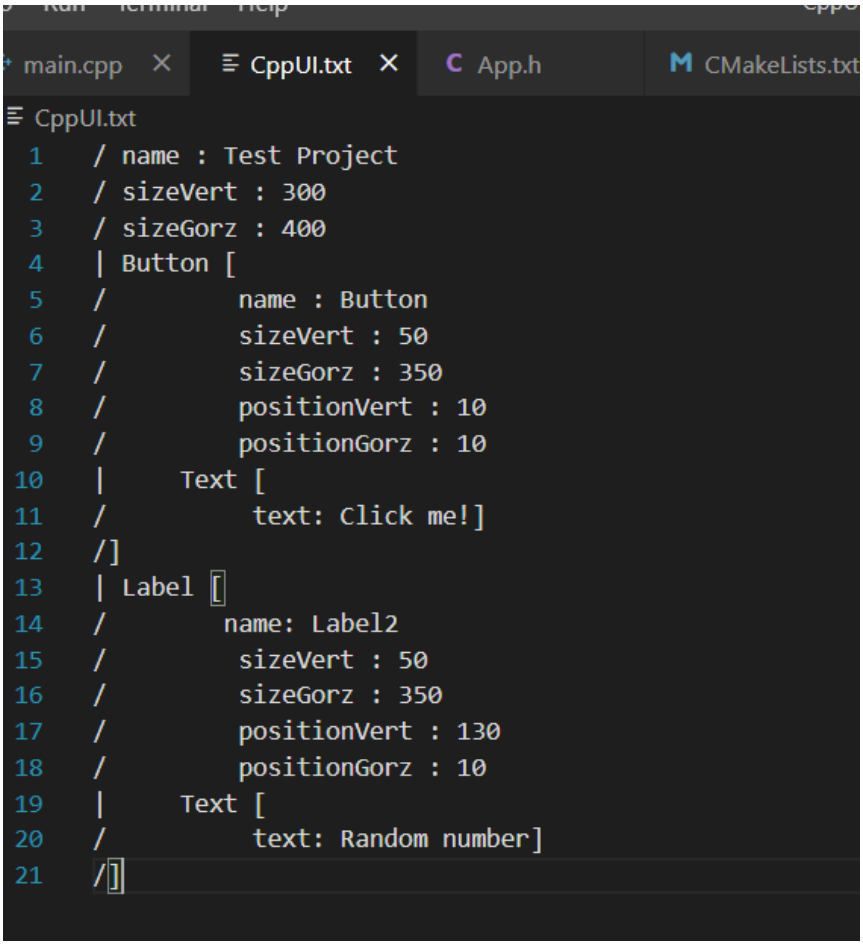



Рисунок 3.1 – Приклад файлу розмітки з кнопкою та текстовим написом

Далі користувачу потрібно створити клас Application та наслідується від класу BaseApplication, що є частиною бібліотеки. Цей клас буде головним класом графічного інтерфейсу. Для опрацювання взаємодії з віджетами (наприклад, натискання на кнопку) потрібно у вигляді методу класу Application реалізувати очікувану поведінку та прив’язати її до відповідної події. На рисунках 3.2 – 3.4 показано приклад програмування інтерфейсу з реалізацією поведінки, що спрацює при натисканні кнопки.

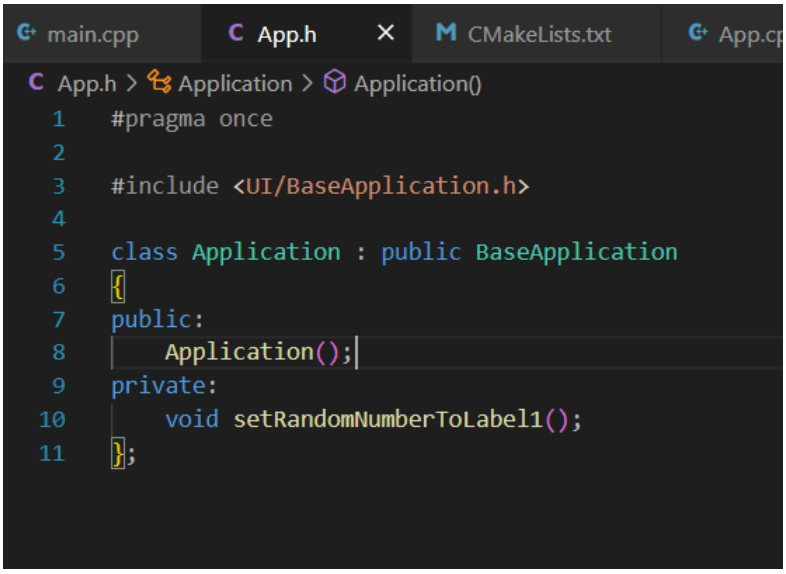



Рисунок 3.2 – Файл “App.h”

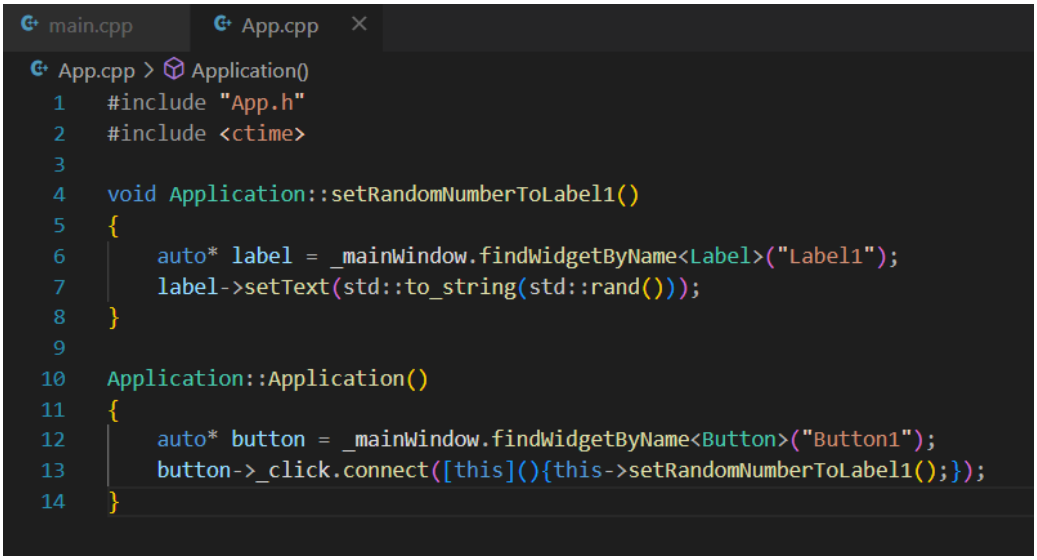



Рисунок 3.3 – Файл “App.cpp”

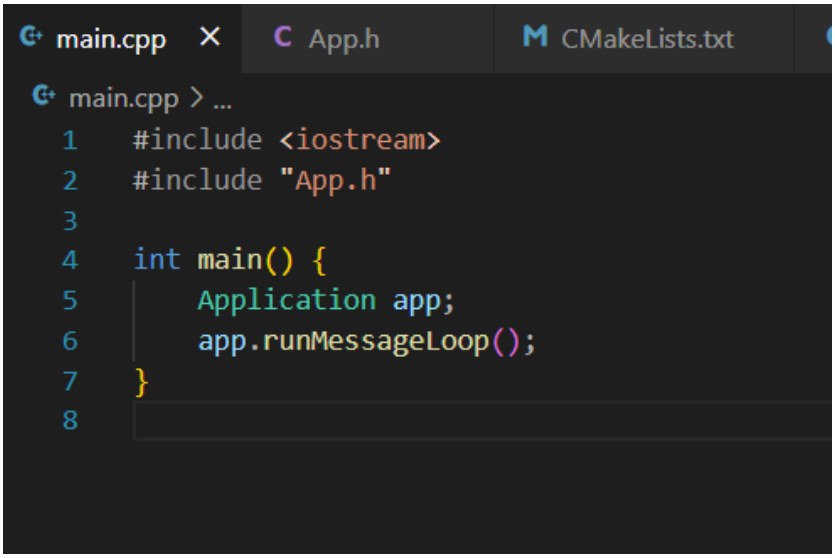



Рисунок 3.4 – Файл “main.cpp”

На рисунку 3.5 можна побачити спроектований для прикладу інтерфейс.

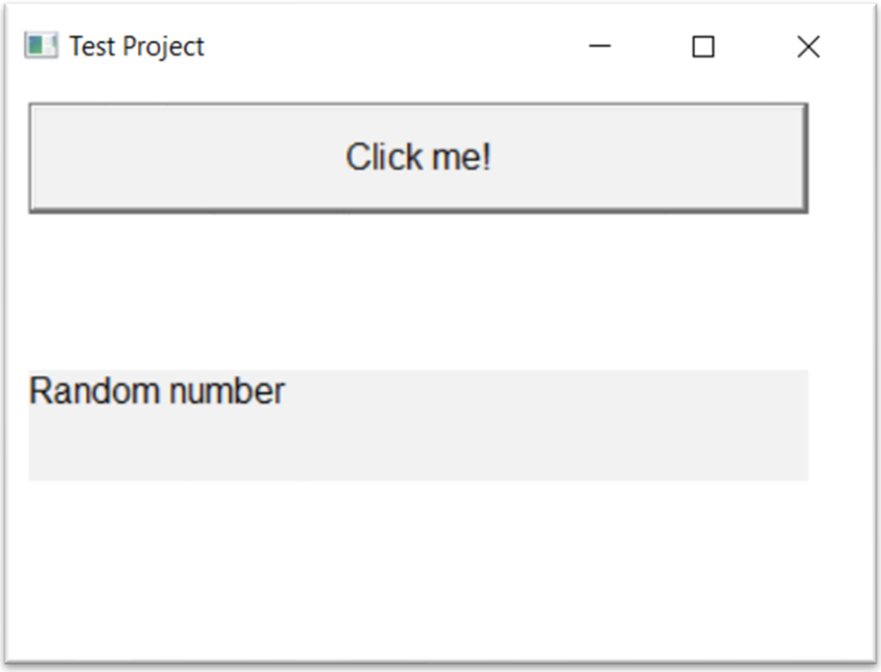



Рисунок 3.5 – Спроектований інтерфейс

На рисунку 3.6 можна побачити стан графічного інтерфейсу після натискання кнопки.

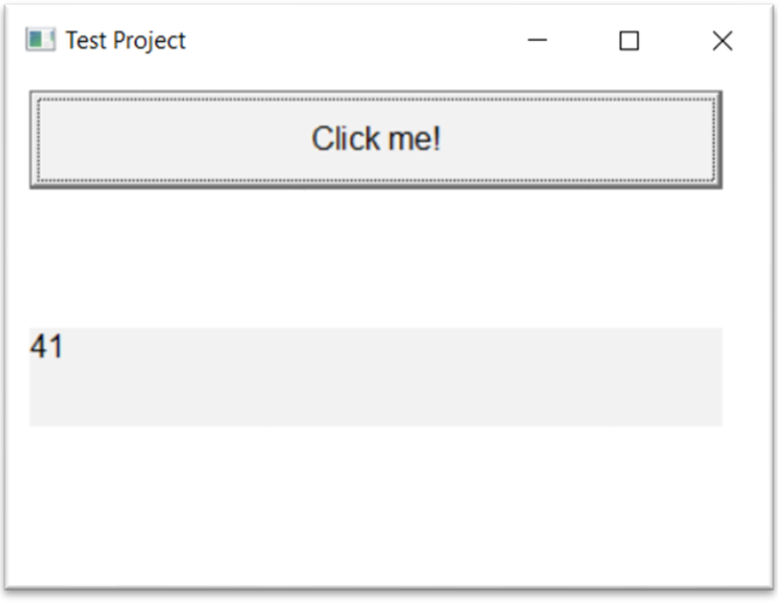



Рисунок 3.6 – Стан інтерфейсу після натискання на кнопку

Виконуючи такі ж кроки, що описані в прикладі, але відповідно до своїх вимог, користувач зможе створити необхідний йому графічний інтерфейсу користувача.