REV	DATA	ZMIANY
0.1		*Konrad Maciejewski (kmaciejewski@student.agh.edu.pl)*

BAZA DANYCH ELEMENTÓW ZŁĄCZNYCH DO KONSTRUKCJI STALOWYCH

Autor: Konrad Maciejewski

Student kierunku Elektronika i Telekomunikacja

Akademia Górniczo-Hutnicza

Spis treści

Lista oznaczeń

CLI	Command-Line Interface
DIN	Deutsches Institut für Normung – norma służąca do standaryzacji różnego rodzaju przedmiotów i procedur
GUI	Graphical User Interface
HDG	Hot Dip Galvanized
JSON	JavaScript Object Notation

1. Wstęp (Introduction)

Dokument dotyczy stworzenia aplikacji bazy danych, zawierającej informacje o elementach złącznych przeznaczonych do montażu części konstrukcji stalowych. Docelowym użytkownikiem tego oprogramowania jest producent wyrobów stalowych, którzy tworzy katalog elementów złącznych. Taki katalog dostarczany jest do wyspecjalizowanych firm budowlanych, które oferują usługi w zakresie opracowania technologii, produkcji i montażu konstrukcji stalowych.

2. Założenia projektowe (assumptions)

Podstawowe założenia projektu:

1. Przygotowanie syntetycznego opisu bazy danych zawierającej informacje o elementach złącznych
2. Zdefiniowanie struktury rekordów obejmujących istotne atrybuty
3. Szczegółowe określenie funkcji świadczonych przez aplikację
4. Opracowanie architektury systemu oraz dalsze określanie wymagań takich jak m.in. wygląd i funkcjonalność interfejsu graficznego aplikacji
5. Opracowanie głównych modułów aplikacji:
    - Dodawanie rekordów
    - Usuwanie rekordów
    - Wyszukiwanie rekordów w bazie danych
    - Odczytywanie danych z pliku JSON
    - Zapisywanie rekordów do pliku JSON
6. Stworzenie interfejsu graficznego użytkownika (GUI) i zapewnienie designu o intuicyjnej strukturze
7. Zapewnienie kompatybilności modułów z interfejsem graficznym
8. Opracowanie drugoplanowych modułów aplikacji takich jak m.in. odświeżanie bazy danych i obsługa sytuacji wyjątkowych
9. Testowanie aplikacji z wykorzystaniem bibliotek zewnętrznych, w celu zapewnienia stabilności i poprawności funkcji

3. Wymagania systemowe (requirements)

Obsługiwane systemy operacyjne	Aplikacja może zostać uruchomiona w następujących systemach operacyjnych: • Windows 11 • Linux (Ubuntu 22.04)
Obsługiwane języki	Aplikacja jest dostępna jedynie w języku angielskim
Dodatkowe wymagania i wskazówki	Zalecane jest zainstalowanie i skonfigurowanie *Visual Studio* (Windows)/*Visual Studio Code* (Ubuntu) wraz z odpowiednimi rozszerzeniami umożliwiającymi rozwijanie, budowanie i debugowanie aplikacji napisanej w języku C++ Niezbędne jest zainstalowanie rozszerzenia *Qt VS Tools* (Windows) oraz pakietu *qt6-base-dev* (Ubuntu), które umożliwią tworzenie, budowanie, debugowanie i uruchamianie aplikacji bazującej na bibliotece *Qt*. Możliwe jest również skorzystanie z dedykowanego oprogramowania *Qt Creator*, zarówno na Windows jak i na Linux. Narzędzia dodatkowe, kluczowe przy budowaniu aplikacji: • *CMake*, wersja 3.16 i nowsze • *Ninja* (build system) • *Python*, wersja 3

4. Funkcjonalność (*functionality*)

Interfejs użytkownika został starannie zaprojektowany w sposób intuicyjny, celem ułatwienia użytkownikowi korzystania z aplikacji, jednocześnie zapewniając funkcjonalność i użyteczność. Okno główne aplikacji zawiera pasek tytułu, elementy sterujące, tabelę z rekordami bazy danych oraz pozostałe elementu interfejsu zarządzające bazą danych.

Program realizuje następujące funkcjonalności:

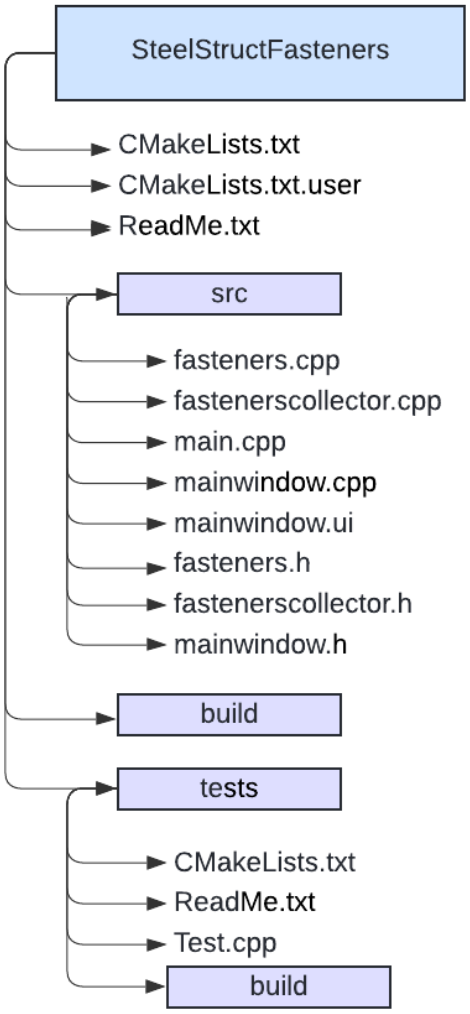
Dodawanie rekordów do bazy danych	Każde pole rekordu wypełniane jest zawartością odpowiedniego pola tekstowego w interfejsie graficznym, a następnie cały rekord wstawiany jest na koniec istniejącego wektora, który kolekcjonuje wpisy w bazie danych, o ile dane zostały wpisane poprawnie
Usuwanie rekordów z bazy danych	Funkcja przyjmuje numer identyfikacyjny, a następnie przeszukuje bazę danych w celu znalezienia rekordu zawierającego podany argument; jeśli zostanie znaleziony, rekord jest usuwany z wektora bazy danych
Wyświetlanie rekordów bazy danych	Funkcja przyjmuje jeden rekord i dodaje nowy wiersz do tabeli w interfejsie graficznym
Odświeżanie rekordów bazy	Funkcja pobiera cały wektor bazy danych ze wszystkimi rekordami i w pętli realizuje wyświetlanie rekordów bazy danych
Przeszukiwanie bazy danych	Umożliwia przeszukiwanie rekordów bazy danych wyszukując po określonej kolumnie; argumentami wejściowymi jest nazwa kolumny i wartość wyszukiwanego pola rekordu; zwraca wektor zawierający wszystkie pasujące rekordy, o ile obie wartości wyszukiwania zostały wypełnione
Dodawanie danych z pliku typu JSON	Otwiera plik JSON i dokonuje parsowania danych, jeśli dane są poprawne i reprezentują tablicę, iteruje po elementach tej tablicy; dla każdego rekordu sprawdza czy jest on poprawny i czy nie istnieje już w bazie danych; jeśli wszystkie dane są poprawne dodaje nowy rekord do bazy danych

Zapisywanie danych do pliku JSON	Iteruje po wszystkich elementach wektora bazy danych, dla każdego z rekordów tworzy obiekt JSON, następnie zapisuje tablicę do dokumentu JSON, a następnie zapisuje dane do pliku JSON

Do aplikacji dołączono również pliki konfiguracyjne do przeprowadzenia testów jednostkowych. Zapewniają one stabilność i poprawność aplikacji i zostały zaimplementowane przy użyciu biblioteki zewnętrznej *QtTest*. Zostały zdefiniowane dwa testy jednostkowe dla aplikacji bazy danych. Pierwszy – sprawdzający działanie funkcji dodawania rekordu do bazy danych i drugi – sprawdzający poprawność funkcji usuwania z bazy danych.
Aplikacja jest kompatybilna z dwoma systemami operacyjnymi, co zwiększa jej dostępność dla użytkowników korzystających z różnych platform.

5. Struktura plików programu (file structure)

Struktura plików projektu prezentowała się jak poniżej:



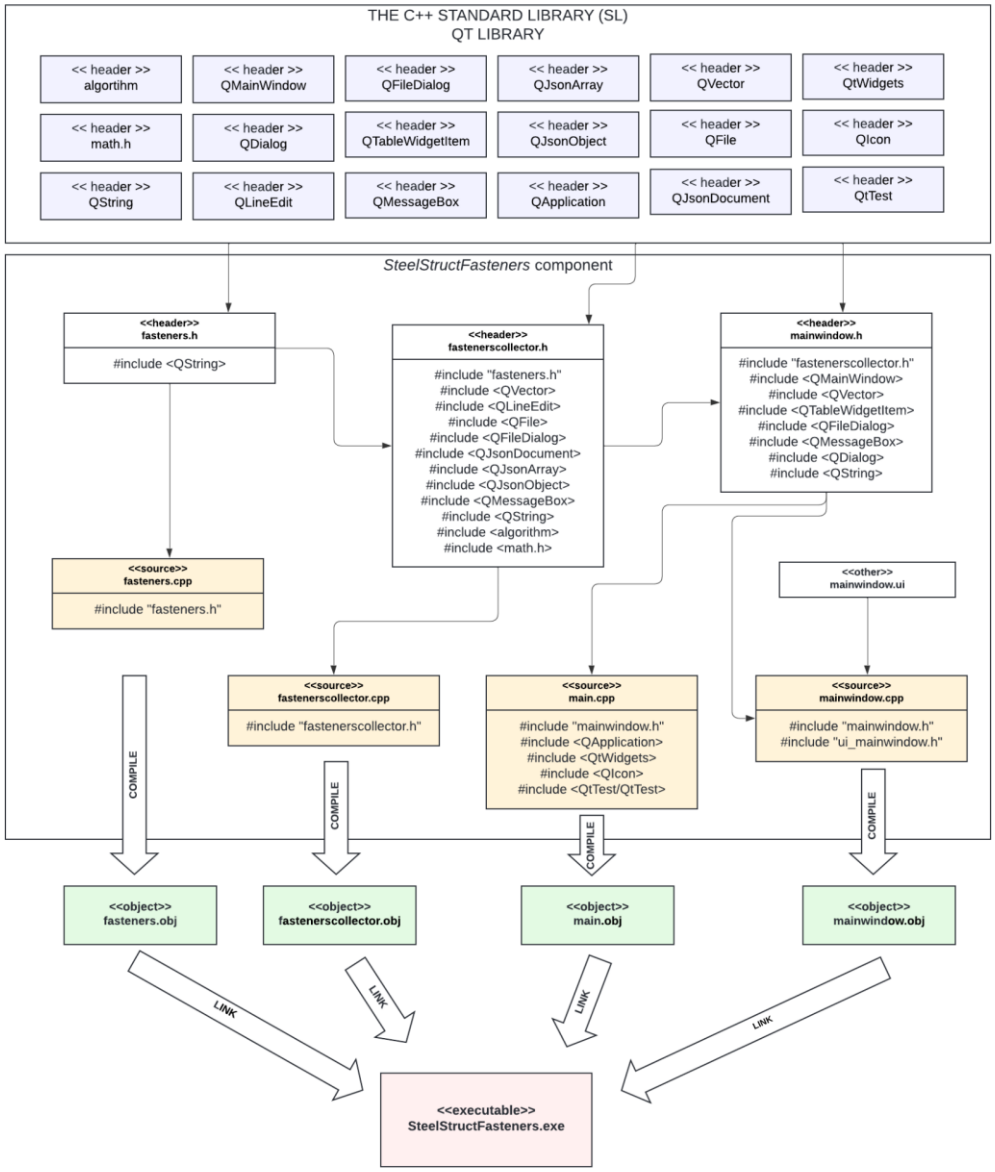
Rys 1. Struktura plików projektu

6. Projekt techniczny (*technical design*)

W tej części zostaną zaprezentowane diagramy UML, które opisują granice, strukturę i zachowanie systemu oraz obiektów w nim zawartych.

6.1 Diagram artefaktów UML

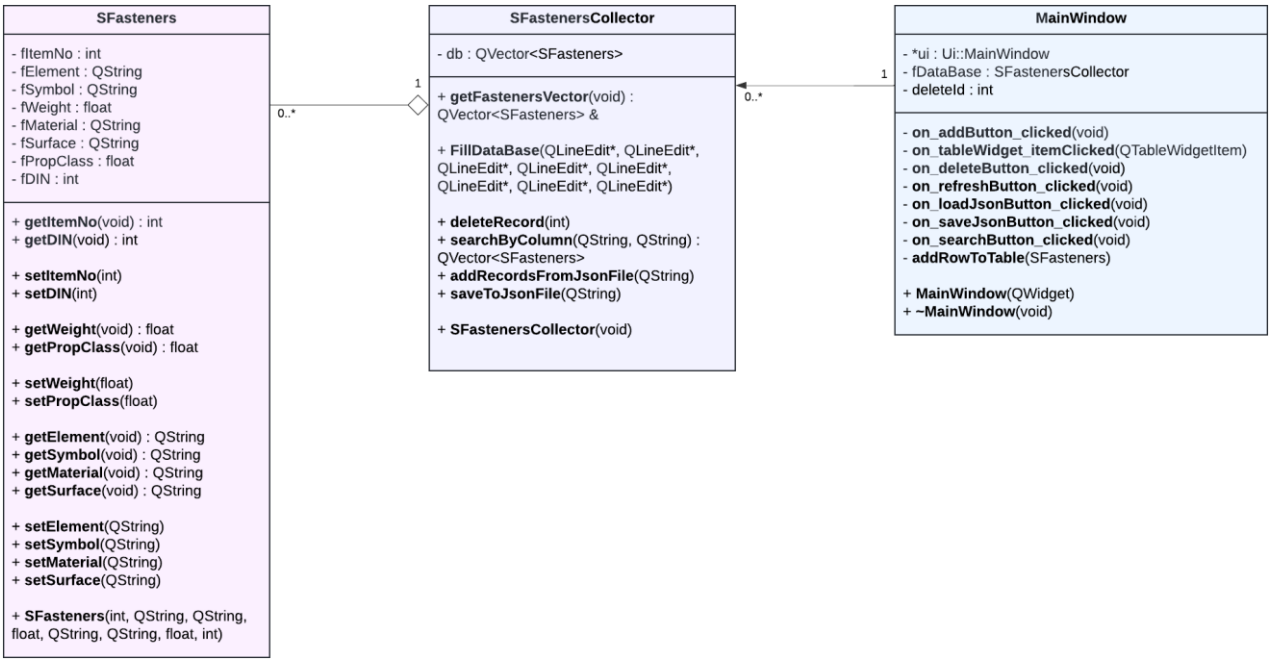
Kod umieszczony jest w dwóch rodzajach plików: plikach nagłówkowych i plikach źródłowych. Nagłówkowe pliki zazwyczaj zawierają jedynie deklaracje, natomiast pliki źródłowe obejmują kompleksowe definicje.



Rys 2. Diagram artefaktów UML ukazujący relację i rolę plików w projekcie SteelStructFasteners

6.2 Hierarchia klas głównych

Aby lepiej odzwierciedlić strukturę klas i relacje pomiędzy nimi stworzono diagram klas UML użytych w oprogramowaniu.

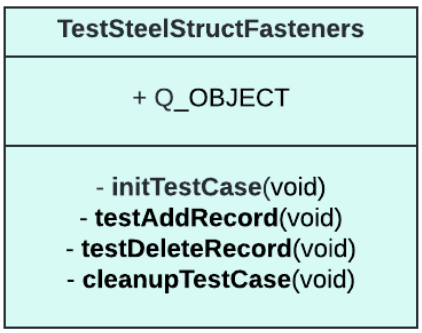


Rys 3. Relacje między klasami SFasteners, SFastenersCollector, MainWindow

Klasa *MainWindow* używa obiektu *SFastenersCollector*, co oznacza, że *MainWindow* krozysta z funkcji i danych dostarczanych przez *SFastenersCollector* Natomiast, *SFastenersCollector* posiada obiekt *SFasteners*, co oznacza, że przechowuje instancje *SFasteners* w swojej bazie danych. Zostały wykorzystane relacje powiązania i agregacji klas.

6.3 Implementacja testów jednostkowych

Testy jednostkowe zostały zdefiniowane przy użyciu biblioteki zewnętrznej *QtTest* w pliku *Test.cpp*. Tym razem deklaracje i definicje funkcji znalazły się w jednym pliku. Poniżej diagram UML klasy *TestSteelStructFasteners*:



Rys 4. Diagram UML dla klasy TestSteelStructFasteners

7. Opis realizacji (*implementation report*)

W trakcie implementacji projektu aplikacji *SteelStructFasteners* została użyta platforma testowa w postaci prywatnego komputera. Poniżej znajdują się szczegóły tego urządzenia.

Model urządzenia: Asus ZenBook z procesorem opartym na architekturze x64, 16 GB RAM
System operacyjny: Windows 11, Linux Ubuntu 22.04 (dzięki wykorzystaniu narzędzia do wirtualizacji Oracle VirtualBox – 4 rdzenie procesora, 10GB RAM)
Narzędzia: *Visual Studio 2022* (Windows) wraz z *Qt VS Tools* i rozszerzeniami do programowania w języku C++, *Visual Studio Code* (Ubuntu), *Qt Creator 12.0.1* (Community), *CMake* 3.27.6, *Ninja* (build tool), *Python* 3.12.0
System kontroli wersji: *Git*, hostowane na prywatnym koncie GitHub

Projekt został podzielony na moduły zgodnie z zasadami programowania obiektowego. Struktura katalogów została zorganizowana w sposób uporządkowany, aby rozgraniczyć pliki testów jednostkowych i pliki głównej aplikacji.

CMake został wykorzystany jako narzędzie do konfigurowania projektu, co umożliwiło jego przenośność pomiędzy różnymi platformami. *Visual Studio/Visual Studio Code* (w zależności od platformy) i *Qt Creator* pełniły funkcję głównych platform do edycji kodu źródłowego. Interfejs graficzny zaprojektowano w *Qt Creator*. Natomiast funkcję budowy aplikacji przypisano tylko programowi *Visual Studio/Visual Studio Code*.

Do monitorowania błędów wykorzystano mechanizmy wbudowane w *Qt*, takie jak wyświetlanie komunikatów błędów za pomocą klasy QMessageBox. W trakcie implementacji stosowano techniki debugowania dostępne w środowisku *Qt Creator*.

Testy jednostkowe były integralną częścią procesu implementacji. Wykorzystano w tym celu zewnętrzną bibliotekę *QtTest*. Testy były wykonywane na dwóch różnych scenariuszach. Do przeprowadzenia testów i weryfikacji wyników użyto narzędzia *CTest*, które zapewnia *CMake*. Wyniki testów wyświetlane były w oknie konsolowym.

Regularne kopie zapasowe były tworzone na dysku, zwłaszcza przed wprowadzaniem większych zmian w kodzie programu.

Podsumowując, proces implementacji aplikacji przebiegał w kontrolowanym, uporządkowanym i zorganizowanym środowisku, z uwzględnieniem praktyk programistycznych.

8. Opis wykonanych testów jednostkowych (*testing report*)

Testy jednostkowe zostały zadeklarowane w pliku *Test.cpp* i są operacjami klasy *TestSteelStructFasteners*.

Oprócz testu początkowego, wywoływanego przed rozpoczęciem wszystkich innych testów i testu końcowego, „sprzątającego" wywoływanego po zakończeniu testów, do obsłużenia pozostały jeszcze dwa scenariusze. Mianowicie, dodawanie rekordu do bazy danych i usuwanie rekordu z bazy danych. Poniżej znajduje się opis wykonanych czynności w celu sprawdzenia poprawności zadeklarowanych funkcji testujących.

1. **Sprawdzanie prawidłowości działania funkcji dodawania rekordu do bazy danych**
   Tworzony jest obiekt *SFastenersCollector* oraz obiekty QLineEdit, które symulują wprowadzenie danych przez użytkownika. Został zdefiniowany przykładowy zbiór danych. Funkcja QVERIFY sprawdza, czy baza danych jest początkowo pusta, a następnie wywoływana jest funkcja FillDataBase, aby dodać rekord. Następnie sprawdzane jest za pomocą QCOMPARE, czy rozmiar wektora przechowującego rekordy bazy danych jest równy 1.
2. **Sprawdzenie prawidłowości działania funkcji usuwania rekordu z bazy danych**
   Podobnie jak w poprzednim teście, tworzony jest jeden obiekt *SFastenersCollector* oraz obiekty QLineEdit w celu dodania kilku rekordów. Zostają dodane trzy przykładowe rekordy do bazy danych. Następnie sprawdzanie jest, czy rozmiar bazy danych jest większy od zera. Następnie pobierany jest identyfikator rekordu o indeksie 0 i przekazywany do funkcji deleteRecord w celu usunięcia. Sprawdzane jest, czy rozmiar bazy danych zmniejszył się o jeden. Dla upewnienia, na końcu sprawdzano, czy rekord o danym identyfikatorze nie istnieje już w bazie danych.

Makro QTEST_MAIN tworzy funkcję main dla testów jednostkowych *Qt*. Wyniki testów zostały wyświetlane w wierszu poleceń (Windows) lub terminalu (Linux). W celu uruchomienia testów jednostkowych użyto komendy *ctest -C <cfg> --verbose --test-dir <dir>*. Zdecydowano się na opcję verbose, celem wyświetlenia szczegółowych informacji.

Poniżej wstawiono logi z CLI po wykonaniu komendy i pogrubioną czcionką zaznaczono najważniejsze informacje.

```
Start 1: SteelStructFastenersTests
1: Test command:
C:\Users\macie\Documents\QTProjects\SteelStructFasteners\tests\build\Debug\SteelStructFastenersTests.exe
1: Working Directory: C:/Users/macie/Documents/QTProjects/SteelStructFasteners/tests/build
1: Test timeout computed to be: 10000000
1: ********* Start testing of TestSteelStructFasteners *********
1: Config: Using QtTest library 6.7.0, Qt 6.7.0 (x86_64-little_endian-llp64 shared (dynamic) debug build; by
MSVC 2019), windows 11
1: QDEBUG : TestSteelStructFasteners::initTestCase() Called before everything else.
1: PASS   : TestSteelStructFasteners::initTestCase()
1: PASS   : TestSteelStructFasteners::testAddRecord()
1: PASS   : TestSteelStructFasteners::testDeleteRecord()
1: QDEBUG : TestSteelStructFasteners::cleanupTestCase() Called after myFirstTest and mySecondTest.
1: PASS   : TestSteelStructFasteners::cleanupTestCase()
```

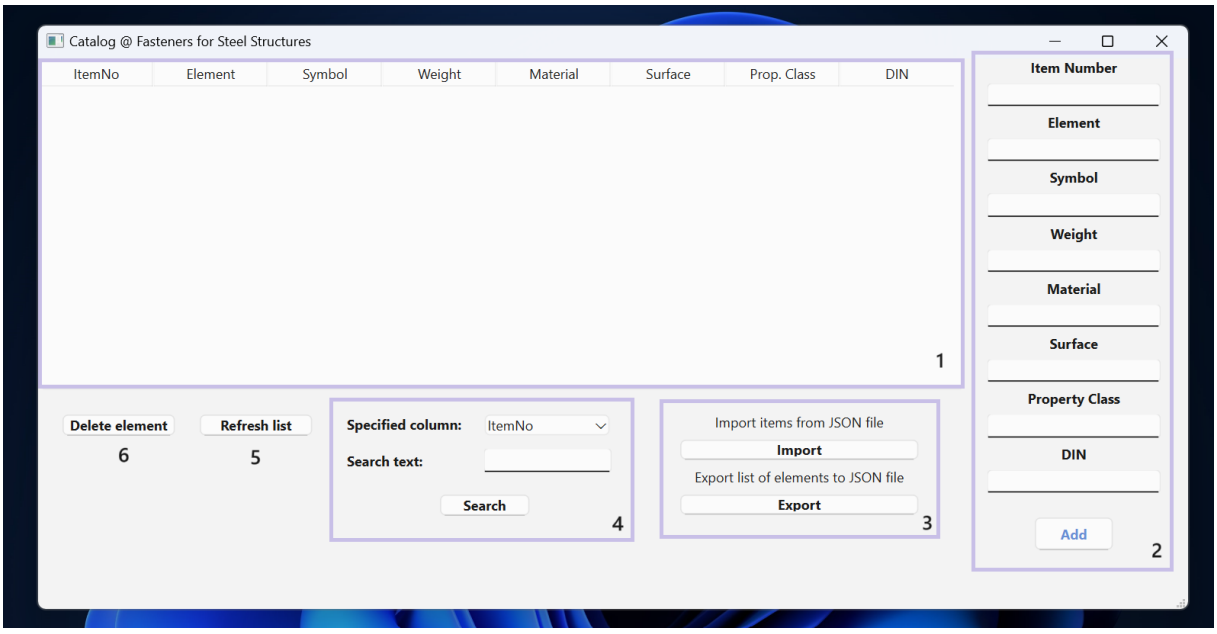
```
1: Totals: 4 passed, 0 failed, 0 skipped, 0 blacklisted, 36ms
1: ********* Finished testing of TestSteelStructFasteners *********
1/1 Test #1: SteelStructFastenersTests .....   Passed    0.90 sec

100% tests passed, 0 tests failed out of 1
Total Test time (real) =   0.92 sec
```

9. Podręcznik użytkownika (*user's manual*)

9.1 Aplikacja testowa dla platformy Windows 11

Aplikacja *SteelStructFasteners.exe* jest przeznaczona do zarządzania bazą danych elementów złącznych do konstrukcji stalowych. Wszystkie funkcjonalności aplikacji są wykonywane z poziomu okna głównego. Interfejs graficzny został zaprojektowany przy użyciu biblioteki *Qt*. Przy początkowym uruchomieniu aplikacji, ekran główny wygląda jak poniżej. Rysunek podzielono również na sekcje, które pełnią odpowiednie funkcje w aplikacji.

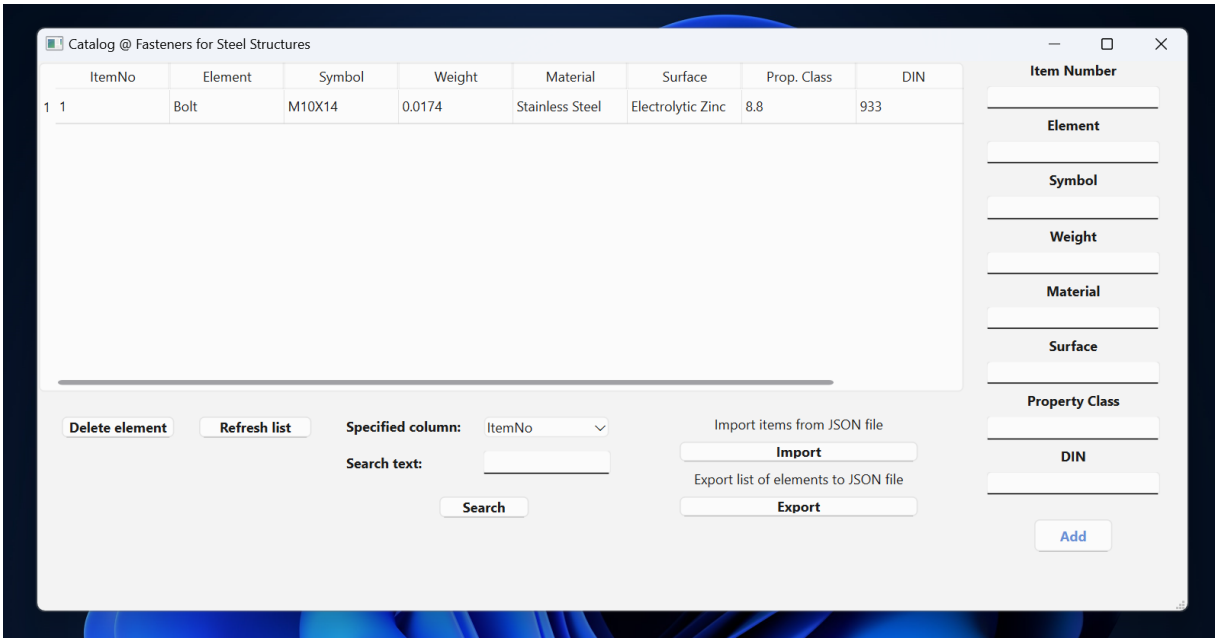


Rys 5. Okno główne aplikacji przy pierwszym uruchomieniu

Opis funkcjonalności realizowanej w każdej sekcji:

1. **Tabela**, w której będą wyświetlane rekordy zgromadzone w bazie danych, definiujące dany element złączny
2. **Ręczne wprowadzanie danych**

   Umożliwia dodawanie nowych rekordów poprzez ręczne wpisywanie wartości w odpowiednich polach tekstowych, które cechują dany element złączny. Pola te zdefiniowano jako: Item Number, Element, Symbol, Weight, Material, Surface, Property Class, DIN. Dodanie element do bazy danych zatwierdza się poprzez kliknięcie przycisku „Add”.



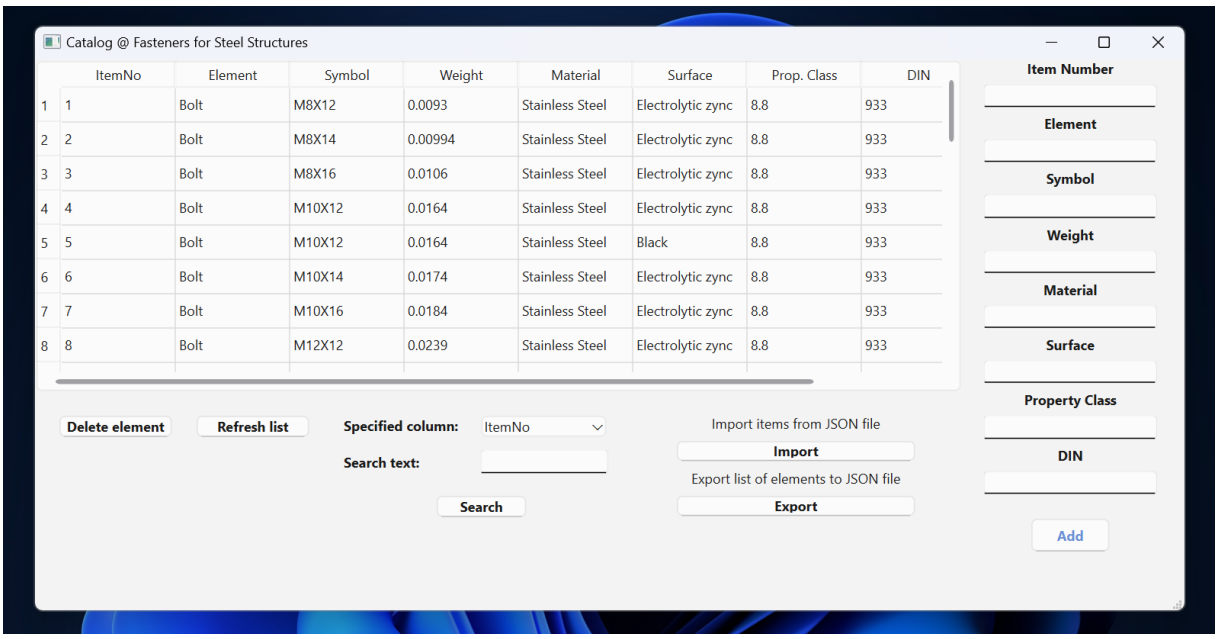
Rys 6. Okno główne aplikacji po dodaniu ręcznie jednego rekordu

3. **Importowanie/eksportowanie elementów z/do pliku typu JSON**

Przycisk **Import** pozwala na bezpośredni import pliku typu JSON z dysku komputera, w którym zapisane są dane elementów, które mają zostać dodane do bazy danych, ułatwia to dodanie większej ilości wpisów

Przycisk **Export** pozwala wyeksportować wszystkie wpisy z bazy danych do pliku typu JSON, można zdefiniować miejsce lokalizacji pliku i jego nazwę po naciśnięciu przycisku

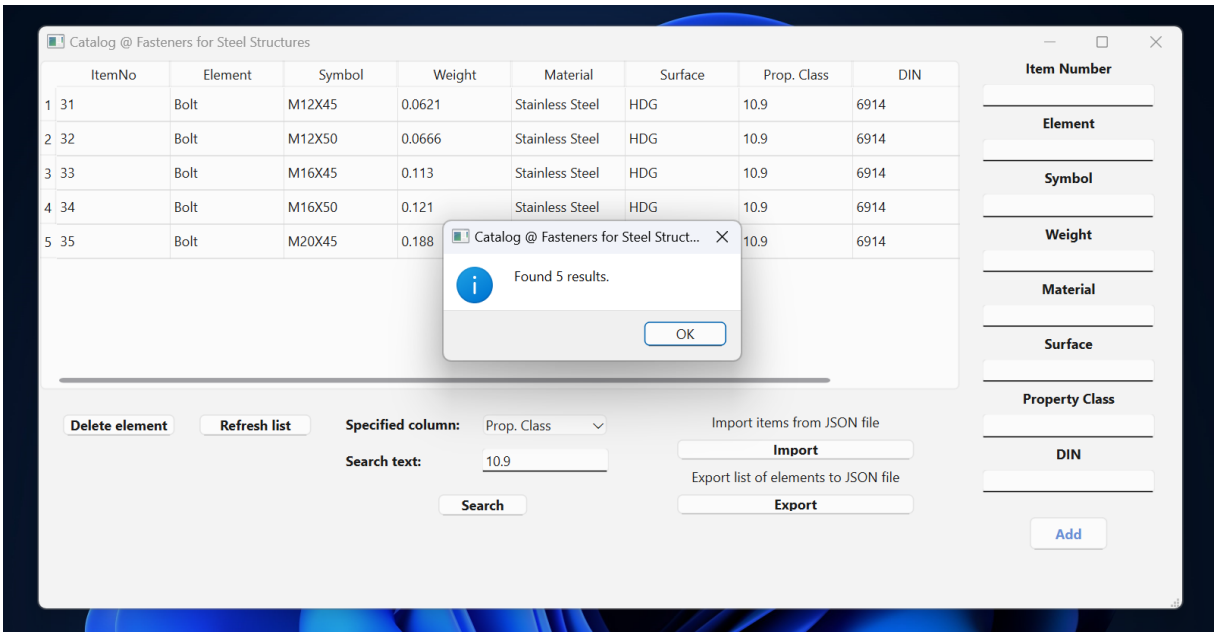
Został przygotowany również przykładowy zestaw danych, które pochodzą z katalogów firmy „Asmet" sp. z o.o., która jest liderem branży elementów złącznych w Polsce. Wspomniany zestaw danych zostanie dołączony do plików projektu (plik: dataset.json) i na nim zostaną przeprowadzone testy aplikacji.



Rys 7. Okno główne aplikacji po wykonaniu funkcji zaimportowania przykładowego zestawu elementów

4. Wyszukiwanie rekordu z określonym polem

Aby wyszukać rekord z konkretną wartością należy określić, która kolumna ma zostać przeszukana, a następnie wpisać dokładną wartość, która jest poszukiwana. Gdy oba te warunki zostały spełnione, należy nacisnąć przycisk „Search”, wtedy wyświetli się informacja ile rekordów zostało znalezionych i zostaną one wylistowane.



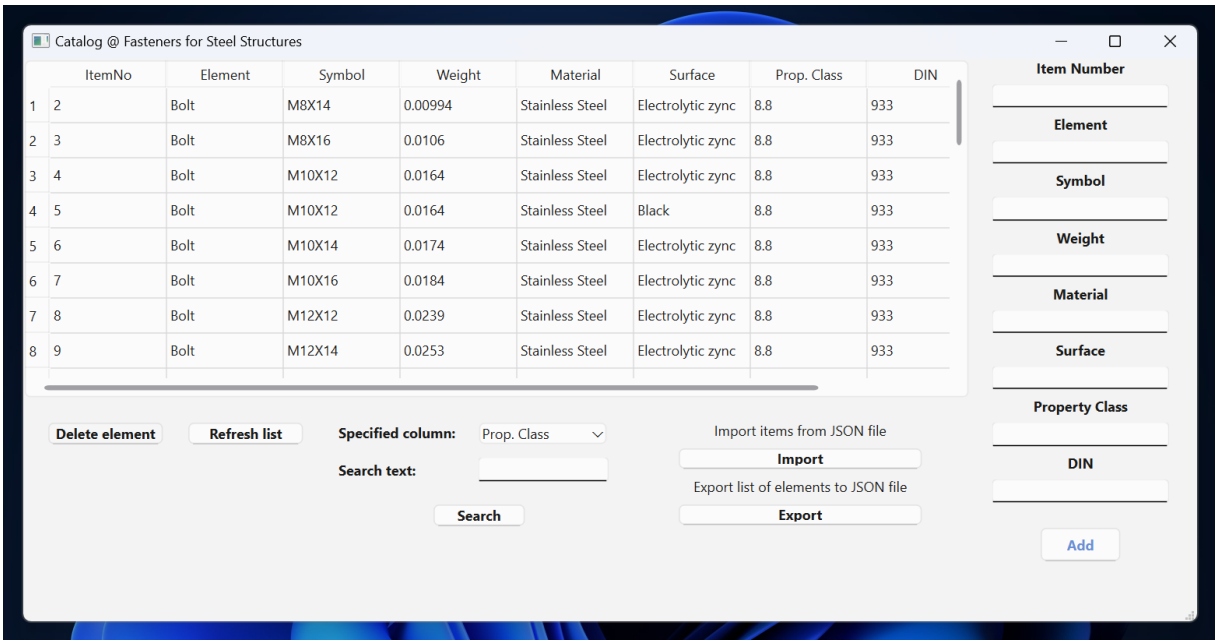
Rys 8. Okno główne po wykonaniu funkcji wyszukania rekordów, których klasa właściwości wynosi 10.9

5. **Odświeżanie bazy danych**

   Umożliwia odświeżenie widoku wszystkich elementów w bazie danych, jest konieczne wtedy, gdy skorzystano z opcji wyszukiwania rekordu, aby powrócić do widoku wszystkich rekordów w bazie danych. Aby skorzystać z tej opcji, należy nacisnąć przycisk „Refresh list".

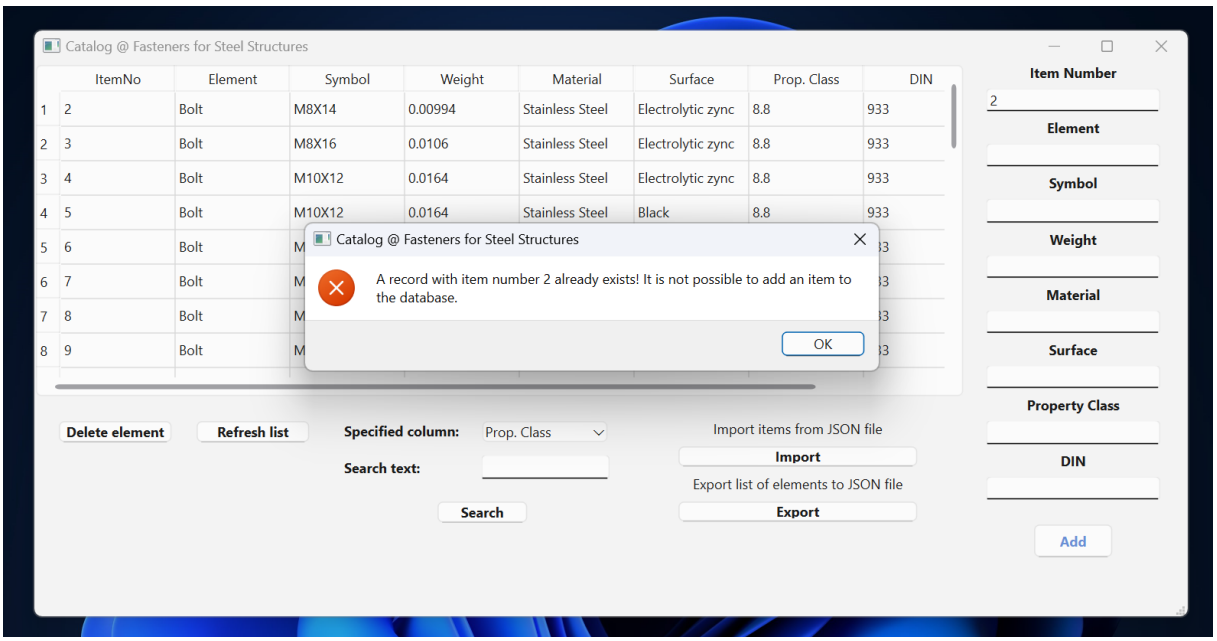
6. **Usuwanie jednego elementu z bazy danych**

   Aby usunąć dany rekord z bazy danych, należy zaznaczyć myszką którekolwiek z jego pól i następnie nacisnąć przycisk „Delete element". Element zostanie usunięty, a baza danych zostanie odświeżona.



Rys 9. Okno główne aplikacji po usunięciu rekordu z numerem 1

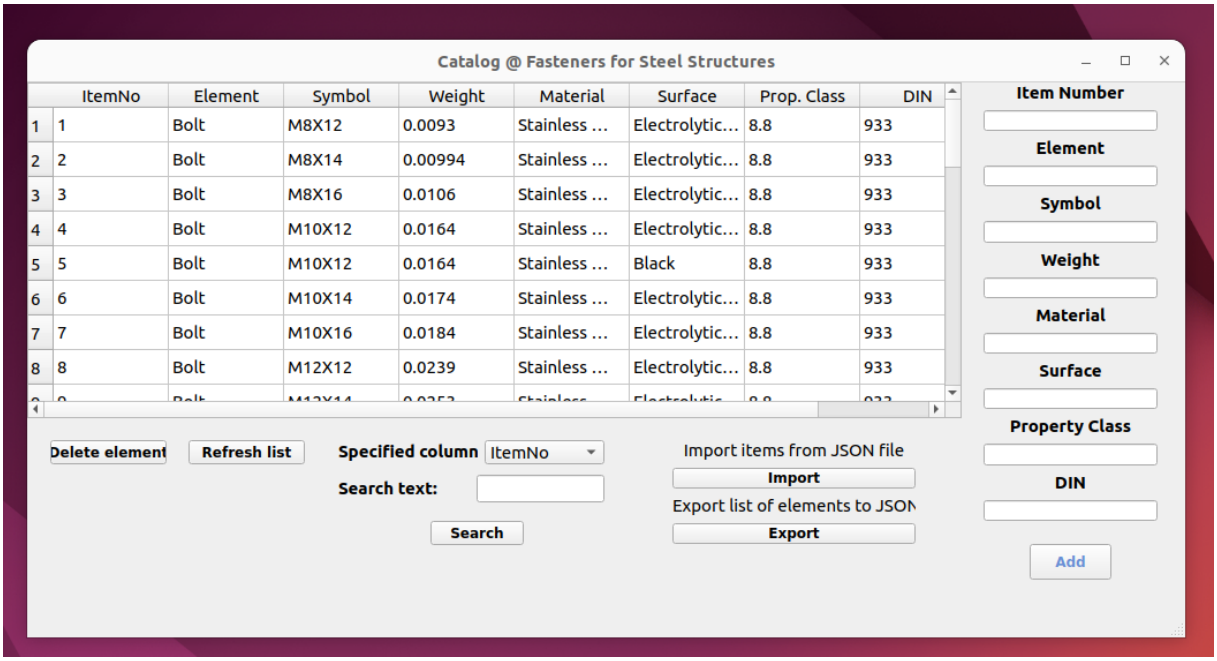
Aplikacja świadczy również funkcje obsługujące sytuacje wyjątkowe. Jedną z takich sytuacji jest intencja wpisania rekordu z istniejącym już w bazie danych Item Number lub pustym/niewłaściwie wypełnionym polem. Inną sytuacją jest niewłaściwe zdefiniowanie warunków wyszukiwania. Jeśli nastąpi którakolwiek z tych sytuacji, zostanie wyświetlone okno z ostrzeżeniem, a rekord nie zostanie dodany, lub nie nastąpi wyszukiwanie w bazie danych.



Rys 10. Okno główne aplikacji, gdy nastąpiła intencja wprowadzenia elementu do bazy danych z istniejącym już Item Number; należy podkreślić, że pozostałe pola są puste, dlatego też nastąpiły by sytuacje wyjątkowe

9.2 Aplikacja testowa dla platformy Linux Ubuntu 22.04

Aplikacja *SteelStructFasteners.exe* na platformie Linux pełni takie same funkcjonalności jak na platformie Windows. Wszystkie zaimplementowane funkcje działają prawidłowo. Wygląd również jest podobny, jednak nieco bardziej dopasowany do wyglądu aplikacji w systemie Linux. Dzięki temu, że framework *Qt* jest również dostarczany na systemy Linux, zaprojektowana aplikacja może być tak dokładnie przystosowana. To wszystko czyni aplikację wieloplatformową.



Rys 11. Okno główne aplikacji w systemie Linux Ubuntu, po zaimportowaniu przykładowego zestawu elementów złącznych

10. Metodologia rozwoju i utrzymania systemu (*system maintenance and deployment*)

Rozwój i utrzymanie takiego zaimplementowanego systemu jest bardzo ważne, ponieważ aplikacja mogłaby znaleźć rzeczywiste zastosowanie w przedsiębiorstwach zajmujących się wyrobami i dystrybucją elementów złącznych. Pierwszym krokiem na pewno byłaby wstępna analiza wymagań użytkowników, określenie priorytetów i celów. Następnie należałoby zaprojektować wstępne rozwiązania, które powinny zostać wprowadzone. Na ten moment baza danych realizuje tylko kilka podstawowych funkcji, w przyszłości mogłaby zostać bardziej rozbudowana np. o funkcję odczytywania danych z plików innego typu niż JSON lub dodanie opcji logowania użytkowników i przydzielanie im odpowiednich praw do edycji lub do odczytu. Kolejnymi krokami byłoby już wdrożenie rozwiązań, utrzymywanie, monitorowanie, aktualizacje aplikacji bazy danych i zapewnianie coraz lepszych zabezpieczeń. Natomiast to jest temat przyszłościowy.

Bibliografia

[1] Cyganek B.: Programowanie w języku C++. Wprowadzenie dla inżynierów. PWN, 2023.

[2] Qt Documentation Website: doc.qt.io

[3] Qt Crash Course for Beginners - Create C++ GUI Apps provided by Sciber channel on YouTube: https://www.youtube.com/watch?v=cXojtB8vS2E

[4] Qt QTableWidget | Design Using Qt Stylesheet | QSS | Qt C++ | Qt Creator | Qt Tutorial provided by Qt with Ketan channel on YouTube: https://www.youtube.com/watch?v=AZ5rEZWeTcE

[5] Karty katalogowe firmy „Asmet" sp. z o.o.: https://asmet.com.pl/karty-katalogowe/