Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji

PROJEKT ZALICZENIOWY: “PUMPAPP” - SYSTEM ZARZĄDZANIA PLANAMI TRENINGOWYMI

JĘZYK PROGRAMOWANIA OBIEKTOWEGO

Autor: Jakub Krzysztof Mikulski
Prowadzący: Prof. dr hab. inż. Bogusław Cyganek

Data: 26 stycznia 2026

REPOZYTORIUM GITHUB:

https://github.com/JMikulski358/PumpApp

Historia prac

Data	Autor	Opis zmian
27.12.2025	Jakub Mikulski	Wybór tematu, wstępna analiza wymagań
03.01.2026	Jakub Mikulski	Projekt architektury, diagramy UML
10.01.2026	Jakub Mikulski	Implementacja klas bazowych (Exercise, Repository)
15.01.2026	Jakub Mikulski	Implementacja przechowywania danych (JSON)
20.01.2026	Jakub Mikulski	Implementacja GUI (Qt Widgets, dialogi)
23.01.2026	Jakub Mikulski	Integracja backendu z GUI, testy funkcjonalne
24.01.2026	Jakub Mikulski	Dodanie testów jednostkowych (Google Test)
25.01.2026	Jakub Mikulski	Publikacja na GitHub, dokumentacja wstępna
26.01.2026	Jakub Mikulski	Finalizacja projektu, kompletna dokumentacja

Spis treści

Lista oznaczeń

Skrót	*Znaczenie*
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
GUI	Graphical User Interface
JSON	JavaScript Object Notation
OOP	Object-Oriented Programming
Qt	Qt Framework (biblioteka GUI)
RAII	Resource Acquisition Is Initialization
STL	Standard Template Library
UML	Unified Modeling Language
RF	Functional Requirements
NRF	Non Functional Requirements

1. Wstęp

1.1 Cel projektu

Celem projektu jest stworzenie systemu zarządzania planami treningowymi na siłowni, zwanego dalej **PumpApp**, który umożliwii użytkownikowi szybkie i efektywne organizowanie i monitorowanie swoich sesji treningowych. Aplikacja łączy w sobie funkcjonalność bazy danych z przejrzystym interfejsem graficznym, pozwalając na:

- Tworzenie i zarządzanie bazą ćwiczeń siłowych
- Projektowanie spersonalizowanych planów treningowych
- Przechowywanie informacji o parametrach treningowych (serie, powtórzenia, ciężar, przerwy)
- Trwałe zapisywanie danych w formacie JSON

Projekt został wykonany w ramach przedmiotu **Język Programowania Obiektowego C++** i stanowi demonstrację praktycznego zastosowania paradygmatu programowania obiektowego wraz z wykorzystaniem wzorców projektowych (design patterns).

1.2 Zakres funkcjonalności

Zarządzanie ćwiczeniami:

- Dodawanie, edycja, usuwanie ćwiczeń
- Kategoryzacja według typu (z obciążeniem / bez obciążenia)
- Przypisywanie grup mięśniowych
- Wyszukiwanie po nazwie

Zarządzanie planami treningowymi:

- Tworzenie nowych planów
- Dodawanie ćwiczeń z parametrami (serie, ilość powtórzeń, ciężar, odpoczynek)
- Edycja i usuwanie planów
- Podgląd szczegółów
- Wyszukiwanie po nazwie

Zapis i przechowywanie:

- Automatyczny zapis do JSON
- Odczyt przy starcie
- Zachowanie stanu między sesjami

1.3 Technologie i narzędzia

Język: C++17 (smart pointery, RAII)
Framework GUI: Qt 6.10.1 (Qt Widgets)
Narzędzia: CMake 3.19+, MinGW 13.1.0, Qt Creator
Kontrola wersji: Git, GitHub
Testowanie: Google Test (GTest)
Platforma: Windows 10/11 (64-bit), Linux (kompatybilność)

1.4 Motywacja

Aplikacja **PumpApp** została zaprojektowana na bazie doświadczeń autora, który jest osobą regularnie trenującą na siłowni. Zdaniem autora każdemu ćwiczącemu przyda się narzędzie do:

- Systematycznego planowania treningów.
- Śledzenia postępów poprzez definiowanie konkretnych parametrów treningowych.
- Unikania powtarzalności i monotonii w treningach poprzez bazę różnorodnych ćwiczeń.

System pozwala na elastyczne zarządzanie treningami przy jednoczesnym zachowaniu prostoty obsługi i przejrzystości interfejsu użytkownika.

2. Wymagania systemowe

W poniższej sekcji autor zamieszcza listę wszystkich wymagań systemowych jakich oczekuje się od aplikacji PumpApp.

2.1 Wymagania funkcjonalne

System **PumpApp** musi spełniać następujące wymagania funkcjonalne:

RF-1: Zarządzanie bazą ćwiczeń

- Dodawanie ćwiczeń z atrybutami: nazwa, opis, grupy mięśniowe, typ
- Edycja i usuwanie ćwiczeń
- Wyszukiwanie ćwiczeń po nazwie

RF-2: Zarządzanie planami

- Tworzenie planów z unikalną nazwą
- Dodawanie ćwiczeń z parametrami (serie, reps, ciężar, odpoczynek)
- Usuwanie ćwiczeń z planu
- Edycja i podgląd planu
- Wyszukiwanie planu po nazwie

RF-3: Zapis i przechowywanie

- Automatyczny zapis danych planu i ćwiczeń do pliku JSON po każdej operacji CRUD
- Wczytywanie danych z pliku przy starcie
- Osobne pliki: exercises.json, plans.json

RF-4: Interfejs użytkownika

- Stworzenie GUI w Qt Widgets
- Dodanie zakładek rozdzielających funkcjonalności
- Implementacja listy i planów z możliwością ich elementów
- Przyciski akcji (Dodaj, Edytuj, Usuń, Podgląd)
- Okna dialogowe do wprowadzania danych

2.2 Wymagania niefunkcjonalne

NFR-1: Użyteczność

- Intuicyjny interfejs aplikacji
- Zrozumiałe komunikaty błędów
- Potwierdzenia przed usunięciem

NFR-2: Niezawodność

- Walidacja danych wejściowych
- Kontrola unikalności nazw
- Obsługa błędów we/wy

NFR-3: Uniwersalność

- Backend niezależny od platformy i GUI
- Kompilacja na Windows i Linux (Qt + CMake)
- JSON zapewnia przenośność danych

NFR-4: Modularność

- Kod komentowany (PL)
- Nazwy semantyczne (ENG)
- Wzorce projektowe ułatwiające rozszerzanie
- Testy jednostkowe (Google Test)

NFR-5: Architektura

- Hierarchia dziedziczenia (polimorfizm)
- Wzorce: Factory Method, Repository, Singleton
- RAII (smart pointery)
- Separacja logiki od GUI

3. Funkcjonalności

System **PumpApp** oferuje kompleksowy zestaw funkcji umożliwiających zarządzanie treningami siłowymi. Poniżej przedstawiono szczegółowy opis poszczególnych funkcjonalności.

3.1 Zarządzanie ćwiczeniami

3.1.1 Dodawanie nowych ćwiczeń

Użytkownik może dodać nowe ćwiczenie do bazy poprzez:

1. Kliknięcie przycisku **"Dodaj"** w zakładce *Ćwiczenia*

2. Wypełnienie formularza w oknie dialogowym:
    - **Nazwa** - unikalna nazwa ćwiczenia (np. "Wyciskanie sztangi na płaskiej").
    - **Opis** - szczegółowy opis techniki wykonania.
    - **Grupy mięśniowe** - wybór z listy rozwijanej lub wpisanie własnej (np. "Chest", "Triceps").
    - **Typ ćwiczenia** - wybór za pomocą radio button:
        - *Bodyweight* (bez obciążenia) - np. pompki, przysiady bez ciężaru.
        - *Weighted* (z obciążeniem) - np. wyciskanie sztangi, martwy ciąg.
3. Potwierdzenie przyciskiem **OK.**

System waliduje dane wejściowe i zapobiega dodaniu ćwiczenia z pustymi polami lub duplikatem nazwy.

3.1.2 Edycja istniejących ćwiczeń

Użytkownik może zmodyfikować dowolne ćwiczenie:

1. Zaznaczenie ćwiczenia na liście.
2. Kliknięcie przycisku **"Edytuj".**
3. Modyfikacja danych w formularzu.
4. Zatwierdzenie zmian przyciskiem **OK.**

Wszystkie atrybuty ćwiczenia mogą być zmieniane, w tym typ ćwiczenia.

3.1.3 Usuwanie ćwiczeń

System umożliwia usunięcie ćwiczenia z bazy:

1. Zaznaczenie ćwiczenia na liście.
2. Kliknięcie przycisku **"Usuń".**
3. Potwierdzenie operacji w oknie dialogowym.

UWAGA: Usunięcie ćwiczenia nie wpływa na plany treningowe, które je zawierają (ćwiczenie jest przechowywane jako kopia w planie).

3.1.4 Wyszukiwanie ćwiczeń

Funkcja wyszukiwania umożliwia szybkie odnalezienie ćwiczenia:

- Wpisanie fragmentu nazwy w pole wyszukiwania.
- Lista jest dynamicznie filtrowana w czasie rzeczywistym.
- Wyczyszczenie pola przywraca pełną listę.

UWAGA: Wielkość liter ma znaczenie

3.1.5 Podgląd listy ćwiczeń

Lista ćwiczeń wyświetla:

- Ikonę typu ćwiczenia:
    - 🏋️ - ćwiczenie z obciążeniem (Weighted).
    - 💪 - ćwiczenie bez obciążenia (Bodyweight).
- Nazwę ćwiczenia.

Lista jest sortowana alfabetycznie.

3.2 Zarządzanie planami treningowymi

3.2.1 Tworzenie nowego planu treningowego

Użytkownik tworzy plan poprzez:

1. Kliknięcie przycisku **"Dodaj"** w zakładce *Plany treningowe.*
2. Wprowadzenie nazwy planu (np. "FBW Poniedziałek", "Push").
3. Dodawanie ćwiczeń do planu:
    - Wybór ćwiczenia z listy rozwijanej
    - Kliknięcie **"Add to Plan"**
    - Wprowadzenie parametrów treningowych:
        - **Serie** - liczba serii (1-20).
        - **Powtórzenia** - liczba powtórzeń w serii (1-100).
        - **Ciężar** - obciążenie w kg (tylko dla ćwiczeń typu Weighted, zakres 0-500 kg (Rekord Pana Hafþór'a Júlíus'a Björnsson'a to ewenement).
        - **Czas odpoczynku** - przerwa między seriami w sekundach (10-600s).
4. Powtórzenie kroku 3 dla kolejnych ćwiczeń.
5. Zatwierdzenie planu przyciskiem **OK.**

System wymaga aby plan zawierał **przynajmniej jedno ćwiczenie** przed zapisaniem.

3.2.2 Edycja planu treningowego

Użytkownik może modyfikować istniejący plan:

1. Zaznaczenie planu na liście.
2. Kliknięcie przycisku **"Edytuj".**
3. W oknie dialogowym można:
    - Zmienić nazwę planu.
    - Dodać nowe ćwiczenia.
    - Usunąć ćwiczenia z planu (przycisk **"Remove from Plan"**).
    - Kolejność ćwiczeń odpowiada kolejności dodawania.
4. Zatwierdzenie zmian przyciskiem **OK.**

3.2.3 Podgląd szczegółów planu

Funkcja **"Podgląd"** wyświetla kompletne informacje o planie:

- Nazwa planu.
- Lista ćwiczeń z pełnymi parametrami

- Informacja wyświetlana jest w oknie dialogowym typu MessageBox.

3.2.4 Usuwanie planów

System umożliwia usunięcie planu:

1. Zaznaczenie planu na liście.
2. Kliknięcie przycisku **"Usuń".**
3. Potwierdzenie operacji.

Usunięcie planu jest nieodwracalne.

3.2.5 Wyszukiwanie planów

Analogicznie do wyszukiwania ćwiczeń - dynamiczne filtrowanie listy planów po nazwie.

3.3 Przechowywanie danych

W celu gromadzenia danych z aplikacji autor zaimplementował następujące funkcjonalności:

- Automatyczny zapis: Po każdej operacji CRUD -> JSON
  Format: exercises.json, plans.json w folderze `data/`
  Odczyt: Przy starcie -> sprawdzenie folderu -> wczytanie -> info w konsoli

Przykładowy zapis do bazy może wyglądać np. tak (dla exercises.json):

```
[
	{
		"name": "Bench Press",
		"description": "Lie on bench and press bar up",
		“muscle”:  “Chest",
		 "type": "weighted"

	}

 ]
```

3.4 Walidacja danych

Aby zapobiec wpisywaniu niepoprawnych danych użyto następujących metod walidacji:

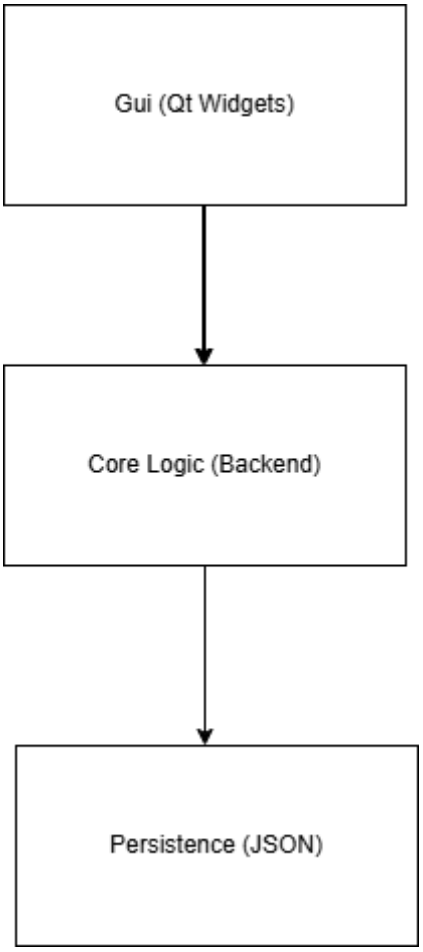
Ćwiczenia:Pola niepuste, unikalna nazwa
Plany: Nazwa niepusta, min. 1 ćwiczenie,unikalna nazwa
Parametry:Zakresy liczbowe (sets, reps, weight, rest)
Komunikaty: QMessageBox przy błędnych danych

4. Analiza problemu

4.1 Programowanie obiektowe

Projekt został oparty na paradygmacie OOP, to jest: modularność, rozszerzalność, łatwość utrzymania.

Ku realizacji tego celu wprowadzono separacja warstw, którą przedstawiono poniżej:



Rys. 4.1.1 Separacja warstw projektu

Zalety tego podejścia to:

- Backend (`core/`) jest **niezależny od GUI** - można łatwo zmienić interfejs (np. na web app).
- Backend jest **niezależny od platformy** - kompiluje się na Windows i Linux.
- Łatwość testowania - klasy można testować bez GUI (Google Test).
- Możliwość rozbudowy - dodanie nowych typów ćwiczeń nie wymaga zmian w GUI.

4.2 Wzorce projektowe

W projekcie zastosowano metodę Factory Method Pattern. Było to podyktowane potrzebą tworzenia obiektów o różnych typach bez uzależnienia ich od konkretnych klas. Metoda ta jest realizowana przez klasę `ExerciseFactory` z metodą `createExercise()`.

Fragment tej klasy prezentujemy poniżej:

```
class ExerciseFactory {

public:

    static std::unique_ptr<Exercise> createExercise(

        ExerciseType type, const std::string& name, ...

    );

};
```

Zalety:

- Klient nie zna konkretnych klas (WeightedExercise, BodyweightExercise)
- Dodanie nowego typu = zmiana tylko w Factory
- Enkapsulacja logiki tworzenia

Klasa ta jest użyta m. in. w: ExerciseDialog, ExerciseRepository::loadFromJSON(), testy

Kolejnym rozwiązaniem projektowym jest Repository Pattern. Pozwala on na oddzielenie logiki dostępu do danych od logiki biznesowej. Klasy Exercise Repository i WorkoutPlanRepository realizują to rozwiązanie jako warstwa abstrakcji.

```
class ExerciseRepository {

private:
```

```
    std::vector<std::shared_ptr<Exercise>> exercises;

    std::string jsonFilePath;

public:

    void addExercise(...);

    std::shared_ptr<Exercise> findByName(...);

    bool removeExercise(...);

    bool saveToJSON();

    bool loadFromJSON();

};
```

Zalety:

- Logika nie wie jak dane są przechowywane (JSON, SQL, XML)
- Łatwa zmiana źródła bez zmian w GUI
- Centralizacja operacji
- Walidacja w jednym miejscu

Autor używa ich m. in w: MainWindow, dialogach i DatabaseManager.

4.2.3 Singleton Pattern

Jak zapewnić że w całej aplikacji istnieje **tylko jedna instancja** managera bazy danych? Jak zapewnić globalny punkt dostępu do tej instancji?

Zdecydowano się na klasę `DatabaseManager` implementującą wzorzec Singleton - wzorzec zapewniający istnienie tylko jednej instancji klasy i zazwyczaj zapewnia dobrze znany, tj. globalny punkt dostępu do niej.:

```
class DatabaseManager {

private:

    DatabaseManager(); // prywatny konstruktor

    DatabaseManager(const DatabaseManager&) = delete;
```

```
public:
    static DatabaseManager& getInstance() {
        static DatabaseManager instance;
        return instance;
    }
};
```

Zalety:

- Pojedyncza instancja (brak niespójności)
- Globalny dostęp
- Lazy initialization

Użycie: MainWindow, dialogi, main.cpp

4.2.4 Template Method Pattern (Hierarchia dziedziczenia)

Jak zdefiniować wspólny interfejs dla różnych typów ćwiczeń, pozwalając jednocześnie na specyficzne zachowanie każdego typu?

Rozwiązaniem może być abstrakcyjna klasa bazowa `Exercise` z czysto wirtualną metodą `getType()`:

```
class Exercise {
protected:
    std::string name, description, targetMuscles;
public:
    virtual ExerciseType getType() const = 0; // czysto wirtualna
    virtual ~Exercise() = default;
};


class WeightedExercise : public Exercise {
```

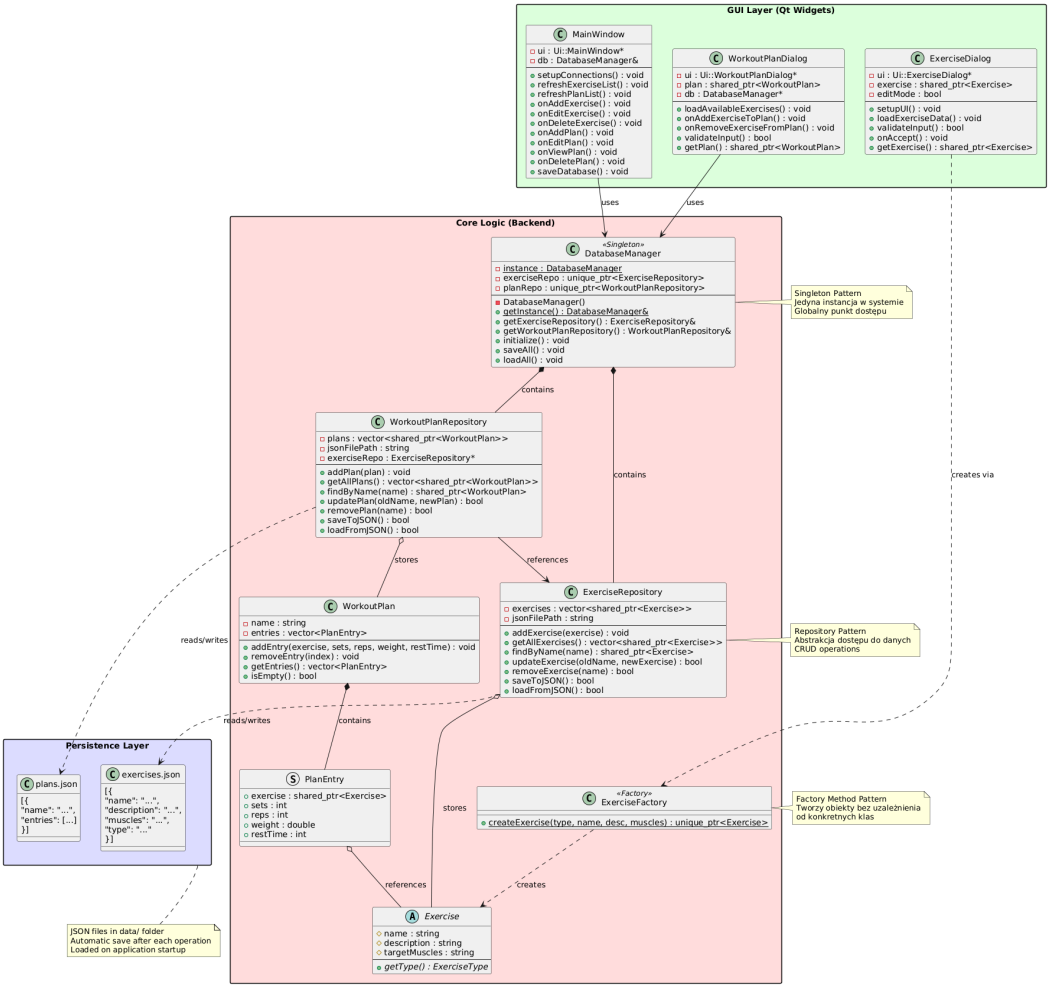
```
public:

    ExerciseType getType() const override { return
ExerciseType::WEIGHTED; }

};
```

Zalety:

- Polimorfizm (vector<shared_ptr<Exercise>>)
- Kod nie musi znać konkretnego typu
- Łatwa rozbudowa (nowa klasa pochodna)

4.3 Architektura systemu - podsumowanie

4.3.1 Diagram zależności modułów



Rysunek 4.3.1.1 Diagram klas UML systemu PumpApp

4.3.2 Kluczowe decyzje projektowe

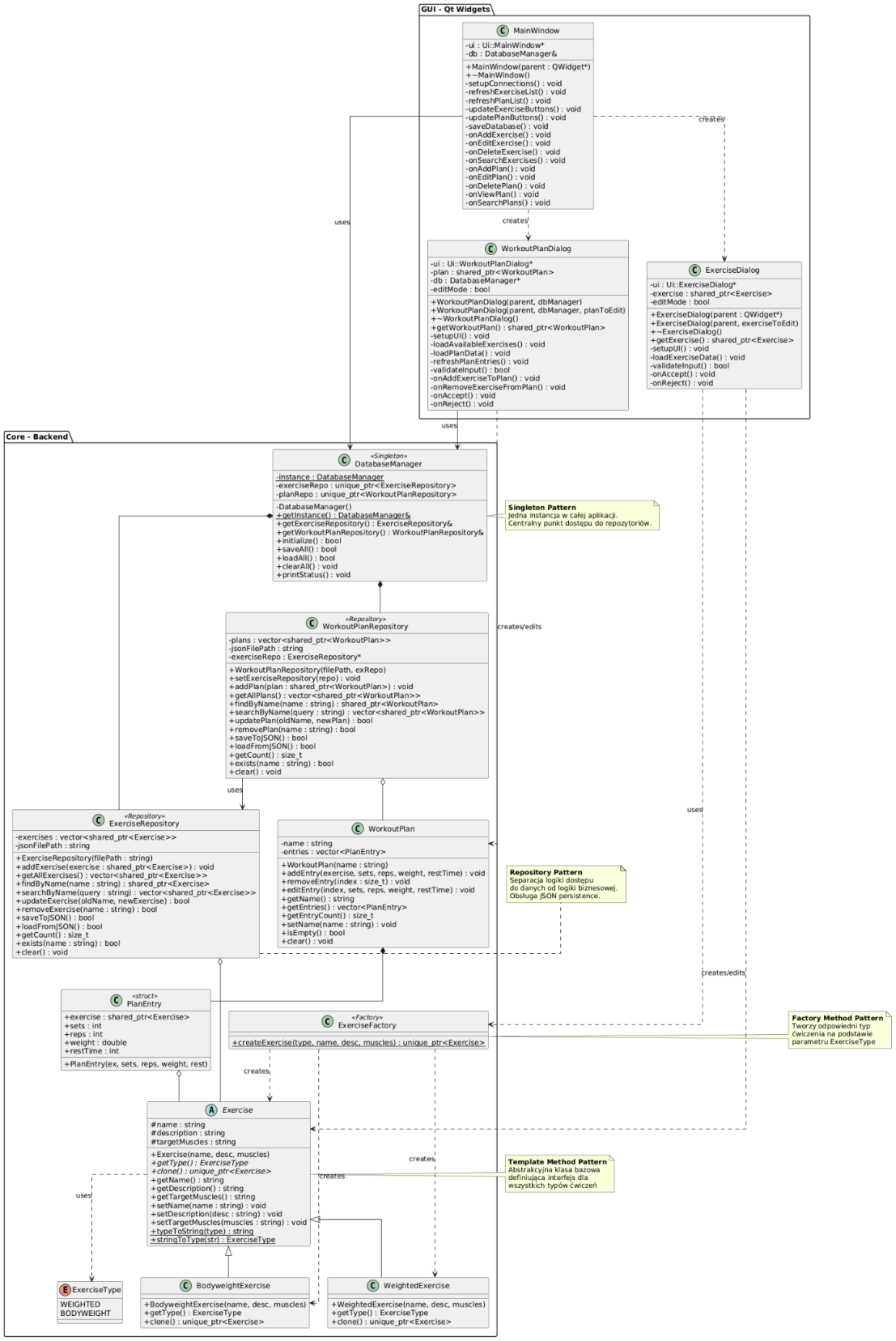
Kluczowe decyzje:

1. **Smart pointery (RAII):** unique_ptr (Factory), shared_ptr (współdzielenie) - automatyczne zarządzanie pamięcią
2. **Separacja Backend/GUI:** core/ niezależne od Qt, gui/ komunikuje przez DatabaseManager
3. **JSON:** Czytelny, łatwy w parsowaniu, wieloplatformowy
4. **Walidacja wielopoziomowa:** GUI, klasy, Repository

5. Projekt Techniczny

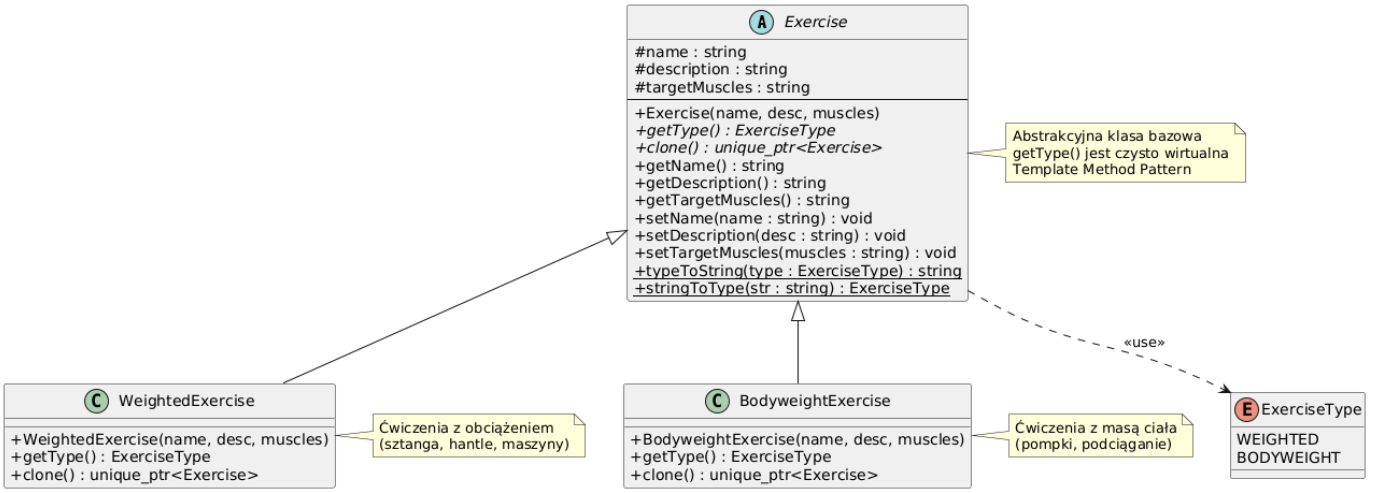
5.1 Diagram klas UML

Poniżej przedstawiono kompletny diagram klas systemu **PumpApp** w notacji UML. Diagram został wygenerowany za pomocą narzędzia PlantUML i prezentuje wszystkie kluczowe klasy wraz z ich atrybutami, metodami oraz relacjami.



Rysunek 5.1: Diagram klas UML systemu PumpApp

5.2 Hierarchia dziedziczenia - Exercise



Rysunek 5.2: Diagram klas UML systemu PumpApp

Klasa Exercise (abstrakcyjna):

- Atrybuty: name, description, targetMuscles
- Metody: virtual getType() = 0, clone(), gettery/settery
- Wzorzec: Template Method Pattern

WeightedExercise: getType() zwraca WEIGHTED
BodyweightExercise: getType() zwraca BODYWEIGHT

Dlaczego abstrakcyjna: Wymuszenie implementacji getType(), polimorfizm

5.3 Factory Method Pattern

ExerciseFactory:

- Metoda: static unique_ptr<Exercise> createExercise(type, name, desc, muscles)
- Switch po type → tworzy odpowiednią klasę
- Rzuca invalid_argument dla nieznanego typu

Zastosowanie: Dialogi, JSON deserializacja, testy

5.4 Repository Pattern

ExerciseRepository:

- Kolekcja: vector<shared_ptr<Exercise>>
- CRUD: add, getAll, findByName, search, update, remove
- Persistence: saveToJSON(), loadFromJSON()
- Pomocnicze: getCount(), exists(), clear()

WorkoutPlanRepository:

- Analogiczny do ExerciseRepository
- Referencja do ExerciseRepository (deserializacja planów)

5.5 Singleton Pattern - DatabaseManager

Odpowiedzialność: Centralne zarządzanie repozytoriami

Implementacja:

```
static DatabaseManager& getInstance() {
    static DatabaseManager instance;
    return instance;
}
private:
    DatabaseManager();
    DatabaseManager(const DatabaseManager&) = delete;
```

Metody: getExerciseRepository(), getWorkoutPlanRepository(), initialize(), saveAll(), loadAll()

5.6 Klasa WorkoutPlan

Struktura PlanEntry:

- exercise: shared_ptr<Exercise>

- sets, reps: int
- weight: double
- restTime: int

Dlaczego shared_ptr: Współdzielenie ćwiczenia przez wiele planów, RAII

WorkoutPlan:

- Atrybuty: name, vector<PlanEntry>
- Metody: addEntry(), removeEntry(), editEntry(), getEntries(), isEmpty()

5.7 Warstwa GUI (Qt Widgets)

MainWindow:

- ui: Ui::MainWindow*
- db: DatabaseManager&
- Metody: setupConnections(), refreshLists(), sloty dla akcji, saveDatabase()

ExerciseDialog:

- exercise: shared_ptr<Exercise>
- editMode: bool
- Metody: setupUI(), loadExerciseData(), validateInput(), onAccept()

WorkoutPlanDialog:

- plan: shared_ptr<WorkoutPlan>
- db: DatabaseManager*
- Metody: loadAvailableExercises(), onAddExerciseToPlan(), onRemoveFromPlan(), validateInput()

Sygnały i sloty Qt: connect(przycisk, &QPushButton::clicked, obiekt, &Class::metoda)

5.8 Typy relacji UML

Dziedziczenie: Exercise -> WeightedExercise (trójkąt, linia ciągła)
Kompozycja: DatabaseManager -> Repositories (czarny romb)
Agregacja: Repository -> Exercise (biały romb)
Zależność: Factory -> Exercise (linia przerywana)
Asocjacja: WorkoutPlanRepository -> ExerciseRepository (linia ciągła)

6. Opis realizacji

6.1 Platforma i narzędzia

System: Windows 11 (64-bit), kompatybilność Linux
IDE: Qt Creator 14.0.1 (edytor, Qt Designer, debugger GDB, wsparcie CMake)
Kompilator: MinGW 13.1.0 (GCC), standard C++17
Build: CMake 3.30 (wieloplatformowy, zarządzanie zależnościami)

6.2 Biblioteki

Qt Framework 6.10.1:

- Qt Widgets: QMainWindow, QDialog, QListWidget, QPushButton, QComboBox, QMessageBox, QInputDialog, sygnały/sloty
- Qt Core: QString (Unicode), QFile/QDir

STL:

- Kontenery: vector
- Smart pointery: unique_ptr, shared_ptr (RAII)
- Algorytmy: find_if, remove_if

Google Test 1.12.1:

- Framework testów jednostkowych
- Automatyczne pobieranie przez CMake (FetchContent)
- 14 testów: Factory, polimorfizm, WorkoutPlan, Repository

6.3 System kontroli wersji

Git + GitHub:

- Repozytorium
- Commitowanie zmian po każdym etapie
- README z instrukcją kompilacji

6.4 Uwagi o AI

Autor wykorzystał Claude.ai jako **przewodnika i nauczyciela** przy:

- Przygotowywaniu testów jednostkowych (GTest)
- Zrozumieniu wzorców projektowych
- Rozwiązywaniu problemów z Qt

Cały kod został **napisany samodzielnie** przez autora. AI służyło wyłącznie do nauki i konsultacji.

6. Opis realizacji

6.1 Platforma i narzędzia

System: Windows 11 (64-bit), kompatybilność z Linux
IDE: Qt Creator 14.0.1 (edytor, Qt Designer, debugger GDB, wsparcie CMake)
Kompilator: MinGW 13.1.0 (GCC), standard C++17
Build: CMake 3.30 (wieloplatformowy, zarządzanie zależnościami)

6.2 Biblioteki

Qt Framework 6.10.1:

- Qt Widgets: QMainWindow, QDialog, QListWidget, QPushButton, QComboBox, QMessageBox, QInputDialog, sygnały/sloty
- Qt Core: QString (Unicode), QFile/QDir

STL:

- Kontenery: vector
- Smart pointery: unique_ptr, shared_ptr (RAII)
- Algorytmy: find_if, remove_if

Google Test 1.12.1:

- Framework testów jednostkowych
- Automatyczne pobieranie przez CMake (FetchContent)
- 14 testów: Factory, polimorfizm, WorkoutPlan, Repository

6.3 System kontroli wersji

Git + GitHub:

- Repozytorium: https://github.com/JMikulski358/PumpApp

6.4 Uwagi o AI

Autor wykorzystał Claude.ai jako **przewodnika i nauczyciela** przy:

- Przygotowywaniu testów jednostkowych (GTest)
- Zrozumieniu wzorców projektowych
- Rozwiązywaniu problemów z Qt

Cały kod został **napisany samodzielnie** przez autora. AI służyło wyłącznie do nauki i konsultacji.

7. Opis wykonanych testów

Celem sprawdzenia poprawności działania stworzonego rozwiązania przeprowadzono szereg testów, które opisano w poniższych podrozdziałach.

7.1 Testy manualne - scenariusze testowe

Aplikacja została przetestowana manualnie według następujących scenariuszy:

ID	Funkcjonalność	Kroki testowe	Oczekiwany rezultat	Status
T01	Dodawanie ćwiczenia	1. Kliknij "Dodaj" w zakładce Ćwiczenia 2. Wypełnij formularz 3. Kliknij OK	Ćwiczenie pojawia się na liście	OK
T02	Edycja ćwiczenia	1. Zaznacz ćwiczenie 2. Kliknij "Edytuj" 3. Zmień dane 4. Kliknij OK	Dane zaktualizowane na liście	OK
T03	Usuwanie ćwiczenia	1. Zaznacz ćwiczenie 2. Kliknij "Usuń" 3. Potwierdź	Ćwiczenie znika z listy	OK
T04	Wyszukiwanie ćwiczeń	1. Wpisz fragment nazwy w pole search 2. Obserwuj listę	Lista filtrowana dynamicznie	OK

T05	Tworzenie planu	1. Kliknij "Dodaj" w zakładce Plany 2. Wprowadź nazwę 3. Dodaj 3 ćwiczenia 4. Kliknij OK	Plan z 3 ćwiczeniami na liście	OK
T06	Parametry treningowe	1. Dodaj ćwiczenie do planu 2. Ustaw: 4 serie, 8 reps, 80kg, 120s	Parametry wyświetlone w podglądzie	OK
T07	Podgląd planu	1. Zaznacz plan 2. Kliknij "Podgląd"	Wyświetlone szczegóły wszystkich ćwiczeń	OK
T08	Dane	1. Dodaj ćwiczenie 2. Zamknij aplikację 3. Uruchom ponownie	Ćwiczenie nadal na liście	OK
T09	Walidacja - puste pola	1. Spróbuj dodać ćwiczenie z pustą nazwą	Komunikat błędu, brak zapisu	OK
T10	Walidacja - duplikat	1. Dodaj ćwiczenie "Pompki" 2. Spróbuj dodać drugie "Pompki"	Komunikat błędu o duplikacie	OK

Tab. 7.1.1 Lista testów manualnych

Projekt autora przeszedł z powodzeniem wszystkie przeprowadzone testy manualne.

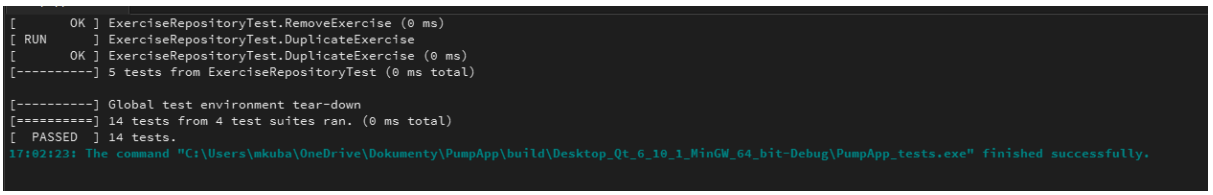
7.2 Testy automatyczne (Google Test)

Na potrzeby projektu przygotowano **14 testów jednostkowych** pokrywających kluczowe komponenty backendu. Do pomocy przy ich tworzeniu autor wykorzystał sztuczną inteligencją ***Claude.ai,*** traktując ją jako przewodnika i nauczyciela.

7.2.1 Pokrycie testami

Moduł	Liczba testów	Status
ExerciseFactory (Factory Pattern)	2	OK
Exercise (Polimorfizm)	2	OK
Workout Plan (CRUD)	5	OK
ExerciseRepository (CRUD + JSON)	5	OK
RAZEM	**14**	**ALL OK**

Tab. 7.2.1.1 Lista testów automatycznych



Zdj. 7.2.1.1 Rezultat wykonanych testów automatycznych

Jak widać z powyższych załączników projekt autora przeszedł pomyślnie wszystkie testy.

7.2.2 Przykładowe testy

Poniżej pokazane przykładowe testy automatyczne:

```
TEST(ExerciseFactoryTest, CreateWeightedExercise) {
    auto exercise = ExerciseFactory::createExercise(
        ExerciseType::WEIGHTED, "Bench Press", "...", "Chest"
    );
    EXPECT_EQ(exercise->getType(), ExerciseType::WEIGHTED);
}
```

Kod 7.2.2.1 Test Factory Pattern

```
TEST(ExerciseTest, PolymorphismWeighted) {
    std::shared_ptr<Exercise> exercise =
        std::make_shared<WeightedExercise>("Squat", "...", "Legs");
    EXPECT_EQ(exercise->getType(), ExerciseType::WEIGHTED);
}
```

Kod 7.2.2.2 Test polimorfizmu

7.3 Wykryte problemy i ich rozwiązania

W trakcie prac wykryto następujące problemy, których opis i rozwiązania zaprezentowana poniżej:

Problem 1: Conflict unique_ptr vs shared_ptr

Opis: Factory zwraca `unique_ptr`, a plan wymaga `shared_ptr`
Rozwiązanie: Konwersja poprzez `std::move()` w miejscu użycia

Problem 2: Brak walidacji liczby min w planie

Opis: Plan mógł być zapisany bez ćwiczeń
Rozwiązanie: Dodano walidację `plan->isEmpty()` w dialogu

Problem 3: Duplikaty nazw ćwiczeń

Opis: Możliwe było dodanie dwóch ćwiczeń o tej samej nazwie
Rozwiązanie: Repository sprawdza unikalność przed dodaniem

7.4 Podsumowanie testów

Projekt autora pomyślnie przeszedł wszystkie przygotowane testy, zachowując się zgodnie z oczekiwaniami.

8. Podręcznik użytkownika (User Manual)

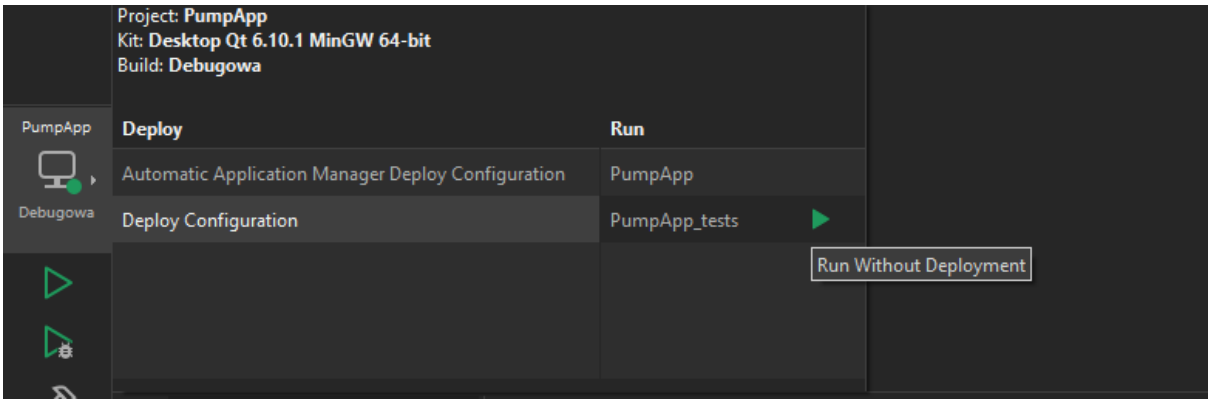
Niniejsza sekcja jest poświęcona użytkowaniu aplikacji stworzonej przez Autora. Autor ma nadzieję przedstawić czytelnikowi za jej pomocą intencję jej użytkowania, a także pokazać jej możliwości.

8.1 Uruchomienie aplikacji

Aplikację zaleca się uruchamiać przez Qt Creator:

1. Otwórz Qt Creator
2. File -> Open File or Project -> wybierz `CMakeLists.txt`
3. Skonfiguruj projekt (Desktop Qt 6.10.1 MinGW 64-bit)
4. Kliknij Run lub **Ctrl+R**

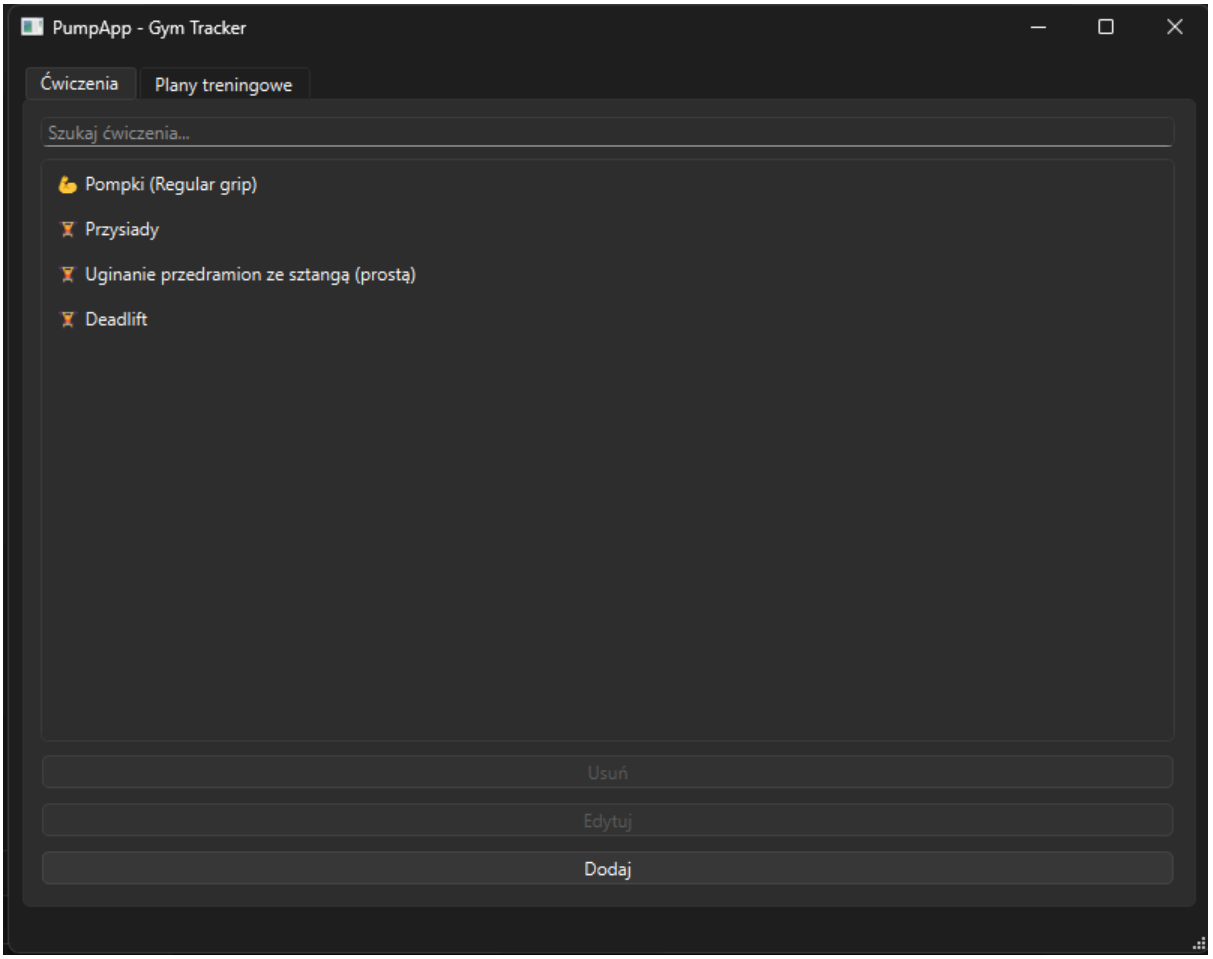
UWAGA: W razie problemów z uruchomieniem, należy wyłączyć testy aplikacji, gdyż mogą one powodować problemy z kompilacją. Opcjonalnie można uruchomić program przez wersję debugową.



Rysunek 8.1.1 - Sposób uruchomienia aplikacji w Qt

Bezpośrednio (Windows):

- Przejdź do folderu build/
- Uruchom PumpApp.exe



Rysunek 8.1.1 - Główne okno aplikacji z zakładkami Exercises i Workout Plans

8.2 Zarządzanie ćwiczeniami

Aplikacja PumpApp pozwala na zarządzanie bazą ćwiczeń na trzy następujące sposoby.

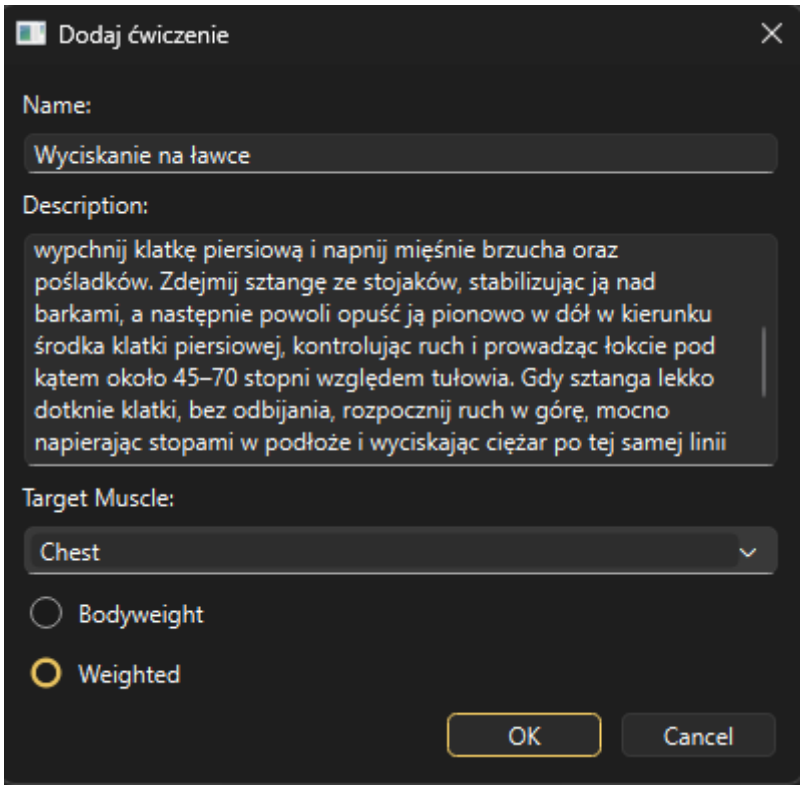
8.2.1 Dodawanie ćwiczenia

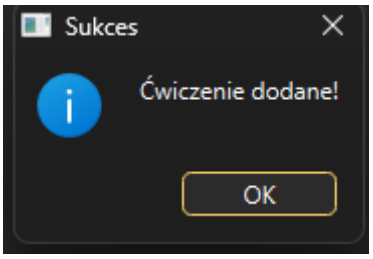
1. Zakładka **Exercises** -> przycisk **Add**
2. Wypełnij formularz:
    - **Name** - nazwa (np. "Bench Press")
    - **Description** - opis techniki
    - **Target Muscles** - lista lub własna wartość (np. "Chest, Triceps")
    - **Type** - Bodyweight, lub Weighted (Zaznacz)

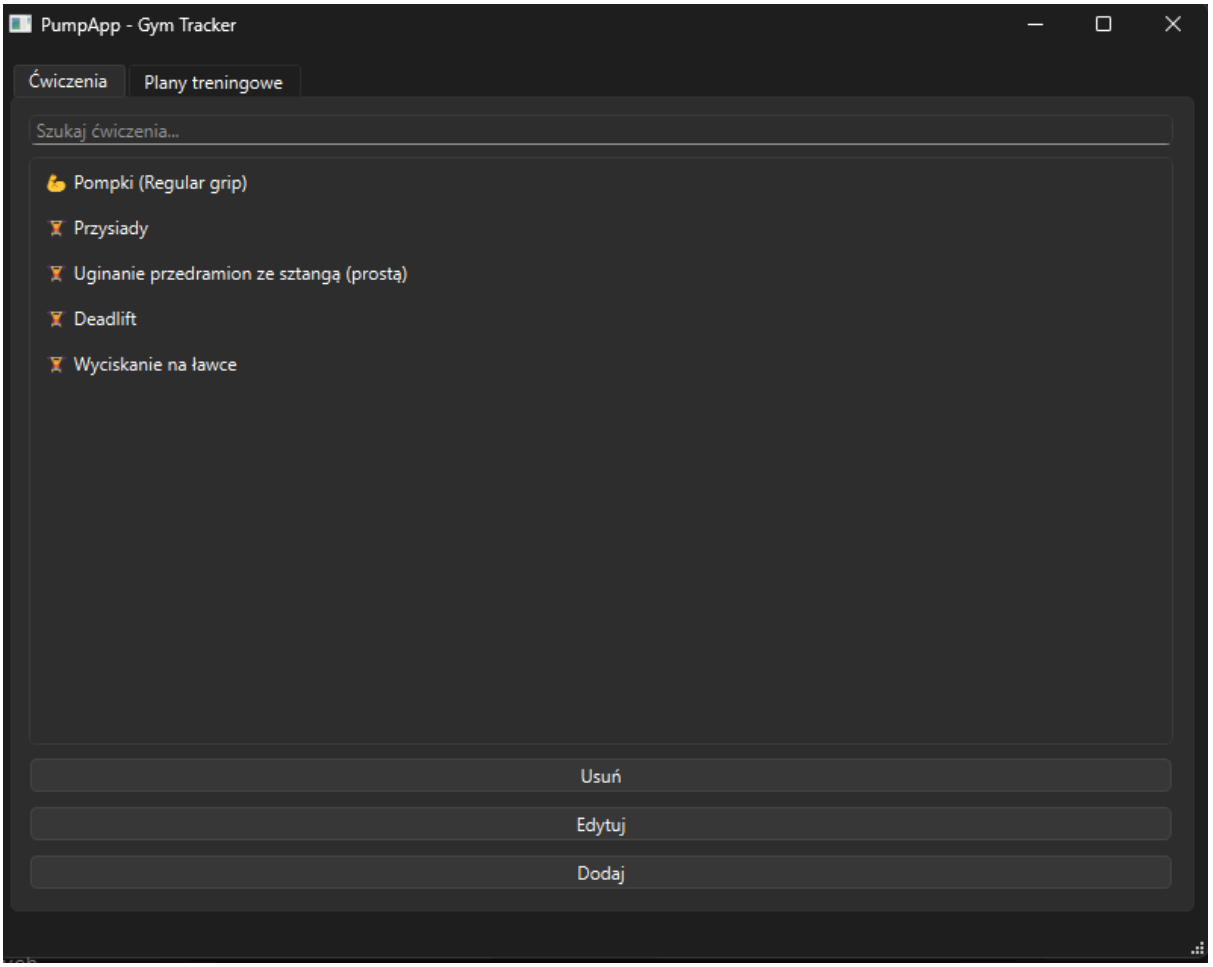
- **OK** - ćwiczenie pojawia się na liście

Walidacja: Aby utworzyć ćwiczenie wszystkie pola muszą zostać wypełnione, a nazwa musi być unikalna.

Proces dodawania przedstawiona na poniższych zdjęciach:







Rysunek 8.2.1.3 - Proces dodwania ćwiczenia: 1. Wypełniony formularz 2. Komunikat o powodzeniu 3. Nowe ćwiczenie na liście (typ (weighted/bodyweight) sygnalizowany ikonką)

8.2.2 Edycja

Zaznacz -> **Edit** -> zmień dane -> **OK**

8.2.3 Usuwanie

Zaznacz -> **Delete** -> potwierdź

UWAGA: Usunięcie ćwiczenia z bazy NIE usuwa go z istniejących planów.

8.2.4 Wyszukiwanie

Wpisz fragment nazwy w pole search -> lista filtruje się automatycznie

UWAGA: W wyszukiwaniu wielkość liter MA znaczenie.

8.3 Zarządzanie planami treningowymi

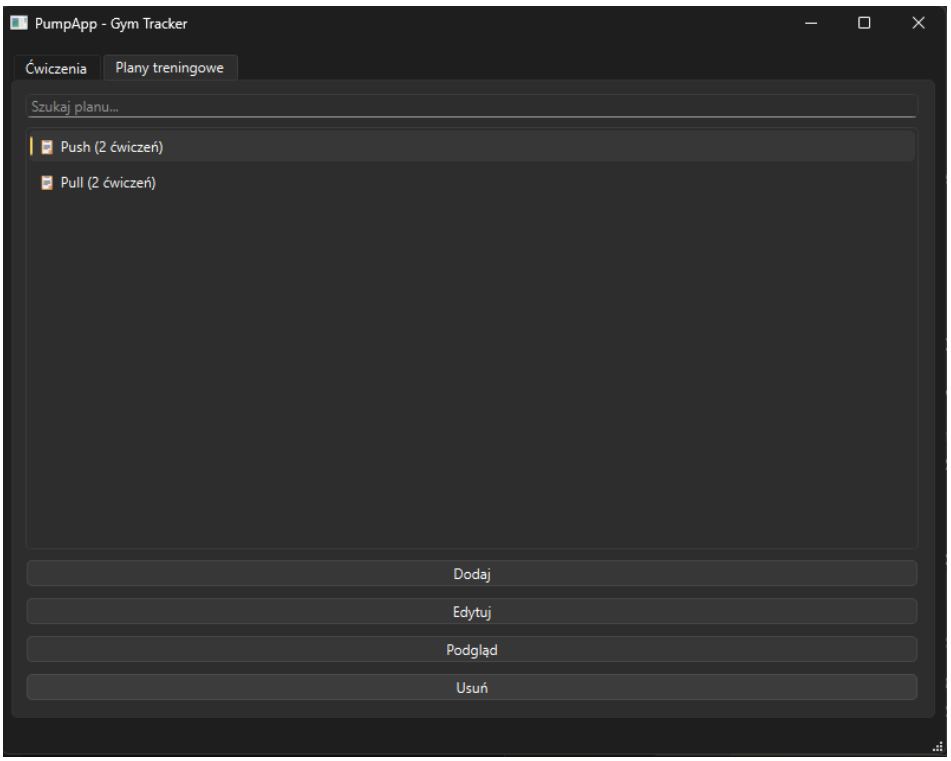
Na analogicznej zasadzie aplikacja pozwala na zarządzanie planami treningowymi.

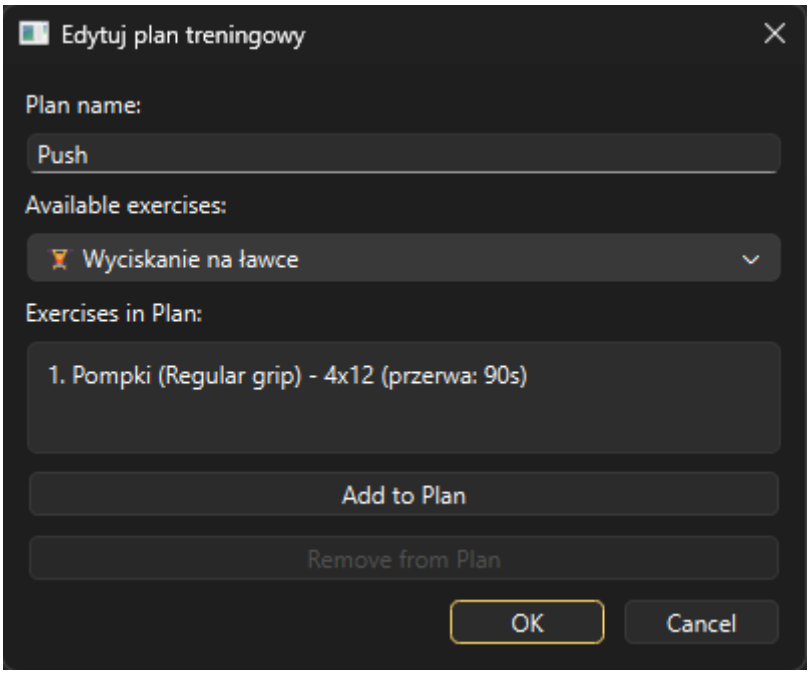
8.3.1 Tworzenie planu

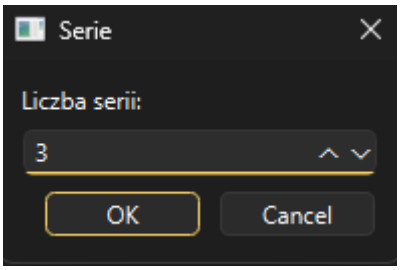
1. Zakładka **Workout Plans** -> **Add**
2. Wpisz nazwę planu (np. "Push Day")
3. Dodaj ćwiczenia:
    - Wybierz z listy -> **Add to Plan**
    - System zapyta o:
        - **Sets** (serie): 1-20
        - **Reps** (powtórzenia): 1-100
        - **Weight** (ciężar w kg): 0-500 (tylko Weighted)
        - **Rest** (odpoczynek w sekundach): 10-600
4. Powtórz dla kolejnych ćwiczeń
5. **OK** -> zapisz plan

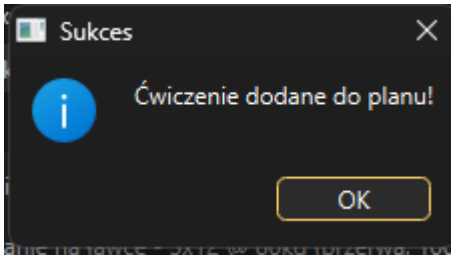
Walidacja: Plan musi mieć min. 1 ćwiczenie, a także unikalną nazwę.

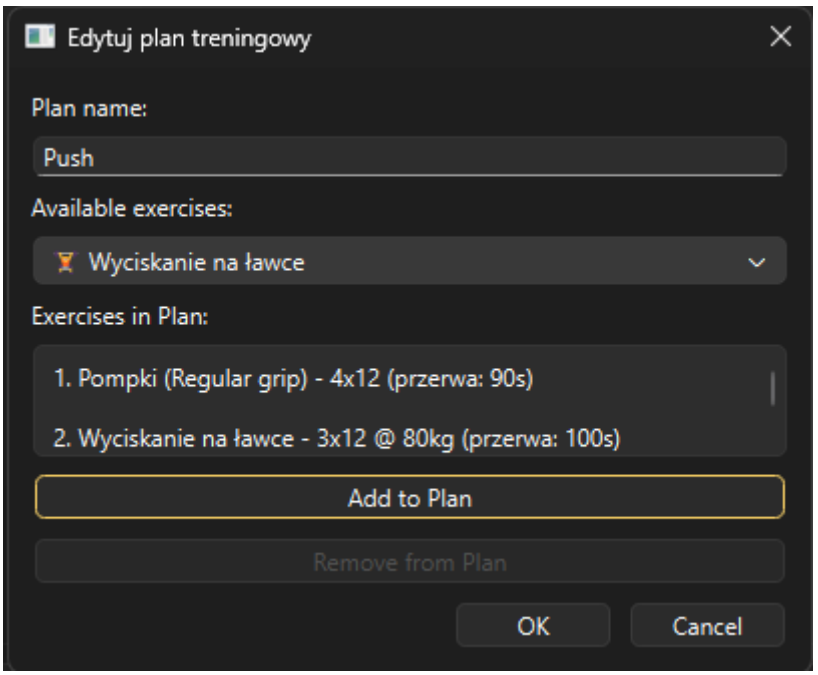
Proces dodawania planu przedstawiono na zdjęciach:


PumpApp - Gym Tracker
Ćwiczenia
Plany treningowe
Szukaj planu...
Push (2 ćwiczeń)
Pull (2 ćwiczeń)
Dodaj
Edytuj
Podgląd
Usuń


Edytuj plan treningowy
Plan name:
Push
Available exercises:
Wyciskanie na ławce
Exercises in Plan:
1. Pompki (Regular grip) - 4x12 (przerwa: 90s)
Add to Plan
Remove from Plan
OK
Cancel


Serie
Liczba serii:
3
OK
Cancel


Sukces
Ćwiczenie dodane do planu!
OK



Rysunek 8.3.1.1 - Proces dodwania planu: 1. Zakładka Workout Plan 2. Wybór ćwiczenia do planu 3. Formularz dodawania ćwiczenia (liczba serii) 4. Ćwiczenie w planie

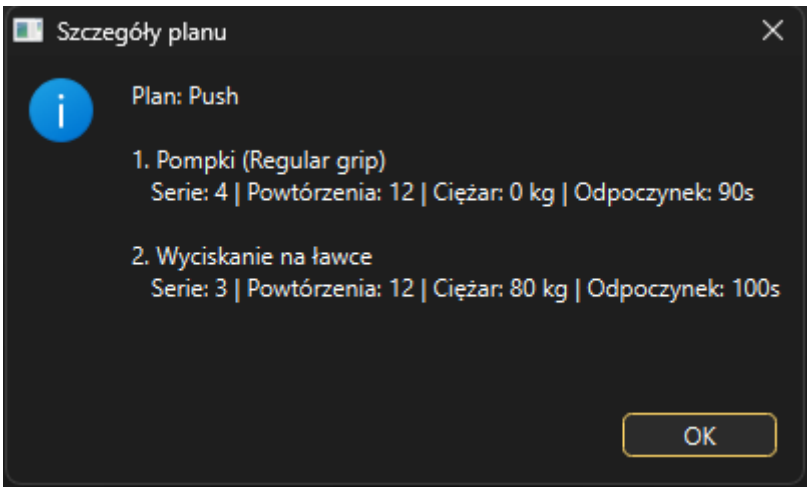
8.3.2 Edycja planu

Zaznacz -> **Edit** -> zmień nazwę / dodaj / usuń ćwiczenia -> **OK**

Brak edycji parametrów: Aby zmienić sets/reps/weight - usuń ćwiczenie i dodaj ponownie.

8.3.3 Podgląd

Zaznacz -> **View** -> okno ze szczegółami (wszystkie ćwiczenia + parametry)



8.3.3.1Podgląd planu w opcji view

8.3.4 Usuwanie

Zaznacz -> **Delete** -> potwierdź (nieodwracalne!)

8.3.5 Wyszukiwanie

Jak w ćwiczeniach - wpisz fragment nazwy.

8.4 Dane

Niniejsza sekcja została poświęcona przechowywaniu danych przez aplikacje.

8.4.1 Automatyczny zapis

System zapisuje dane automatycznie po każdej operacji do:

- `data/exercises.json`
- `data/plans.json`

8.4.2 Backup

1. Zamknij aplikację
2. Skopiuj folder `data/` w bezpieczne miejsce
3. Przywrócenie: zastąp folder kopią

8.4.3 Transfer danych

Skopiuj `data/` na pendrive -> wklej na drugim komputerze.

8.5 Rozwiązywanie potencjalnych problemów

Błąd "Qt6Core.dll not found":

- **Najlepsze:** Uruchom przez Qt Creator
- **Alternatywa:** Qt Command Prompt -> `windeployqt PumpApp.exe`

Dane zniknęły:

- Sprawdź folder `data/`
- Otwórz JSON w Notatniku - jeśli uszkodzony, usuń (aplikacja stworzy nowy)
- Przywróć z backupu

Duplikat ćwiczenia:

- Nazwa już istnieje - użyj search lub zmień nazwę

Pusty plan po zapisie:

- Musisz dodać min. 1 ćwiczenie przed OK

Aplikacja się zawiesza:

- Sprawdź rozmiar JSON (<100MB)
- Zamknij przez Task Manager
- Zgłoś na GitHub Issues

9. Podsumowanie i wnioski

Projekt **PumpApp** został zrealizowany zgodnie z założeniami przedmiotu Język Programowania Obiektowego C++. System stanowi funkcjonalną aplikację desktopową do zarządzania treningami siłowymi, łączącą solidny backend oparty na paradygmacie OOP z przejrzystym interfejsem graficznym Qt Widgets.

9.1 Osiągnięte cele

Wszystkie założone cele zostały zrealizowane:

1. **Funkcjonalność CRUD** - pełne zarządzanie ćwiczeniami i planami treningowymi
2. **Wzorce projektowe** - implementacja Factory Method, Repository, Singleton, hierarchia dziedziczenia
3. **Separacja warstw** - backend niezależny od GUI, możliwość łatwejmigracji interfejsu
4. **Trwałe przechowywanie danych** - automatyczny zapis do JSON po każdej operacji
5. **Walidacja danych** - wielopoziomowa kontrola poprawności na poziomie GUI, logiki i repozytorium
6. **Testy automatyczne** - 14 testów jednostkowych (Google Test), wszystkie zakończone sukcesem
7. **Wieloplatformowość** - aplikacja kompiluje się na Windows i Linux dzięki Qt + CMake

9.2 Napotkane wyzwania i ich rozwiązania

W trakcie realizacji projektu autor napotkał następujące wyzwania:

1. Konflikt typów smart pointerów

Factory zwracało `unique_ptr<Exercise>`, podczas gdy plan treningowy wymaga `shared_ptr<Exercise>` (ćwiczenie może być współdzielone przez wiele planów). Rozwiązano ten problem poprzez konwersję `std::move()` w miejscu dodawania ćwiczenia do planu.

2. Deserializacja planów z JSON

Plan treningowy przechowuje tylko **nazwę ćwiczenia** w JSON, nie cały obiekt. Przy odczycie konieczne było odnalezienie odpowiedniego obiektu w repozytorium ćwiczeń - stąd `WorkoutPlanRepository` wymaga referencji do `ExerciseRepository`.

3. Walidacja minimalnej zawartości planu

Początkowo możliwe było zapisanie pustego planu treningowego. Dodano walidację `plan->isEmpty()` w `WorkoutPlanDialog::validateInput()`.

4. Duplikaty nazw
System początkowo pozwalał na dodanie dwóch ćwiczeń o identycznej nazwie. Rozwiązano dodając metodę `Repository::exists()` sprawdzającą unikalność przed dodaniem.

9.3 Możliwości rozwoju

System PumpApp stanowi solidną bazę do dalszej rozbudowy. Możliwe kierunki rozwoju:

Statystyki i analiza:

- Historia wykonanych treningów z datami
- Wykresy postępów (zwiększenie obciążenia, liczby powtórzeń)
- Analiza trendów (np. które partie mięśniowe są mogą być zaniedbane)

Rozszerzenie typów ćwiczeń:

- `IsometricExercise` - ćwiczenia izometryczne (plank, martwy zwis)
- `CardioExercise` - bieżnia, rower, wioślarz (z czasem i spalaniem kalorii)
- `StretchingExercise` - ćwiczenia rozciągające

Integracja z urządzeniami:

- Import danych z smartwatchy / opasek fitness
- Eksport planów do aplikacji mobilnych

Funkcje społecznościowe:

- Udostępnianie planów treningowych między użytkownikami
- Baza danych online (MySQL/PostgreSQL zamiast JSON)

System nagród:

- System osiągnięć i odznak
- Streak - licznik kolejnych dni treningowych

9.4 Wnioski końcowe

Realizacja projektu **PumpApp** pozwoliła autorowi na praktyczne zastosowanie paradygmatu programowania obiektowego oraz poznanie wzorców projektowych stosowanych w rzeczywistych systemach. Kluczowe lekcje wyniesione z projektu:

1. **Design patterns rozwiązują konkretne problemy** - Factory eliminuje uzależnienie od klas konkretnych, Repository oddziela logikę od persistence, Singleton gwarantuje spójność danych.
2. **Separacja warstw zwiększa elastyczność** - backend niezależny od GUI można łatwo przetestować, przenieść do innego interfejsu (web app, CLI) lub zintegrować z innymi systemami.
3. **Smart pointery eliminują memory leaks** - konsekwentne stosowanie `unique_ptr` i `shared_ptr` (RAII) zapewnia automatyczne zarządzanie pamięcią bez ryzyka wycieków.
4. **Walidacja na wielu poziomach zwiększa niezawodność** - GUI, logika i warstwa danych powinny niezależnie sprawdzać poprawność, tworząc system obronny przed błędami.
5. **Testy automatyczne oszczędzają czas** - inwestycja w Google Test zwróciła się przy refaktoringu - autor mógł śmiało zmieniać kod backendu, mając pewność że nie zepsuł istniejącej funkcjonalności.

Projekt **PumpApp** spełnił oczekiwania jako narzędzie edukacyjne oraz jako potencjalnie użyteczna aplikacja dla osób trenujących na siłowni.

10. Bibliografia

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

[7] Google Test Documentation: *GoogleTest Primer*. https://google.github.io/googletest/primer.html

[8] CMake Documentation: *CMake Tutorial*. https://cmake.org/cmake/help/latest/guide/tutorial/index.html

[9] cppreference.com: *C++ Standard Library Reference*. https://en.cppreference.com