**STRESZCZENIE**

Naszym zadaniem w ramach projektu dyplomowego było zrealizowanie aplikacji pozwalającej na generowanie testów egzaminacyjnych z języka C++. Powyższy program ułatwia tworzenie arkuszy egzaminacyjnych dzięki możliwości skompilowania i wykonania kodu dołączonego do zadań oraz wygenerowania poprawnych odpowiedzi widocznych w arkuszu dla nauczyciela. Projekt zrealizowaliśmy w formie aplikacji desktopowej na systemy operacyjne Linux i Windows.

W trakcie pracy nad projektem zapoznaliśmy się ze zbliżonymi rozwiązaniami istniejącymi na rynku, stworzyliśmy specyfikację wymagań, opracowaliśmy model architektury systemu oraz model klas. Wybraliśmy narzędzia i technologie, które następnie wykorzystaliśmy do implementacji rozwiązania. Realizację projektu zakończyliśmy utworzeniem dokumentacji wraz z instrukcją użytkownika.

Podział zadań w trakcie implementacji systemu nie był ścisły, a członkowie zespołu mieli swój udział w rozwijaniu wszystkich funkcjonalności, jednak ogólny rozkład odpowiedzialności za komponenty systemu prezentował się następująco:

* Natalia Kowalik - interfejs graficzny
* Paweł Lal - logika interfejsu (obsługa zachowań użytkownika)
* Martyna Łuczkowska - generowanie plików PDF (treść i układ arkusza)
* Marta Mazur - logika aplikacji (obsługa różnych typów zadań)

Udział poszczególnych członków zespołu w tworzeniu rozdziałów dokumentacji przedstawia się następująco:

* Natalia Kowalik – rozdziały 1 i 5
* Paweł Lal – rozdziały 3 i 8
* Martyna Łuczkowska – rozdziały 2 i 7
* Marta Mazur – rozdziały 4 i 6

**Słowa kluczowe:** aplikacja desktopowa,arkusz egzaminacyjny, generowanie testu, zadania

**Dziedzina nauki i techniki zgodnie z wymogami OECD:** Nauki o komputerach i informatyka, Pedagogika

**ABSTRACT**

Within the project our task was to implement an application, which allows the user to generate C++ exam tests. This program facilitates the creation of the examination papers with the ability to compile and execute the source code attached to the tasks and generate the correct answers, which are visible in the teacher’s version. The project is realized as a desktop application running on Windows and Linux operational systems.

While working on the project we found several similar solutions on the market, we created a requirements specification and also developed a system architecture model and a class model. We had chosen the tools and technologies, which we were using to implement the project. At the end we created documentation with user manual.

The division of tasks during the implementation of the system was not very strict, and members of the team had contributed to the development all of the functionalities. The overall division of the responsibility for system components:

* Natalia Kowalik – graphical user interface
* Paweł Lal – interface logic (handling users behavior)
* Martyna Łuczkowska – generation of PDF files (the content and layout of the test)
* Marta Mazur – application logic (handling various types of tasks)

The team members contribution in creating documentation sections as follows:

* Natalia Kowalik – chapters 1, 5
* Paweł Lal – chapters 3, 8
* Martyna Łuczkowska – chapters 2, 7
* Marta Mazur – chapters 4, 6

**Keywords:** desktop application,examination paper, test generation, tasks

**Field of science and technology:** Computer and information sciences, Educational sciences

**SPIS TREŚCI**

[1. WSTĘP I CEL PRACY (Natalia Kowalik) 8](#_Toc468937702)

[1.1 Przybliżenie tematu 8](#_Toc468937703)

[1.2 Cel projektu 8](#_Toc468937704)

[1.3 Określenie problemu 9](#_Toc468937705)

[1.4 Użytkownicy 9](#_Toc468937706)

[1.5 Cele funkcjonalne 9](#_Toc468937707)

[1.6 Ograniczenia i założenia 10](#_Toc468937708)

[1.7 Koncepcja rozwiązania 10](#_Toc468937709)

[2. ISTNIEJĄCE ROZWIĄZANIA (Martyna Łuczkowska) 11](#_Toc468937710)

[2.1. Online C Programming Test IndiaBIX™ Technologies 11](#_Toc468937711)

[2.2. Kompilator CppDroid – C/C++ IDE 12](#_Toc468937712)

[2.3. Kompilator i edytor tekstu 13](#_Toc468937713)

[2.4. Podsumowanie 14](#_Toc468937714)

[3. KONCEPCJA ROZWIĄZANIA (Paweł Lal) 16](#_Toc468937715)

[3.1 Wymagania 16](#_Toc468937716)

[3.1.1 Wymagania funkcjonalne 16](#_Toc468937717)

[3.1.2 Wymagania na dane 16](#_Toc468937718)

[3.1.3 Wymagania jakościowe 17](#_Toc468937719)

[3.2 Model przypadków użycia 17](#_Toc468937720)

[3.2.1 Skrócony diagram przypadków użycia 17](#_Toc468937721)

[3.2.2 Skrócony opis przypadków użycia 19](#_Toc468937722)

[4. PROJEKT (Marta Mazur) 25](#_Toc468937723)

[4.1. Projekt architektury systemu 25](#_Toc468937724)

[4.1.1. Koncepcja systemu 25](#_Toc468937725)

[4.1.2. Struktura systemu 25](#_Toc468937726)

[4.1.3. Warstwy architektoniczne 26](#_Toc468937727)

[4.1.4. Komponenty programowe 27](#_Toc468937728)

[4.2. Projekt logiki aplikacji 28](#_Toc468937729)

[4.2.1. Koncepcja modelu klas 28](#_Toc468937730)

[4.2.2. Diagram klas 28](#_Toc468937731)

[4.3. Projekt interfejsu użytkownika 32](#_Toc468937732)

[4.3.1. Schemat układu okna głównego 32](#_Toc468937733)

[4.3.2. Formularze 33](#_Toc468937734)

[4.4. Projekt struktury danych 34](#_Toc468937735)

[4.4.1. Koncepcja struktury danych 34](#_Toc468937736)

[4.4.2. Pliki danych 34](#_Toc468937737)

[5. IMPLEMENTACJA (Natalia Kowalik) 36](#_Toc468937738)

[5.1 Wybór technologii 36](#_Toc468937739)

[5.2 Zastosowane technologie i biblioteki 36](#_Toc468937740)

[5.2.1 JavaFX 36](#_Toc468937741)

[5.2.2 JAXB 37](#_Toc468937742)

[5.2.3 Apache PDFBox 37](#_Toc468937743)

[5.2.4 Kompilator 37](#_Toc468937744)

[5.2.5 Apache Maven 37](#_Toc468937745)

[5.2.6 RichTextFX 38](#_Toc468937746)

[5.2.7 CSS 38](#_Toc468937747)

[5.3 Środowisko wytwórcze 38](#_Toc468937748)

[5.3.1 System kontroli wersji 38](#_Toc468937749)

[5.3.2 Zintegrowane środowisko programistyczne 40](#_Toc468937750)

[5.3.3 Tworzenie diagramów 40](#_Toc468937751)

[5.3.4 Narzędzia wspierające komunikację 40](#_Toc468937752)

[5.3.5 Narzędzia wspierające dokumentację 40](#_Toc468937753)

[5.3.6 Narzędzia wspomagające organizację projektu 40](#_Toc468937754)

[5.3.7 Narzędzia wspomagające testowanie 41](#_Toc468937755)

[5.4 Struktura kodu 41](#_Toc468937756)

[6. INSTRUKCJA UŻYTKOWNIKA (Marta Mazur) 44](#_Toc468937757)

[6.1 Instalacja i uruchomienie w systemie Windows 44](#_Toc468937758)

[6.2 Instalacja i uruchomienie w systemie Linux 44](#_Toc468937759)

[6.3. Zarządzanie zadaniami 44](#_Toc468937760)

[6.3.1. Tworzenie zadania 44](#_Toc468937761)

[6.3.2. Edycja zadania 45](#_Toc468937762)

[6.4. Dostępne typy zadań 45](#_Toc468937763)

[6.4.1. Wyjście programu 45](#_Toc468937764)

[6.4.2. Wyjście funkcji 47](#_Toc468937765)

[6.4.3. Wartość zmiennej 48](#_Toc468937766)

[6.4.4. Zwrócona wartość 49](#_Toc468937767)

[6.4.5. Numery linii 50](#_Toc468937768)

[6.4.6. Uzupełnianie luk 51](#_Toc468937769)

[6.4.7. Typ własny 53](#_Toc468937770)

[6.5. Markery 53](#_Toc468937771)

[6.6. Okno z odpowiedziami 54](#_Toc468937772)

[6.7. Wykonanie i wykonanie testowe 54](#_Toc468937773)

[6.8. Zarządzanie egzaminem 54](#_Toc468937774)

[6.8.1. Menu główne 54](#_Toc468937775)

[6.8.2. Menu kontekstowe 55](#_Toc468937776)

[6.9. Generowanie PDF 55](#_Toc468937777)

[6.9.1. Ustawienia arkusza 55](#_Toc468937778)

[6.9.2. Okno kompilacji egzaminu 55](#_Toc468937779)

[6.9.3. Warunki poprawnej generacji egzaminu 56](#_Toc468937780)

[6.10. Opcje zaawansowane 56](#_Toc468937781)

[6.10.1. Wygląd PDF 56](#_Toc468937782)

[6.10.2. Dodatkowe opcje PDF 56](#_Toc468937783)

[6.10.3. Limit czasu wykonania 57](#_Toc468937784)

[6.10.4. Polecenia domyślne 57](#_Toc468937785)

[7. TESTY (Martyna Łuczkowska) 58](#_Toc468937786)

[7.1. Testy precyzujące wymagania 58](#_Toc468937787)

[7.2. Testy wstępne dotyczące działania aplikacji na różnych systemach operacyjnych 59](#_Toc468937788)

[7.3. Testy bieżące (testowanie alfa) 60](#_Toc468937789)

[7.4. Testy akceptacyjne (testowanie beta) 64](#_Toc468937790)

[7.5. Podsumowanie 64](#_Toc468937791)

[8. PODSUMOWANIE I WNIOSKI (Paweł Lal) 65](#_Toc468937792)

[8.1. Zrealizowana funkcjonalność 65](#_Toc468937793)

[8.2. Niezrealizowana funkcjonalność 66](#_Toc468937794)

[8.2.1 Arkusz interaktywny 66](#_Toc468937795)

[8.2.2 Zapis arkusza w formacie edytowalnym 66](#_Toc468937796)

[8.3. Napotkane trudności 67](#_Toc468937797)

[8.3.1. Zmiana kolejności zadań 67](#_Toc468937798)

[8.3.2. Ubogość biblioteki pdfBox 67](#_Toc468937799)

[8.3.3. Narzędzie do projektowania interfejsu 67](#_Toc468937800)

[8.3.4. Powolne tworzenie się procesów w systemie Windows 67](#_Toc468937801)

[8.3.5. Brak testów jednostkowych i integracyjnych 67](#_Toc468937802)

[8.4. Zadania na przyszłość 68](#_Toc468937803)

[8.4.1. Umożliwienie zapisu arkusza w formacie edytowalnym 68](#_Toc468937804)

[8.4.2. Umożliwienie generacji arkusza interaktywnego 68](#_Toc468937805)

[8.4.3. Obsługa błędów wykonania dla dostarczonego kodu 68](#_Toc468937806)

[8.4.4. Pokrycie kluczowych funkcjonalności testami 68](#_Toc468937807)

[8.4.5. Proste podpowiedzi i kolorowanie składni 69](#_Toc468937808)

[8.5. Podsumowanie 69](#_Toc468937809)

[WYKAZ LITERATURY 70](#_Toc468937810)

[WYKAZ RYSUNKÓW 71](#_Toc468937811)

[WYKAZ TABEL 72](#_Toc468937812)

[DODATEK A: Diagram przypadków użycia 74](#_Toc468937813)

[DODATEK B: Opis przypadków użycia 75](#_Toc468937814)

# 1. WSTĘP I CEL PRACY

## 1.1 Przybliżenie tematu

Przedmiot Podstawy Programowania, realizowany przez nauczycieli akademickich z Katedry Algorytmów i Modelowania Systemów, jest dla wielu studentów pierwszym przedmiotem, który wprowadza elementarną wiedzę z zakresu programowania strukturalnego. W ramach zajęć studenci są zaznajamiani z najprostszymi strukturami danych i algorytmami. W tym celu wykorzystuje się przykłady zaimplementowane w języku C++, tym samym ucząc podstaw tego języka.

        W trakcie realizacji przedmiotu student zdobywa wiedzę na temat rozwiązywania problemów poprzez tworzenie algorytmów, jak również zasady pisania poprawnego i przejrzystego kodu. Dowiaduje się też o historii i zastosowaniach języka C i C++.

        Po zakończeniu zajęć z przedmiotu Podstawy Programowania student powinien nabyć umiejętność tworzenia prostych programów w języku C++, a także znać najważniejsze elementy języka, takie jak: operatory, operacje, funkcje, tablice, pętle i wskaźniki. Znajomość zakresu materiału może zostać sprawdzona za pomocą egzaminu, w którym każde z zadań posiada załączony kod źródłowy w języku C++ [1].

        Wiedza studenta jest weryfikowana za pomocą różnych typów zadań. Wyodrębnionych zostało sześć kategorii, a dodatkowo prowadzący przedmiot ma możliwość dodania nowej. W ostatnim przypadku ręcznie dodaje składowe zadania: treść polecenia, kod źródłowy, odpowiedź, i inne.

Domyślne typy zadań:

* egzaminowany musi podać, co zostanie wypisane przez program w wyniku jego wykonania,
* egzaminowany musi podać, jaka wartość zostanie zwrócona przez wskazane wywołania funkcji,
* egzaminowany musi podać, co zostanie wypisane na ekran w wyniku wywołania funkcji,
* egzaminowany musi uzupełnić luki tak, aby program został poprawnie skompilowany,
* egzaminowany musi podać wartości wskazanych zmiennych po zakończeniu realizacji danej funkcji,
* egzaminowany musi podać numery linii, które powodują błąd kompilacji programu.

## 1.2 Cel projektu

Celem projektu inżynierskiego jest stworzenie oprogramowania, które umożliwi automatyczne generowanie testów egzaminacyjnych ze znajomości języka C++. Dodatkowym założeniem jest tworzenie szablonu z poprawnymi odpowiedziami do zadań. Ma wyeliminować błędy występujące podczas tworzenia egzaminów, poprzez sprawdzenie poprawności kodów i kompletności zadań. Powinien również przyspieszyć tworzenie arkuszy egzaminacyjnych dzięki braku konieczności korzystania z kilku aplikacji, automatycznej kompilacji i automatycznego generowania odpowiedzi. Ma także uprościć sposób tworzenia egzaminów, dostarczając gotowe typy zadań.

        System taki powinien być przenośny pomiędzy systemami operacyjnymi Windows i Linux. Powinien także zapewniać możliwość tworzenia pytań o z góry zdefiniowanych typach, ale również pozwalać na dodanie pytania własnego typu.

        Osoba opracowująca arkusz egzaminacyjny dostarcza kod w języku C++ oraz pewne dodatkowe informacje dla każdego z zadań, takie jak treść polecenia, liczbę odpowiedzi, limit czasu wykonania kodu, czy wygląd samego arkusza.

## 1.3 Określenie problemu

Adresowanym problemem są błędy w trakcie tworzenia egzaminów z przedmiotu Podstawy Programowania spowodowane koniecznością kopiowania kodu źródłowego i ręcznego tworzenia różnych wersji testu egzaminacyjnego.

        Dotychczasowe tworzenie testów wymagało przygotowania osobno arkusza egzaminacyjnego – dla studenta, i arkusza odpowiedzi – dla nauczyciela. Ponadto egzaminy tworzone są w kilku wariantach, aby studenci rozwiązywali różniące się od siebie zadania. Powyższe okoliczności narzucają przechowywanie tego samego fragmentu kodu w wielu miejscach (na przykład plikach). Każda zmiana w kodzie wymaga aktualizacji wszystkich wariantów egzaminu i arkuszy odpowiedzi. Przeoczenie powoduje niekiedy występowanie rozbieżności lub błędów.

        Kolejnym problemem jest uciążliwa konieczność upewniania się po każdej wprowadzonej zmianie w kodzie, czy jest on poprawny – to znaczy, czy nie komunikuje błędów podczas kompilacji i ewentualnie, czy wynik wykonania programu jest prawidłowy.

## 1.4 Użytkownicy

Aplikacja będzie używana przez osoby prowadzące przedmiot, na którym studenci zapoznają się z podstawami programowania w języku C++.

        Potrzebami użytkowników programu są: tworzenie testów egzaminacyjnych na podstawie dostarczonego kodu źródłowego w języku C++, automatyczne generowanie odpowiedzi do zadań, sprawdzenie, czy dołączony kod zostaje skompilowany bez błędów i wyświetlenie wyniku jego wykonania. Ponadto wymagane było zaimplementowanie dodatkowych funkcjonalności, jakimi są zapisywanie i odczytywanie wcześniej utworzonych zadań i arkuszy, test egzaminacyjny generowany w formacie gotowym do druku, a także synchronizacja treści arkusza w wersji dla studenta i nauczyciela dla tych samych danych wejściowych.

        Zadaniem użytkownika jest dostarczenie kodu źródłowego do każdego zadania, a także dołączenie poleceń do zadań, kiedy domyślne polecenie będzie niewystarczające.

## 1.5 Cele funkcjonalne

Użytkownik powinien mieć możliwość wygenerowania arkuszy w wersjach dla studenta i nauczyciela na podstawie tych samych danych. Oznacza to, że dla niezmiennych danych wejściowych: jednakowych typów zadań, treści poleceń i kodu źródłowego, można wygenerować oba warianty testu.

        Przed stworzeniem testu treść zadania musi być automatycznie sprawdzana pod kątem kompletności tj. obecności w nim niepustego polecenia oraz kodu źródłowego. Kod weryfikowany jest pod kątem kompilowania się, a więc użytkownik dostaje informację, czy program powoduje błędy kompilacji.

        Wygenerowany arkusz egzaminacyjny musi posiadać miejsca na odpowiedzi studenta oraz jego dane. Powinien też mieć pola, do których zostanie wprowadzony tytuł egzaminu i zbiorczy komentarz. Wygenerowany arkusz egzaminacyjny w wersji dla nauczyciela musi zawierać odpowiedzi do zadań.

        Po wykonaniu programu automatycznie powinien zostać wypisany wynik działania programu. Użytkownik musi mieć sposobność do zadecydowania, czy wynik ten będzie odpowiedzią do zadania. W przeciwnym razie powinien móc ręcznie wprowadzić odpowiedź.

        System musi umożliwiać wykorzystanie wcześniej utworzonych zadań i arkuszy poprzez udostępnienie opcji zapisu i odczytu poszczególnych zadań oraz arkusza do/z pliku w formacie XML.

        Użytkownik podczas tworzenia zadania musi mieć możliwość skorzystania ze z góry zdefiniowanych szablonów zadań.

Student powinien mieć możliwość sprawdzenia swoich umiejętności poprzez rozwiązanie testu interaktywnego, zbliżonego do testu egzaminacyjnego, który sprawdzi poprawność udzielonych odpowiedzi.

## 1.6 Ograniczenia i założenia

Najważniejszym z założeń w naszym projekcie jest zrealizowanie go w taki sposób, by dostarczał jednakowe funkcjonalności na systemach operacyjnych Windows i Linux, włączając wykorzystanie zewnętrznego kompilatora.  Ograniczeniami są: konieczność korzystania ze wspieranych systemów operacyjnych, dostarczanie kodu napisanego wyłącznie w językach C lub C++ oraz odpowiedzialność za wprowadzenie poprawnych danych po stronie użytkownika, na przykład wskazanie miejsc, w których znajdą się luki do uzupełnienia.

## 1.7 Koncepcja rozwiązania

Generator testów egzaminacyjnych z języka C++ został stworzony jako przenośna aplikacja desktopowa, działająca na systemach operacyjnych z rodzin Windows i Linux. Korzysta ona z zewnętrznego kompilatora g++, który pozwala na sprawdzanie poprawności kodu źródłowego w zadaniach. System posiada również funkcjonalność umożliwiającą tworzenie plików w formacie PDF, co sprawia, że arkusze egzaminacyjne po wygenerowaniu są gotowe do druku.

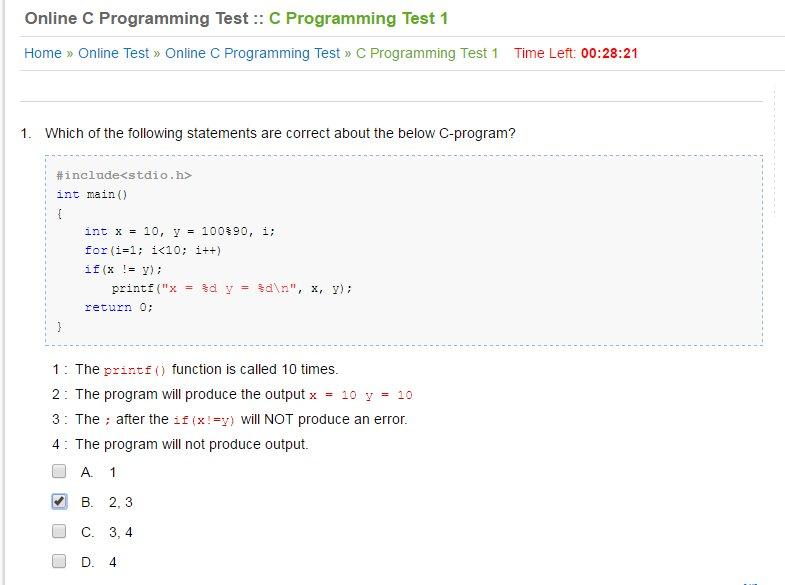
# 2. ISTNIEJĄCE ROZWIĄZANIA

Aby udowodnić, że stworzenie takiego programu ułatwi pracę jego użytkownikom, należałoby opowiedzieć o aplikacjach, które już istnieją i rozwiązują problem, który on posiada. Niestety, nie ma pojedynczego narzędzia, które bezpośrednio wspiera układanie egzaminów z języków programowania. Istnieje jednak wiele programów, które wspomagają studentów w nauce pisania kodu i wykorzystują podobne technologie do tych zastosowanych w Generatorze testów egzaminacyjnych z języka C++. Żadna z tych aplikacji nie wspiera bezpośrednio tworzenia dokumentu uzupełnianego automatycznie o dane uzyskane poprzez wywołanie zewnętrznego procesu kompilacji kodu. Dlatego też w tym rozdziale uwaga zostanie skupiona na narzędziach lub zbiorach narzędzi, które zajmują się testowaniem posiadanych umiejętności z programowania w języku C++ lub wykorzystują podobne technologie do tych wykorzystanych w Generatorze. Można do nich zaliczyć np. bardzo licznie występujące aplikacje internetowe, które mają na celu sprawdzenie wiedzy ze znajomości języków programowania poprzez zaznaczanie lub wpisywanie poprawnego wyniku wykonania kodu źródłowego czy też zaawansowane IDE wspierające naukę oraz zbiór narzędzi, który dotychczas był wykorzystywany przez użytkownika do tworzenia egzaminów.

## 2.1. Online C Programming Test IndiaBIX™ Technologies

Pierwszy przykład, który można przytoczyć jako oprogramowanie służące do testowania wiedzy to narzędzie stworzone przez firmę IndiaBIX™ Technologies o  nazwie *Online C Programming Test*. Jest to aplikacja webowa, stworzona w języku angielskim, która udostępnia możliwość wyboru testu spośród kilkunastu dostępnych, nie tylko z języka C, ale również z Java czy baz danych. Gdy użytkownik wybierze już interesujący go test, aplikacja zaczyna odmierzać czas, w którym rozwiązujący musi się zmieścić Po ukończeniu lub po skończeniu czasu użytkownikowi podawany jest jego wynik w postaci: liczba punktów zdobytych/liczba punktów możliwych do zdobycia. Oprócz tego, program udostępnia dla każdego zadania poprawne odpowiedzi wraz z wyjaśnieniem, dlaczego dana odpowiedź jest poprawna, gdy użytkownik odpowie błędnie.

Interfejs użytkownika tej aplikacji przedstawiony został na poniższym rysunku (Rys. 2.1.). Można stwierdzić, że jest to interfejs przyjazny użytkownikowi. Pod poleceniem znajduje się wydzielone miejsce na część zawierającą kod w języku C.  Ten fragment pisany jest czcionką o stałej szerokości znaków, by ułatwić czytanie potencjalnym użytkownikom. Pod tą częścią znajduje się kolumna kolumna pól wyboru, gdzie należy zaznaczyć jedną lub wiele odpowiedzi, które rozwiązujący test uznaje za poprawne. W lewym górnym rogu znajduje się dobrze widoczny, czerwony zegar uświadamiający użytkownikowi, ile jeszcze czasu pozostało mu do automatycznego zakończenia testu.

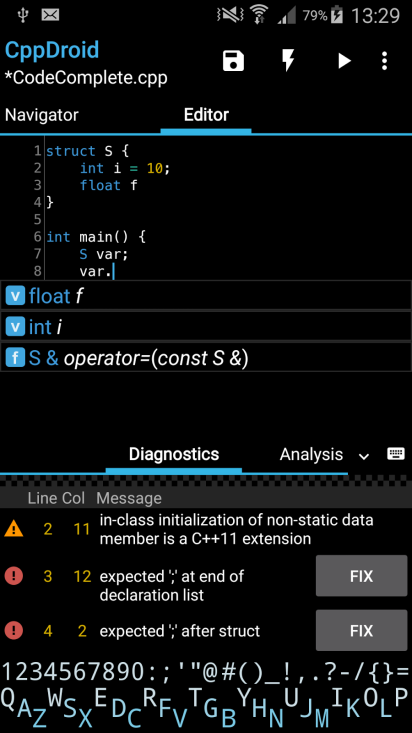
Rys. 2.1. Wygląd aplikacji testowej stworzonej przez IndiaBIX

Wadą tego rozwiązania jest fakt, że w momencie, gdy zajdzie potrzeba zmiany zadania, nowy kod należy skompilować przy pomocy zewnętrznego kompilatora by sprawdzić jego poprawność, ponownie umieścić na stronie wraz z poprawną odpowiedzią, którą również trzeba zweryfikować. Warto zauważyć, że może to prowadzić do wielu błędów, jednak zwracając uwagę na fakt, że częstotliwość zmian jest niewielka oraz za tworzenie testu odpowiada więcej niż jedna osoba, która próbuje wykryć błędy, można uznać, że jest to akceptowalne.

Kolejną wadą aplikacji jest brak możliwości automatycznego przechowywania kodu źródłowego zadań po wybraniu pliku, w którym ma zostać zapisany. Prowadzi to do sytuacji, w której aby znaleźć konkretne zadanie, by np. wykorzystać kod do kolejnego testu, należy osobiście zapisywać kod zadań w oddzielnych plikach, co powoduje, że zwiększamy czas tworzenia nowego testu, albo przeszukiwać wszystkie testy w celu odnalezienia tego jedynego kodu, który miał zostać wykorzystany.

Największymi zaletami tego rozwiązania jest możliwość dynamicznego sprawdzania poprawności odpowiedzi przez osobę zdającą test po jego ukończeniu a także duża dostępność aplikacji, gdyż, aby móc z niej korzystać, wystarczy jedynie przeglądarka internetowa z połączeniem internetowym.

## 2.2. Kompilator CppDroid – C/C++ IDE

Poprzedni przykład aplikacji dotyczył tworzenia testów w języku C++ bez użycia kompilatora. Jeżeli był on wykorzystany to zewnętrznie, jako oddzielne narzędzie. Tym razem omówiony zostanie przykład, który, chociaż nie tworzy testów, jednak wykorzystuje kompilator. Jest to aplikacja CppDroid stworzona na platformę Android. Program to IDE, które dodatkowo posiada bardzo rozwinięte podpowiedzi dotyczące składni języka C i C++ jak również konfigurowalnym dla użytkownika wyglądem w postaci możliwości zmian skórek i motywów narzędzia. To zabieg marketingowy, który ma na celu zwrócenie uwagi użytkownika i zachęcenie go do instalacji i korzystania.

Rys. 2.2. Wygląd aplikacji CppDroid

Na powyższym rysunku można zauważyć jak proste jest korzystanie z CppDroid. Jak widać, wyświetlane komunikaty są czytelne i zrozumiałe, nawet dla bardzo początkujących programistów. Program posiada również gotowe przykłady kodu, na których można się wzorować lub je edytować, skompilować i sprawdzić jak zmieni się działanie programu. Oprócz przykładów, twórcy aplikacji idą o krok dalej i udostępniają poradniki do nauki wraz z przykładami, które również można przetestować za pomocą tylko i wyłącznie telefonu z systemem Android.

Wadą tego rozwiązania jest przede wszystkim fakt, że nie jest ono darmowe. Podstawowe funkcje nie wymagają od nas dodatkowych opłat, jednak, aby korzystać z takich możliwości jak samouczek czy przykładowe kody, należy zapłacić drobną kwotę (na dzień 03.12.2016 r. 1.99$ za możliwość pięciokrotnego otwarcia jakiegokolwiek przykładu lub samouczka w czasie jednego startu aplikacji).

## 2.3. Kompilator i edytor tekstu

Powyższe przykłady aplikacji wykorzystują podobne technologie czy założenia, co Generator testów egzaminacyjnych z języka C++, jednak nie byłyby pomocne podczas tworzenia egzaminów dla prowadzącego przedmiot. W tym podrozdziale zostanie omówiony zbiór dwóch narzędzi, które połączone umożliwiają nauczycielowi napisanie testu dla studentów.

Dotychczas do tworzenia egzaminów z przedmiotu Podstawy Programowania, prowadzący korzystał ze standardowych narzędzi dostępnych na prawie każdym komputerze programisty – zewnętrznego kompilatora języka C++ a także edytora tekstowego. Piszący test wykorzystywał kompilator do napisania poprawnego kodu, który następnie kopiował i wklejał do edytora tekstowego w celu ułożenia zadań w odpowiedniej kolejności oraz formatowania wyglądu.

To rozwiązanie niesie ze sobą wiele problemów. Przede wszystkim jest czasochłonne. Łatwo to zauważyć w przypadku pomyłki przy kompilacji czy chęci uzyskania odpowiedzi niewiele różniącej się od poprzedniej. Tworzący egzamin musi ponownie skopiować treść zadania do edytora tekstu, sformatować czcionkę, jej rozmiar jak również wyjście kompilatora wstawić do osobnego dokumentu, w którym znajdować się będą polecenia wraz z gotowymi odpowiedziami dla osób sprawdzających egzamin.

Kolejną wadą jest duża szansa popełnienia pomyłki. Może ona wystąpić, gdy kod z kompilatora zostanie skopiowany do dokumentu a następnie zmieniany bezpośrednio w edytorze tekstu. Może się wtedy okazać, że zmiana spowodowała, że zadanie przestało się kompilować. Gdy ten fakt zostanie przeoczony przez osobę tworzącą test, prawdopodobnie będzie to prowadzić do konieczności zaliczenia błędnie napisanego zadania wszystkim studentom lub nawet powtórzenia sprawdzianu. Nie jest to korzystne dla prowadzącego, gdyż jego zadaniem jest weryfikacja faktycznej wiedzy studentów.

Ze względu na specyfikę przedmiotu, egzaminy najczęściej przeprowadzane i sprawdzane są przez więcej niż jedną osobę. Wiąże się to z koniecznością stworzenia dwóch dokumentów: jeden dla studenta z pustymi miejscami na odpowiedź, której będzie mógł udzielić oraz drugi – dla nauczyciela – z uzupełnionymi poprawnymi odpowiedziami, żeby ułatwić ocenę pracy.

Jednak takie rozwiązanie ma również swoje zalety. Jedną z nich jest jego wysoka skalowalność. W przypadku, gdy zmieni się formuła egzaminu, sposób jego tworzenia nie wpłynie na konieczność zastosowania innego rozwiązania. Podobnie stanie się w przypadku, gdy nastąpi zmiana języka programowania, dla którego prowadzony jest przedmiot Podstawy Programowania. Wówczas prowadzący będzie mógł użyć innego kompilatora, jednak sama idea tworzenia egzaminu pozostanie taka sama.

Zatem jeżeli prowadzącemu zajęcia zależy na narzędziu lub zbiorze narzędzi, które jest jak najbardziej uniwersalne, to powinien zostać przy tym tradycyjnym sposobie.

## 2.4. Podsumowanie

Omówione przykłady, zwłaszcza pierwsze dwa, nie są dokładnymi przykładami istniejących rozwiązań dla generatora testów egzaminacyjnych z języka C++. Zastosowane są tam raczej podobne zmysły czy technologie. W pierwszym przypadku możemy zauważyć analogię w postaci testu z języka C++. Jest to rozwiązanie trochę inne, webowe z dynamicznym sprawdzaniem poprawności odpowiedzi, czego w przypadku generatora testów egzaminacyjnych z języka C++ brakuje. Zaś drugi przypadek pokazuje aplikację, która działa na podstawie wbudowanego kompilatora, podobnie jak stworzony generator. Obudowuje jego działanie w interfejs użytkownika i opcje niedostępne dla innych tego typu programów. Tutaj również można zauważyć analogię z generatorem testów egzaminacyjnych z języka C++.

Trzeci przykład jest zupełnie inny. Jest to rozwiązanie wykorzystywane przez prowadzących na wielu uczelniach. Polega ono na wpisaniu kodu do kompilatora, a gdy jest poprawny, skopiowaniu go do edytora tekstu, by móc sformatować dokument dla studenta.

Jak widać, nie istnieje rozwiązanie, które wymagałoby od osoby tworzącej egzamin jedynie umiejętności tworzenia pytań testowych. Więcej czasu zajmuje jednak kopiowanie kodu, formatowanie wyglądu dokumentu i sprawdzanie występowania jakichkolwiek pomyłek niż merytoryczna i efektywna praca. Z tego właśnie powodu, powstał produkt taki jak Generator testów egzaminacyjnych z języka C++.

Jego główną zaletą  jest to, że osoba tworząca sprawdzian dla studentów nie musi przejmować się wyglądem egzaminu, gdyż jest on zaimplementowany z góry. Istnieje możliwość dokonania drobnych zmian, jak np. zmiana czcionki czy jej rozmiaru dla kodu i polecenia.

Korzystając z programu, nie istnieje konieczność kopiowania kodu ze względu na wbudowaną w aplikacji obsługę kompilatora języka C++, zadanie możemy skompilować z poziomu interfejsu użytkownika generatora testów egzaminacyjnych z języka C++ i jednocześnie sprawdzić czy odpowiedzi, które zostaną następnie zapisane na egzaminie dla nauczyciela są takie, jakich wstępnie osoba tworząca egzamin oczekiwała, więc generowanie testów dla studentów staje się prostsze, czytelniejsze i znacznie mniej podatne na błędy tworzącego.

Jedną z kolejnych zalet jest możliwość zapisania zarówno egzaminu jak i zadania do pliku xml i wczytania w dowolnym momencie, np., gdy zajdzie potrzeba wykorzystania stworzonego zasobu ponownie. Nie ma potrzeby, jak w przypadku testów online stworzonych przez IndiaBIX™ Technologies, przeszukiwania źródeł strony (w tym przypadku egzaminów z zeszłych lat) by odnaleźć konkretne zadanie, gdyż można je oddzielnie indeksować np. w konkretnym katalogu i generator to ułatwia poprzez zastosowane rozwiązanie zapamiętywania katalogu, w którym ostatnio egzamin został zapisany.

# 3. KONCEPCJA ROZWIĄZANIA

## 3.1 Wymagania

Wymagania są istotną częścią projektu informatycznego i stanowią odzwierciedlenie potrzeb klienta. Przed rozpoczęciem właściwych prac nad implementacją, konieczne więc było ich sformułowanie wraz z opiekunem naszej pracy. Wynikiem wspólnych rozmów są przedstawione poniżej wymagania. Większość z nich wynika bezpośrednio z celów biznesowych i funkcjonalnych, natomiast pozostałe zostały dodane w trakcie implementacji projektu. Warto zauważyć, że nie zostały określone wymagania w zakresie wydajności. Jest to spowodowane faktem, iż opiekun naszej pracy nie oczekiwał konkretnego wzrostu szybkości pracy. Kluczowe było bowiem stworzenie uniwersalnego środowiska dla wytwarzania egzaminów, które uniezależniłoby użytkownika od korzystania z kilku zewnętrznych aplikacji do tego celu.

### 3.1.1 Wymagania funkcjonalne

* umożliwienie użytkownikowi wygenerowania testu osobno w wersji dla studenta i nauczyciela na podstawie tych samych danych,
* możliwość wyboru typu zadania spośród kilku, z góry zdefiniowanych, typów:
  + „wyjście programu”,
  + „wyjście funkcji”,
  + „zwrócona liczba”,
  + „wartości zmiennych”,
  + „numery linii”,
  + „uzupełnianie pól”,
  + „własny”,
* możliwość zmiany domyślnych poleceń dla ww. typów zadań,
* reagowanie na błędy wykonania kodów dołączonych do zadań:
  + informowanie o wystąpieniu błędów wykonania (np. zapętlanie się programu)
  + zapobieganie zawieszeniu się aplikacji w przypadku wykonania zapętlającego się programu,
* automatyczne sprawdzenie kompletności zadania przed generacją arkusza:
  + sprawdzenie czy kod i polecenie są dostarczone do zadania,
  + sprawdzenie kompilowania się dostarczonego kodu,
* zapewnienie spójności danych w arkuszu z danymi zawartymi w programie (podczas generacji arkusza),
* automatyczne wyznaczanie odpowiedzi do zadania po poprawnym wykonaniu dostarczonego kodu,

### 3.1.2 Wymagania na dane

* umożliwienie zapisu/odczytu utworzonego egzaminu na/z dysku,
* przechowywanie domyślnych treści poleceń do poszczególnych typów zadań na dysku,
* umożliwienie zapisu/odczytu utworzonego zadania na/z dysku,

### 3.1.3 Wymagania jakościowe

#### 3.1.3.1 Wymagania w zakresie wiarygodności

* odporność na błędy użytkownika
  + sprawdzanie formatu danych wprowadzanych przez użytkownika
  + sprawdzanie kompletności danych wprowadzanych przez użytkownika
* informowanie użytkownika o wystąpieniu błędów

#### 3.1.3.2 Wymagania w zakresie elastyczności

* działanie w systemie operacyjnym Linux i Windows
* możliwość kompilacji i wykonania kodu w języku C++ w standardzie ISO/IEC 14882:2014

#### 3.1.3.3 Wymagania w zakresie użyteczności

* dostęp do wszystkich funkcjonalności programu w 3 lub mniej kliknięciach
* generowanie arkuszy egzaminacyjnych w formacie PDF

## 3.2 Model przypadków użycia

### 3.2.1 Skrócony diagram przypadków użycia

Na rysunku 3.1 znajduje się skrócony diagram przypadków użycia. Przedstawia on przypadki, które były kluczowe w realizacji projektu. Pełny diagram znajduje się w dodatku 1.

***C:\Users\Pawel\Downloads\Skrócony diagram przypadków użycia.png***

Rys. 3.1. Skrócony diagram przypadków użycia

### 3.2.2 Skrócony opis przypadków użycia

W tabelach 3.1 - 3.11 opisane zostały przypadki użycia przedstawione na rysunku 3.1. Opis wszystkich zdefiniowanych przypadków użycia znajduje się w dodatku 2.

**Tabela 3.1.** Zarządzanie zadaniami

|  |  |
| --- | --- |
| Nazwa przypadku: | Zarządzanie zadaniami |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie z menu opcji „Zadanie” |
| Warunki początkowe: | Brak |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie zawartości menu „Zadanie”. |

**Tabela 3.2.** Dodawanie zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Dodawanie zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Dodaj” |
| Warunki początkowe: | Otwarte menu „Zadanie” |
| Opis przebiegu interakcji: | 1. Program otwiera okno kreatora zadania i blokuje przycisk dodawania zadania.  2. Użytkownik wybiera typ zadania.  3. Program odblokowuje przycisk dodawania zadania.  4. Użytkownik wybiera plik z kodem do zadania.  5. Użytkownik wybiera opcję „Zapisz”. |
| Przebiegi alternatywne: | 2a.1 Użytkownik pomija krok 2 i przechodzi do pkt. 5.  2a.3 Program nie pozwala na dodanie nowego zadania.  2a.4 Użytkownik wraca do pkt. 2.  4a Użytkownik pomija krok 4. |
| Sytuacje wyjątkowe: | 4a.1 Użytkownik wpisuje niepoprawną nazwę pliku (nieistniejący plik).  4a.2 Program komunikuje brak pliku o podanej nazwie.  4b.1 Użytkownik wpisuje niepoprawną nazwę pliku (zły format pliku).  4b.2 Program ignoruje wprowadzony plik. |
| Warunki końcowe: | Dodanie nowego zadania do arkusza i wyświetlenie go. |

**Tabela 3.3.** Usuwanie zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Usuwanie zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Usuń” |
| Warunki początkowe: | Otwarte menu „Zadanie”, niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Usunięcie aktualnego zadania z arkusza. |

**Tabela 3.4.** Edycja zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Edycja zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Edytuj” |
| Warunki początkowe: | Otwarte menu „Zadanie”, niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | 1. Program otwiera okno kreatora zadania, uzupełnia je danymi o aktualnym zadaniu i odblokowuje przycisk dodania zadania.  2. Użytkownik wybiera typ zadania.  3. Użytkownik wybiera plik z kodem do zadania.  4. Użytkownik wybiera opcję „Zapisz”. |
| Przebiegi alternatywne: | 2a Użytkownik pomija krok 2.  3a Użytkownik pomija krok 3. |
| Sytuacje wyjątkowe: | 3a.1 Użytkownik wpisuje niepoprawną nazwę pliku (nieistniejący plik).  3a.2 Program komunikuje brak pliku o podanej nazwie.  3b.1 Użytkownik wpisuje niepoprawną nazwę pliku (zły format pliku).  3b.2 Program ignoruje wprowadzony plik. |
| Warunki końcowe: | Zastąpienie treści aktualnego zadania nowo wprowadzonymi danymi. |

**Tabela 3.5.** Edycja polecenia domyślnego

|  |  |
| --- | --- |
| Nazwa przypadku: | Edycja polecenia domyślnego |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie zakładki „Polecenia domyślne” |
| Warunki początkowe: | Otwarte okno „Ustawienia” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno edycji poleceń domyślnych.  2. Użytkownik wybiera typ zadania.  3. Program wyświetla polecenie domyślne dla wybranego typu zadania w polu do edycji.  4. Użytkownik edytuje treść polecenia.  5. Użytkownik wybiera opcję „Zastosuj”. |
| Przebiegi alternatywne: | 2a.1 Użytkownik pomija krok 2 i przechodzi do pkt. 5.  2a.2 Program nie podejmuje żadnego działania.  2a.3 Użytkownik powraca do pkt. 2.  4a Użytkownik pomija krok 4. |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Zastąpienie treści wybranego polecenia domyślnego zawartością pola do edycji. |

**Tabela 3.6.** Wykonywanie kodu dołączonego do zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Wykonywanie kodu dołączonego do zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Wykonaj” lub „Wykonaj testowo” |
| Warunki początkowe: | Niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | 1. Program kompiluje kod dołączony do zadania.  2. Wynik kompilacji zostaje wyświetlony na ekranie.  3. Program wykonuje program powstały po kompilacji.  4. Wynik programu zostaje wyświetlony na ekranie. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1 Dołączony kod nie kompiluje się.  1a.2 Wynik nieudanej kompilacji zostaje wyświetlony na ekranie.  1b.1 Zadanie nie posiada kodu.  1b.2 Program wyświetla informację o pustym pliku wejściowym dla kompilatora.  3a.1 Dołączony kod posiada błąd wykonania (np. nieskończona pętla).  3a.2 Program wyświetla komunikat o przekroczeniu czasu wykonania. |
| Warunki końcowe: | Wyświetlenie wyniku kompilacji i wykonania. |

**Tabela 3.7.** Zmiana kolejności zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Zmiana kolejności zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie z menu kontekstowego zadania opcji „Przesuń w …” |
| Warunki początkowe: | Niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | 1. Program zamienia kolejnością aktualnie zadanie z zadaniem znajdującym się po wybranej stronie. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1 W arkuszu znajduje się tylko jedno zadanie.  1a.2 Pozycja zadania zostaje odświeżona.  1b.1 Została wybrana opcja „Przesuń w lewo” dla zadania znajdującego się skrajnie z lewej strony.  1b.2 Pozycja zadania zostaje odświeżona.  1c.1 Została wybrana opcja „Przesuń w prawo” dla zadania znajdującego się skrajnie z prawej strony.  1c.2 Pozycja zadania zostaje odświeżona. |
| Warunki końcowe: | Zmiana kolejności zadań w arkuszu. |

**Tabela 3.8.** Zarządzanie arkuszem do tworzenia egzaminu

|  |  |
| --- | --- |
| Nazwa przypadku: | Zarządzanie arkuszem do tworzenia egzaminu |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie z menu opcji „Arkusz” |
| Warunki początkowe: | Brak |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie zawartości menu „Arkusz”. |

**Tabela 3.9.** Odczyt arkusza

|  |  |
| --- | --- |
| Nazwa przypadku: | Odczyt arkusza |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Wczytaj arkusz” |
| Warunki początkowe: | Otwarte menu „Arkusz” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno wyboru pliku.  2. Użytkownik wybiera żądany plik.  3. Użytkownik potwierdza wybór przyciskiem „Otwórz”.  4. Program tworzy nowy arkusz i ładuje do niego dane z pliku. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 2a.1 Użytkownik wybiera plik niebędący arkuszem  2a.2 Program komunikuje błąd odczytu danych z pliku i przerywa proces ładowania arkusza. |
| Warunki końcowe: | Załadowanie danych z pliku do nowego arkusza. |

**Tabela 3.10.** Zapis arkusza

|  |  |
| --- | --- |
| Nazwa przypadku: | Zapis arkusza |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Zapisz” |
| Warunki początkowe: | Otwarte menu „Arkusz” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno wyboru miejsca zapisu.  2. Użytkownik wybiera żądaną lokalizację (wraz z nazwą pliku).  3. Program zapisuje zawartość arkusza we wskazanym pliku. |
| Przebiegi alternatywne: | 1a.1 W danej instancji programu został już raz zapisany arkusz.  1a.2 Program zapisuje zawartość arkusza we wcześniej wskazanym pliku. |
| Sytuacje wyjątkowe: | 3a.1 Użytkownik nie ma praw do zapisu pliku w danej lokacji.  3a.2 Program wyświetla komunikat o niemożności zapisu pliku w podanej lokacji. |
| Warunki końcowe: | Zapisanie danych zawartych w arkuszu, we wskazanym pliku. |

**Tabela 3.11.** Generacja arkusza egzaminacyjnego

|  |  |
| --- | --- |
| Nazwa przypadku: | Generacja arkusza egzaminacyjnego |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Generuj PDF” |
| Warunki początkowe: | Otwarte menu „Arkusz” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno generacji arkusza egzaminacyjnego |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1 Nie jest otworzony żaden arkusz.  1a.2 Program wyświetla komunikat o niemożności wygenerowania pustego arkusza. |
| Warunki końcowe: | Wyświetlenie okna generacji arkusza egzaminacyjnego. |

# 4. PROJEKT

## 4.1. Projekt architektury systemu

### 4.1.1. Koncepcja systemu

System jest zrealizowany w formie aplikacji desktopowej na systemy operacyjne Windows i Linux, napisanej w języku Java, uruchamianej w środowisku JRE 8 [2][3]. Do stworzenia interfejsu graficznego [9] użyta została technologia JavaFX [4]. Do generowania plików w formacie PDF została wykorzystana biblioteka Apache PDFBox [5].

System korzysta z zewnętrznego kompilatora g++. W systemie operacyjnym Windows do uruchomienia kompilatora wykorzystane zostało również środowisko MinGW.

Dane programu są przechowywane na dysku w formie plików XML.

### 4.1.2. Struktura systemu

#### 4.1.2.1. Struktura sprzętowa

W naszym systemie wyszczególniony jest jeden komponent sprzętowy, a mianowicie komputer z zainstalowanym systemem operacyjnym Linux lub Windows oraz środowiskiem Java.

#### 4.1.2.2. Struktura programowa

System opiera się na typowej architekturze trójwarstwowej, w skład której wchodzą: warstwa prezentacji, warstwa usług i warstwa danych (opisane w tabelach 4.1 – 4.3). Do implementacji warstwy prezentacji i warstwy usług zastosowaliśmy wzorzec architektoniczny Model - View - Controller w formie niejako wymuszonej przez technologię JavaFX. Podstawowe komponenty zdefiniowane w każdej z warstw przedstawiono na rysunku Rys.4.1.1 i opisano w tabelach 4.4 – 4.12.

https://lh6.googleusercontent.com/mDSIWh4GJewDfmPwMzV_ph2CkXJK1iz9O467q-A6P0k8Fk9ZoJq2hFoYDzcqliD0As6umEIPFDEI3NAnkXxBGFIFSgpFAXNCv9V1367oz1BFdX_8fyuC9mj_0WG1JQtgs6Gmg3MlyOxvaDFkrQ

Rys. 4.1. Diagram komponentów systemu

*4.1.3. Warstwy architektoniczne*

**Tabela 4.1** Charakterystyka warstwy prezentacji

|  |  |
| --- | --- |
| LAYR\_001 | **Warstwa prezentacji** |
| Opis: | Interfejs graficzny w postaci aplikacji desktopowej zrealizowanej za pomocą technologii JavaFX. Składają się na niego przede wszystkim główne okno programu, okno zapisu PDF, okno zarządzania zadaniem i okno ustawień zaawansowanych. |
| Elementy wzorca MVC: | Widok (pliki FXML), Kontroler |

**Tabela 4.2** Charakterystyka warstwy usług

|  |  |
| --- | --- |
| LAYR\_002 | **Warstwa usług** |
| Opis:  Opis:O | Aplikacja w języku Java składająca się z modułów do zarządzania egzaminem, kompilacji i wykonania kodu C++ (za pomocą kompilatora g++ a w systemie operacyjnym Windows również środowiska MinGW) oraz generowania PDF (z wykorzystaniem biblioteki PDFBox). |
| Elementy wzorca MVC: | Model |

**Tabela 4.3** Charakterystyka warstwy danych

|  |  |
| --- | --- |
| LAYR\_003 | **Warstwa danych** |
| Opis: | Zapisywane na dysku pliki w formacie XML zawierające dane egzaminu, pojedynczego zadania lub domyślne polecenie do danego typu zadania. |

### 4.1.4. Komponenty programowe

Podstawowe komponenty warstwy prezentacji:

**Tabela 4.4** Charakterystyka komponentu GenerateExamController

|  |  |
| --- | --- |
| CTRL\_001 | **GenerateExamController** |
| Opis: | *Komponent realizujący operacje związane z obsługą przycisków w oknie tworzenia pliku PDF* |
| Powiązania: | SERV\_01 Exam |

**Tabela 4.5** Charakterystyka komponentu MainController

|  |  |
| --- | --- |
| CTRL\_002 | **MainController** |
| Opis: | *Komponent realizujący operacje związane z obsługą przycisków w głównym oknie programu* |
| Powiązania: | SERV\_01 Exam |

**Tabela 4.6** Charakterystyka komponentu ManageTaskController

|  |  |
| --- | --- |
| CTRL\_003 | **ManageTaskController** |
| Opis: | *Komponent realizujący operacje związane z obsługą przycisków w oknie tworzenia i edycji zadania* |
| Powiązania: | SERV\_02 Task |

Podstawowe komponenty warstwy usług:

**Tabela 4.7** Charakterystyka komponentu Exam

|  |  |
| --- | --- |
| SERV\_001 | **Exam** |
| Opis: | *Komponent realizujący operacje związane z zarządzaniem zbiorem zadań* |
| Powiązania: | SERV\_004 XML Manager |

**Tabela 4.8** Charakterystyka komponentu Task

|  |  |
| --- | --- |
| SERV\_002 | **Task** |
| Opis: | *Komponent realizujący operacje związane z zarządzaniem pojedynczym zadaniem* |
| Powiązania: | SERV\_004 XML Manager |

**Tabela 4.9** Charakterystyka komponentu GCC

|  |  |
| --- | --- |
| SERV\_003 | **GCC** |
| Opis: | *Komponent realizujący operacje związane z kompilacją i wykonaniem kodu dołączonego do zadania* |
| Powiązania: | System zewnętrzny G++ |

**Tabela 4.10** Charakterystyka komponentu XML Manager

|  |  |
| --- | --- |
| SERV\_004 | **XML Manager** |
| Opis: | *Komponent realizujący operacje związane z wczytywaniem / zapisywaniem danych do pliku XML* |
| Powiązania: | DCMP\_01 Plik XML |

**Tabela 4.11** Charakterystyka komponentu GCC

|  |  |
| --- | --- |
| SERV\_005 | **PDF Manager** |
| Opis: | *Komponent realizujący operacje związane z generowaniem arkusza w formacie PDF* |
| Powiązania: | Biblioteka PDFBox |

Podstawowe komponenty warstwy danych:

**Tabela 4.12** Charakterystyka komponentu GCC

|  |  |
| --- | --- |
| DCMP\_001 | **Plik XML** |
| Opis: | *Pliki w formacie  XML przechowujące dane egzaminu, pojedynczego zadania i domyślne polecenia dla poszczególnych typów zadań.* |
| Powiązania: | brak |

## 4.2. Projekt logiki aplikacji

### 4.2.1. Koncepcja modelu klas

W systemie zostało zdefiniowanych kilka głównych klas. PDF implementuje generowanie plików w formacie PDF. GCC zawiera metody odpowiedzialne za kompilację i wywołanie kodu. Task jest klasą reprezentującą pojedyncze zadanie, a Exam przechowuje zbiór obiektów klasy Task i pozwala na zarządzanie tym zbiorem. Aby możliwe było zróżnicowanie sposobu generowania odpowiedzi w zależności od typu zadania, utworzona została klasa abstrakcyjna TaskType i dziedziczące po niej klasy reprezentujące poszczególne typy zadań. Poza tym istnieją klasy kontrolerów odpowiadające poszczególnym widokom.

### 4.2.2. Diagram klas

Untitled Diagram.png

Rys. 4.2. Diagram klas

**Tabela 4.13** Charakterystyka klasy MainController

|  |  |  |
| --- | --- | --- |
| CLS\_001 | **MainController** | |
| Opis: | Klasa zawierająca instancję klasy Exam i status zakładek. Pozwala na obsługę przycisków w głównym oknie programu. | |
| Właściwości: | (exam: Exam, status: List<String>) | *egzamin, lista statusów zakładek* |
| execute(): void | *obsługuje przycisk wykonania kodu* |
| testExecute(): void | *obsługuje przycisk wykonania testowego kodu* |
| createPDF(): void | *obsługuje przycisk generacji PDF* |
| addTask(): void | *obsługuje przycisk dodawania zadania* |
| editTask(): void | *obsługuje przycisk edycji zadania* |
| deleteTask(): void | *obsługuje przycisk usuwania zadania* |
| update(int): void | *aktualizuje zawartość okna* |
| addTab(): void | *dodaje nową zakładkę* |
| deleteTab(): void | *usuwa aktualną zakładkę* |
| save(int): void | *zapisuje stan komponentów* |
| saveToXML(): void | *obsługuje przycisk zapisu arkusza* |
| loadFromXML(): void | *obsługuje przycisk odczytu arkusza* |
| saveTask(): void | *obsługuje przycisk zapisu zadania* |
| loadTask(): void | *obsługuje przycisk odczytu zadania* |

**Tabela 4.14** Charakterystyka klasy ManageTaskController

|  |  |  |
| --- | --- | --- |
| CLS\_002 | **ManageTaskController** | |
| Opis: | Klasa zawierająca treść polecenia, kod źródłowy i typ zadania, będący instancją klasy TaskType. Pozwala na obsługę przycisków w oknie zarządzania zadaniem. | |
| Właściwości: | (content: String, codeList: List<String>, type: TaskType) | *treść polecenia, kod źródłowy, typ zadania* |
| editTask(Task): void | *ładuje zadanie do edycji* |
| updateText(Content): void | *aktualizuje polecenie* |
| addType(int): void | *dodaje do okna treść polecenia dla wybranego typu zadania* |
| finishEdition(): void | *obsługuje przysick końca edycji* |
| cancelEdition(): void | *obsługuje przycisk anulowania edycji* |
| selectCodeFile(): void | *obsługuje przycisk wyboru pliku z kodem* |

**Tabela 4.15** Charakterystyka klasy GenerateExamController

|  |  |  |
| --- | --- | --- |
| CLS\_003 | **GenerateExamController** | |
| Opis: | Klasa pozwalająca na obsługę przycisków w oknie tworzenia pliku PDF, zapisująca arkusz i posiadająca ustawienia zaawansowane. | |
| Właściwości | saveFile(): void | *obsługuje przycisk zapisywania pliku* |
| cancel(): void | *obsługuje przycisk anulowania tworzenia pliku pdf* |
| browse(): void | *obsługuje przycisk wybierania katalogu* |
| advancedOptions(): void | *otwiera okno ustawień zaawansowanych* |

**Tabela 4.16** Charakterystyka klasy GCC

|  |  |  |
| --- | --- | --- |
| CLS\_004 | **GCC** | |
| Opis: | Klasa umożliwiająca kompilację oraz wykonywanie wprowadzonego w oknie programu kodu. | |
| Właściwości | (osName: String, path: String, file: File) | *nazwa systemu operacyjnego, ścieżka do katalogu z plikami, plik z kodem* |
| createFile(List<String>): boolean | *tworzy plik z dostarczonych linii kodu* |
| compile (List<String>): boolean | *dokonuje kompilacji pliku z kodem* |
| execute(List<String>, List<String>, String): void | *wykonuje wygenerowany w czasie kompilacji plik* |

**Tabela 4.17** Charakterystyka klasy Task

|  |  |  |
| --- | --- | --- |
| CLS\_005 | **Task** | |
| Opis: | Klasa reprezentująca w programie pojedyncze zadanie. Zawiera elementy tworzące zadanie: polecenie, dołączony kod i odpowiedzi. | |
| Właściwości | (content: Content, answers: List<String>, result: String, type: TaskType, text: Text, compiler: GCC) | *polecenie, odpowiedzi, wynik kompilacji lub wykonania, typ, kod zadania, kompilator* |
| calculateGaps(): void | *wyznacza odpowiedzi do zadania z lukami* |
| save(): void | *otwiera okno zapisu zadania* |
| load(): boolean | *otwiera okno odczytu zadania* |

**Tabela 4.18** Charakterystyka klasy TaskType

|  |  |  |
| --- | --- | --- |
| CLS\_006 | **TaskType** | |
| Opis: | Klasa zawierająca strategie dla parsowania kodu, wykonania i generowania odpowiedzi w zależności od typu zadania. | |
| Właściwości | (defaultContents: String, name: String, numOfAnswers: int) | *domyślne polecenie, nazwa typu zadania, liczba odpowiedzi do zadania* |
| generateAnswers(Task, List<String>, List<String>): void | *generuje odpowiedzi do zadania* |
| callExecute(Task, List<String>): void | *wykonuje kompilację kodu zawartego w zadaniu* |
| callTestExecute(Task, List<String>): void | *wykonuje testową kompilację kodu zawartego w zadaniu* |
| preparePdfAnswers(Task): void | *przygotowuje odpowiedzi, które znajdą się w pliku .pdf* |

**Tabela 4.19** Charakterystyka klasy Exam

|  |  |  |
| --- | --- | --- |
| CLS\_007 | **Exam** | |
| Opis: | Klasa stanowiąca kontekst dla modelu danych aplikacji. Reprezentuje pojedynczy arkusz egzaminacyjny (w obu wersjach jednocześnie). | |
| Właściwości | (tasks: List<Tasks>, names: List<String>, index: int) | *lista obiektów typu Task z ich nazwami i indeksami* |
| addTask(Task): void | *dodaje zadanie z poleceniem, kodem źródłowym i typem zadania do egzaminu* |
| editTask(Task): void | *pozwala na edycję aktualnie wybranego zadania* |
| deleteTask(int): void | *usuwa aktualnie wybrane zadanie* |
| changeTasksOrder(int, int): void | *zmienia kolejność zadań w egzaminie* |
| compileExam(): boolean | *kompiluje kod wszystkich zadań w egzaminie* |
| save(): void | *zapisuje arkusz egzaminacyjny (xml)* |
| load(): void | *wczytuje arkusz z pliku* |

**Tabela 4.20** Charakterystyka klasy PDF

|  |  |  |
| --- | --- | --- |
| CLS\_008 | **PDF** | |
| Opis: | Klasa umożliwiająca generację pliku w formacie PDF na podstawie danych zawartych w arkuszu. | |
| Właściwości | (testType: String, pdfFileType: String, pdfFileName: String, commandFontSize: int, codeFontSize: int, date: String, pdfFile: File, exam: Exam) | *typ testu (dla studenta, dla nauczyciela), nazwa i typ pliku, rozmiar czcionek, data egzaminu, plik arkusza, instancja klasy Exam* |
| generatePdf(): void | *tworzy plik z egzaminem w formacie PDF* |
| saveFile(): void | *zapisuje arkusz do pliku* |

## 4.3. Projekt interfejsu użytkownika

### 4.3.1. Schemat układu okna głównego

**Tabela 4.21** Schemat układu okna głównego

|  |  |
| --- | --- |
| Panel górny | |
| Panel lewy górny | Panel prawy |
| Panel lewy środkowy |  |
| Panel lewy dolny |

**Tabela 4.22** Charakterystyka panelu górnego

|  |  |
| --- | --- |
| UPNL\_001 | **Panel górny** |
| Opis: | Panel zawierający pasek menu głównego. |
| Komponenty: | menu główne: Arkusz, Zadanie, Ustawienia, Pomoc |

**Tabela 4.23** Charakterystyka panelu lewego górnego

|  |  |
| --- | --- |
| UPNL\_002 | **Panel lewy górny** |
| Opis: | Panel zawierający pole tekstowe z poleceniem i przyciski służące do formatowania tekstu polecenia. |
| Komponenty: | pole tekstowe polecenia, przyciski formatowania polecenia |

**Tabela 4.24** Charakterystyka panelu lewego środkowego

|  |  |
| --- | --- |
| UPNL\_003 | **Panel lewy środkowy** |
| Opis: | Panel zawierający pole tekstowe z odpowiedziami do zadania. |
| Komponenty: | pole tekstowe odpowiedzi, przyciski do oznaczania etykiet i odpowiedzi |

**Tabela 4.25** Charakterystyka panelu lewego dolnego

|  |  |
| --- | --- |
| UPNL\_004 | **Panel lewy dolny** |
| Opis: | Panel zawierający pole tekstowe (bez możliwości edycji) z wyjściem kompilatora lub programu dołączonego do zadania. |
| Komponenty: | pole tekstowe z wyjściem programu, przyciski wykonywania kodu |

**Tabela 4.26** Charakterystyka panelu prawego

|  |  |
| --- | --- |
| UPNL\_005 | **Panel prawy** |
| Opis: | Panel zawierający pole z kodem dołączonym do zadania i przyciski służące do zaznaczania fragmentów kodu (np. luk). |
| Komponenty: | pole z kodem źródłowym, przyciski “markery” |

*4.3.2. Formularze*

**Tabela 4.27** Charakterystyka formularza zapisu PDF

|  |  |  |
| --- | --- | --- |
| FORM\_001 | **Formularz zapisu PDF** | |
| Opis: | Formularz służący do wprowadzenia danych koniecznych do wygenerowania arkusza PDF. | |
| Dane: | Data testu: Date | *Data, która zostanie wydrukowana w nagłówku arkusza* |
| Typ testu: Enum | *Typ testu (nauczyciel, student)* |

**Tabela 4.28** Charakterystyka formularza zarządzania zadaniem

|  |  |  |
| --- | --- | --- |
| FORM\_002 | **Formularz zarządzania zadaniem** | |
| Opis: | Formularz służący do wprowadzenia danych zadania (podczas edycji lub tworzenia nowego zadania). | |
| Dane: | Typ zadania: Enum | *Typ zadania (Wyjście programu, Wyjście funkcji, Zwrócona liczba, Wartości zmiennych, Numery linii, Uzupełnianie luk, Typ własny)* |
| Kod: String | *Kod dołączony do zadania* |
| Polecenie: String | *Treść zadania* |

**Tabela 4.29** Charakterystyka formularza zmiany wyglądu dokumentu

|  |  |  |
| --- | --- | --- |
| FORM\_003 | **Formularz zmiany wyglądu dokumentu** | |
| Opis: | Formularz służący do zmiany typu i rozmiaru czcionek użytych do generowania arkusza PDF. | |
| Dane: | Czcionka kodu: Enum | *Czcionka dla kodu dołączonego do zadania (typ zdefiniowany w systemie)* |
| Rozmiar czcionki kodu: int | *Rozmiar czcionki dla kodu dołączonego do zadania* |
| Czcionka polecenia: Enum | *Czcionka dla polecenia (typ zdefiniowany w systemie)* |
| Rozmiar czcionki polecenia: int | *Rozmiar czcionki polecenia* |

**Tabela 4.30** Charakterystyka formularza zmiany limitu czasu wykonania

|  |  |  |
| --- | --- | --- |
| FORM\_004 | **Formularz zmiany limitu czasu wykonania** | |
| Opis: | Formularz służący do zmiany limitu czasu wykonania kodu dołączonego do zadania. | |
| Dane: | Limit czasu wykonania: int | *Limit czasu wykonania kodu dołączonego do zadania* |

**Tabela 4.31** Charakterystyka formularza zmiany szerokości elementów zadania

|  |  |  |
| --- | --- | --- |
| FORM\_005 | **Formularz zmiany szerokości elementów zadania** | |
| Opis: | Formularz służący do zmiany szerokości pól na polecenie i kod w arkuszu PDF. | |
| Dane: | Szerokość polecenia: int | Część szerokości arkusza PDF jaką zajmie polecenie (w procentach). |

## 4.4. Projekt struktury danych

### 4.4.1. Koncepcja struktury danych

Dane w systemie przechowywane są wyłącznie w plikach, nie przewiduje się korzystania z bazy danych. Każdy z plików z danymi ma format XML. Stałym plikiem w systemie jest plik zawierający domyślne polecenia, które mogą być wykorzystywane w zadaniach. Plik ten znajduje się w katalogu roboczym programu. Ponadto jest możliwość utworzenia pliku z pojedynczym zadaniem. Taki plik zawiera treść polecenia i kodu wraz z ich formatami. Ostatnim typem pliku jest plik z arkuszem. Znajdują się w nim zadania i ich nazwy, a także indeks aktualnie aktywnej zakładki z zadaniem. Pliki z zadaniami i egzaminami są zapisywane w lokalizacjach wskazanych przez użytkownika.

### 4.4.2. Pliki danych

**Tabela 4.32** Charakterystyka pliku zadania

|  |  |  |  |
| --- | --- | --- | --- |
| FILE\_001 | **Zadanie** | | |
| Opis: | Plik zadania reprezentujący pojedyncze zadanie wchodzące w skład egzaminu. Umożliwia wielokrotne użycie zapisanego zadania w programie. | | |
| Dane: | Polecenie: List<Content> | Treść: String | *część treści polecenia, podzielona ze względu na format* |
| Format: Enum | *czcionka użyta do danej części polecenia (typ zdefiniowany wewnątrz systemu)* |
| Kod: List<Text> | Treść: String | *część kodu zadania, podzielona ze względu na markery* |
| Marker: String | *marker użyty do danej części zadania* |
| Etykiety: List<String> | | *lista etykiet w odpowiedzi do zadania* |
| Odpowiedź: List<String> | | *lista stanowiąca odpowiedź do zadania* |
| Wynik: String | | *wynik kompilacji* |
| Typ:  Type | Nazwa: String | *nazwa typu zadania* |
| Liczba odpowiedzi: int | *liczba odpowiedzi do zadania* |
| Stała odpowiedź: boolean | *flaga odpowiadająca za możliwość zmiany odpowiedzi do zadania przez program po wykonaniu kodu dołączonego do zadania* |
| Format: |  | XML | |

**Tabela 4.33** Charakterystyka pliku arkusza

|  |  |  |
| --- | --- | --- |
| FILE\_002 | **Arkusz** | |
| Opis: | Plik arkusza zawierający dane o zadaniach wchodzących w skład egzaminu. Umożliwia wielokrotne użycie zapisanego arkusza w programie. | |
| Dane: | Nazwy zadań: List<String> | *nazwy zakładek z zadaniami w arkuszu* |
| Bieżący indeks: int | *indeks aktywnej zakładki* |
| Maksymalny indeks: int | *następny w kolejności indeks zakładki* |
| Lista zadań: List<Task> | *lista zadań wchodzących w skład arkusza - ich format jest identyczny z formatem pojedynczego zadania (patrz: FILE\_001)* |
| Format: | XML | |

**Tabela 4.34** Charakterystyka pliku polecenia

|  |  |  |  |
| --- | --- | --- | --- |
| FILE\_003 | **Polecenia** | | |
| Opis: | Plik poleceń używany podczas tworzenia i edycji jednego z domyślnych typów zadań. | | |
| Dane: | Dane poleceń: List<TaskData> | Nazwa: String | *nazwa typu zadania* |
| Treść: String | *treść polecenia* |
| Format: | XML | | |

# 5. IMPLEMENTACJA

## 5.1 Wybór technologii

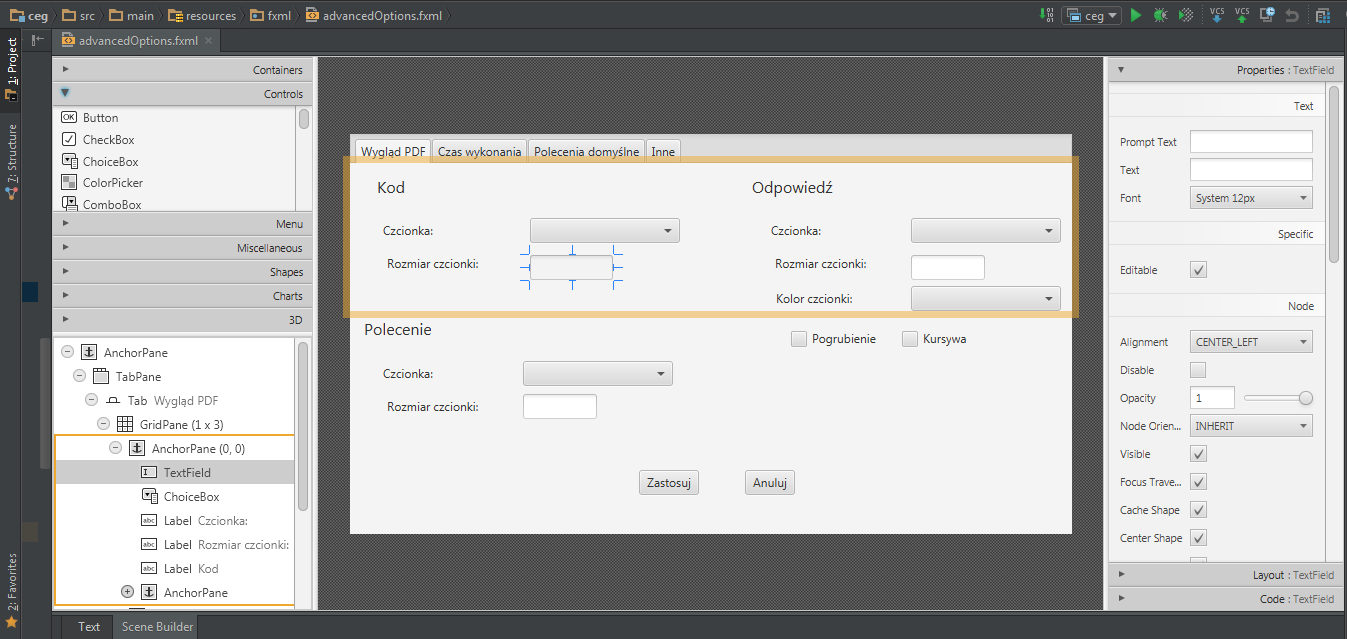
System zrealizowany został w formie aplikacji desktopowej na systemy operacyjne Windows i Linux. Ze względu na wymaganą przenośność systemu, jako język aplikacji została wybrana Java SE w najnowszej wersji 8 [2], uruchamiana w środowisku JRE 8. Pozwala ona na uruchomienie programu na każdym z wybranych systemów operacyjnych. Posiada wsparcie dla wyrażeń Lambda [3], wprowadza bibliotekę Stream i zwiększa wydajność programowania. Ponadto dysponuje rozbudowanymi bibliotekami standardowymi i szczegółową dokumentacją, a wbudowane mechanizmy ułatwiają programistom zarządzanie pamięcią.

## 5.2 Zastosowane technologie i biblioteki

### 5.2.1 JavaFX

Do stworzenia interfejsu graficznego wykorzystana została technologia JavaFX [4] z uwagi na bogaty zbiór bibliotek dla grafiki oraz możliwości szybkiego tworzenia zaawansowanych interfejsów graficznych [9]. Widok jest zbudowany z wykorzystaniem plików FXML – w oparciu o XML, dodając jednak odpowiednie znaczniki wewnątrz. Inicjalizacją obiektów, ustawieniem ich właściwości i stworzeniem zależności między nimi zajmuje się klasa FXMLoader. Widok można modyfikować  zarówno z poziomu IDE, jak i za pomocą narzędzia JavaFX Scene Builder (Rys 5.1.).

Jako model architektury wykorzystany został wzorzec MVC z użyciem FXML, który pozwala na podział programu na moduły porządkujące kod oraz ułatwiające pracę z nim. Widok zdefiniowany jest w pliku XML, klasa kontrolera jest jasno określona i odpowiada głównie za spięcie widoku z logiką biznesową i opcjonalnie za zdefiniowanie obsługi zdarzeń. Rozdzielenie widoku od modelu danych umożliwiło między innymi wprowadzenie modyfikacji w jednym module bez konieczności zmiany pozostałych lub ponowne użycie fragmentów kodu.



Rys. 5.1. Zrzut ekranu przedstawiający widok narzędzia JavaFX Scene Builder

### 5.2.2 JAXB

Java Architecture for XML Binding (JAXB) [6] jest frameworkiem, który umożliwia programistom odwzorowanie klas w Javie na reprezentację XML. Dwiema głównymi funkcjonalnościami oferowanymi przez JAXB jest zdolność do zapisywania obiektów Java do plików XML (marshalling) i odwrotność tej operacji, a więc konwersja zawartości XML do obiektów Java (unmarshalling). Innymi słowy, udostępnia przechowywanie i pobieranie danych w pamięci w dowolnym formacie XML. Framework wykorzystywany jest w naszym projekcie podczas zapisu i odczytu pojedynczego zadania i całego arkusza, a także przy odczytywaniu i modyfikacji poleceń domyślnych do zdefiniowanych typów zadań.

### 5.2.3 Apache PDFBox

Do tworzenia dokumentu w formacie PDF wykorzystana została otwarta biblioteka oparta na języku Java – Apache PDFBox [5] w najnowszej wersji 2.0.3. Charakteryzuje się łatwością dołączenia do projektu i użycia z językiem Java. Udostępnia pracę z dokumentami w formacie PDF, w tym odczytywanie i tworzenie, a przy tym jest konfigurowalna. Posiada też obszerną dokumentację i jest aktywnie wspierana. Wybraliśmy ją ze względu na możliwość tworzenia nie tylko zwykłych dokumentów PDF, ale także interaktywnych arkuszy, które były jednym z celów funkcjonalnych naszego systemu.

### 5.2.4 Kompilator

Generator testów egzaminacyjnych korzysta z zewnętrznego kompilatora g++. W systemie operacyjnym Windows do uruchomienia kompilatora wykorzystane zostało środowisko MinGW, które zawiera w sobie g++ i jest dostarczane użytkownikowi wraz z aplikacją. Spowodowane było to brakiem domyślnego kompilatora dla języka C++ w systemie Windows. Kompilator g++ jest wymagany od użytkowników używających naszej aplikacji na systemach Linux.

### 5.2.5 Apache Maven

Aby ułatwić zarządzanie projektem i jego zależnościami, postanowiliśmy użyć narzędzia automatyzującego budowę oprogramowania na platformę Java. Wybór padł na Apache Maven [7], ponieważ jest on najbardziej nowoczesny, zalecany do budowania biblioteki PDFBox, a przy tym jest nam najlepiej znany. Ułatwia proces budowania aplikacji, używając wtyczek, pobieranych automatycznie przy pierwszym zbudowaniu projektu. Maven wymaga połączenia z internetem przy pierwszej kompilacji, w celu pobrania źródeł bibliotek oraz dokumentu XML‑owego o nazwie POM, który zawiera wszystkie niezbędne informacje o projekcie i określa sposób budowy aplikacji. W pliku tym znalazły się wszystkie zależności, dotyczące tworzenia dokumentu PDF, pakiet RichTextFX [8], udostępniający zaawansowane pole tekstowe oraz SystemUtils - klasa, pozwalająca na pobranie nazwy systemu operacyjnego użytkownika.

### 5.2.6 RichTextFX

RichTextFX [8] udostępnia zaawansowane pole tekstowe dla technologii JavaFX. Pozwala ono na wprowadzenie do aplikacji markerów, które wyróżnią fragmenty tekstu bądź kodu kolorami, pogrubieniem, czy podkreśleniem. Umożliwia również dodanie numerowania linii.

### 5.2.7 CSS

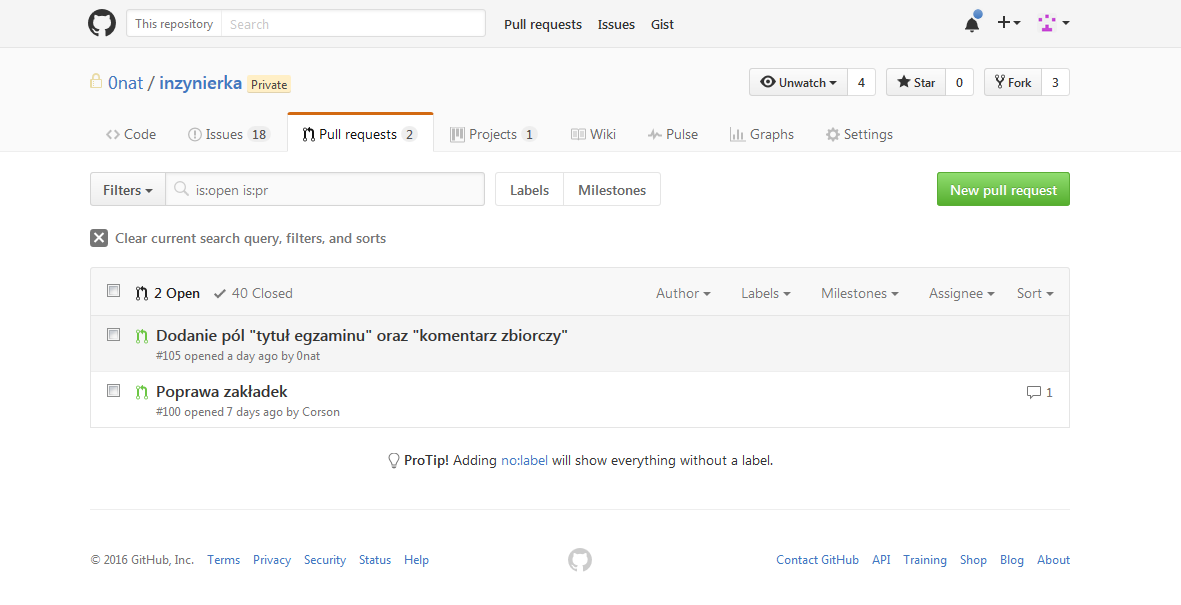
Do modyfikacji wyglądu elementów okien wykorzystaliśmy język CSS. Dzięki użyciu osobnego pliku zawierającego style, zostały one oddzielone od definicji interfejsu użytkownika.

## 5.3 Środowisko wytwórcze

### 5.3.1 System kontroli wersji

Zespół korzystał ze zdalnego repozytorium na serwisie GitHub (Rys 5.2.) do współdzielenia plików na komputerach i urządzeniach przenośnych. Przechowywane są tam zarówno dokumentacja, jak i cały projekt z kodem, dla zachowania wszystkich wersji plików, a zatem dla lepszego ich uporządkowania. Serwis udostępnia prywatne repozytoria dla studentów posiadających politechniczną skrzynkę pocztową, toteż nie było konieczne szukanie alternatywy. Wykorzystuje system kontroli wersji Git, który dostarcza wsparcia dla rozgałęzionego procesu tworzenia oprogramowania, dzięki czemu członkowie zespołu mogą pracować niezależnie, a wprowadzone przez nich zmiany są łączone z powrotem w jedną gałąź. Ponadto każdy z programistów posiada lokalną kopię repozytorium, zatem może pracować bez połączenia z internetem.

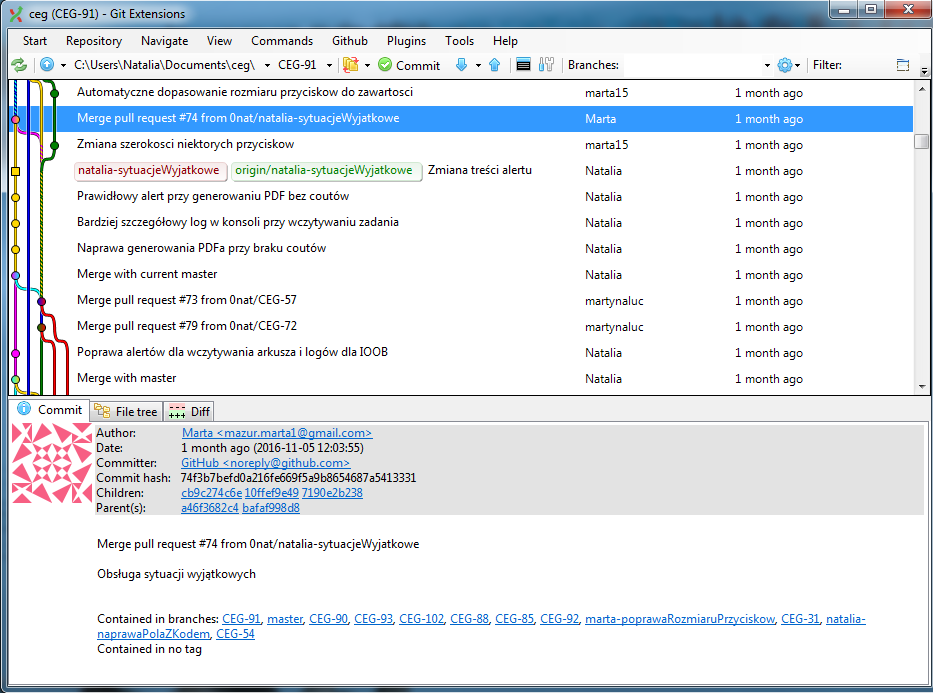
        Podczas naszej pracy, z powodu wielkości zespołu, postanowiliśmy nie włączać zmian z lokalnych gałęzi bezpośrednio na główną gałąź projektu, aby uniknąć dodawania nieprzetestowanych funkcjonalności. W tym celu każda dołączana funkcjonalność musiała znaleźć się na osobnej gałęzi, a następnie za pomocą opcji *pull request*, pozostali członkowie zespołu otrzymywali informację o prośbie o przetestowanie nowego fragmentu aplikacji. Jeśli funkcjonalność działała poprawnie, nowy kod był scalany z głównym kodem projektu.



Rys. 5.2. Zrzut ekranu przedstawiający zdalne repozytorium projektu inżynierskiego

na platformie GitHub

Aby ułatwić pracę z systemem Git, postanowiliśmy używać dodatku Git Extensions (Rys 5.3.). Jest to graficzny interfejs, pozwalający na wydawanie komend bez wpisywania ich do konsoli. Umożliwia wygodne przeglądanie historii commitów i plików oraz sprawdzenie, kto edytował daną część pliku jako ostatni.



Rys. 5.3. Zrzut ekranu przedstawiający dodatek Git Extensions

### 5.3.2 Zintegrowane środowisko programistyczne

Każdy z członków zespołu korzystał z wybranego przez siebie środowiska programistycznego dla języka Java, które pozwala na tworzenie i edycję kodu, jego kompilację i testowanie. Używanymi programami były NetBeans IDE i IntelliJ IDEA, w zależności od tego, z którym dana osoba była lepiej zaznajomiona.

### 5.3.3 Tworzenie diagramów

Do tworzenia diagramów wykorzystaliśmy darmowy internetowy edytor draw.io. Pozwala na tworzenie diagramów sekwencji działań, UML, związków encji, modeli i innych. Umożliwia ich zapis bezpośrednio na dysk Google Drive w taki sposób, że każda osoba posiadająca link do diagramu, może rozpocząć jego edycję jednym kliknięciem.

### 5.3.4 Narzędzia wspierające komunikację

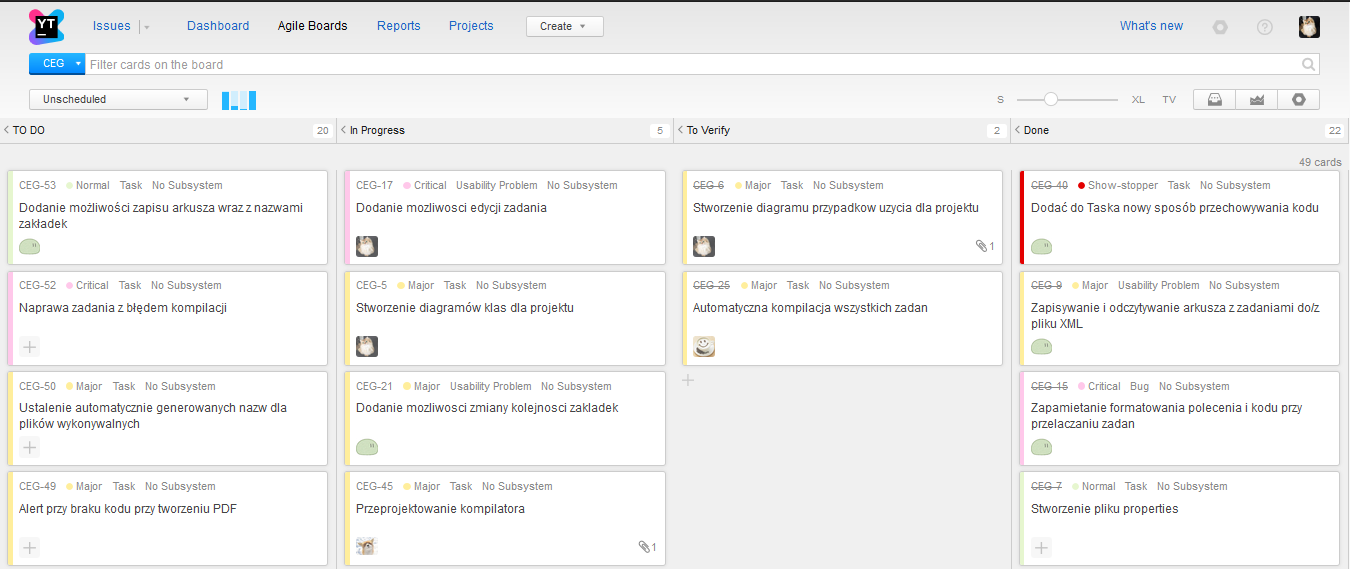
Narzędziami, które wykorzystywaliśmy do komunikacji między sobą był serwis internetowy Facebook oraz klient poczty e-mail. Pierwszy z nich pozwalał na utworzenie zamkniętej grupy dla projektu oraz komunikację za pomocą wbudowanej aplikacji Messenger. Drugi – umożliwił przekazywanie informacji między członkami zespołu i opiekunem pracy.

### 5.3.5 Narzędzia wspierające dokumentację

Aby móc tworzyć i równocześnie edytować dokumentację z dowolnego miejsca czy urządzenia, postanowiliśmy korzystać z Google Docs. Do edycji plików, które nie musiały być współdzielone w tym samym czasie, używaliśmy Microsoft Word 2013. Serwis GitHub udostępnił nam możliwość przechowywania dokumentacji wraz z jej wcześniejszymi wersjami.

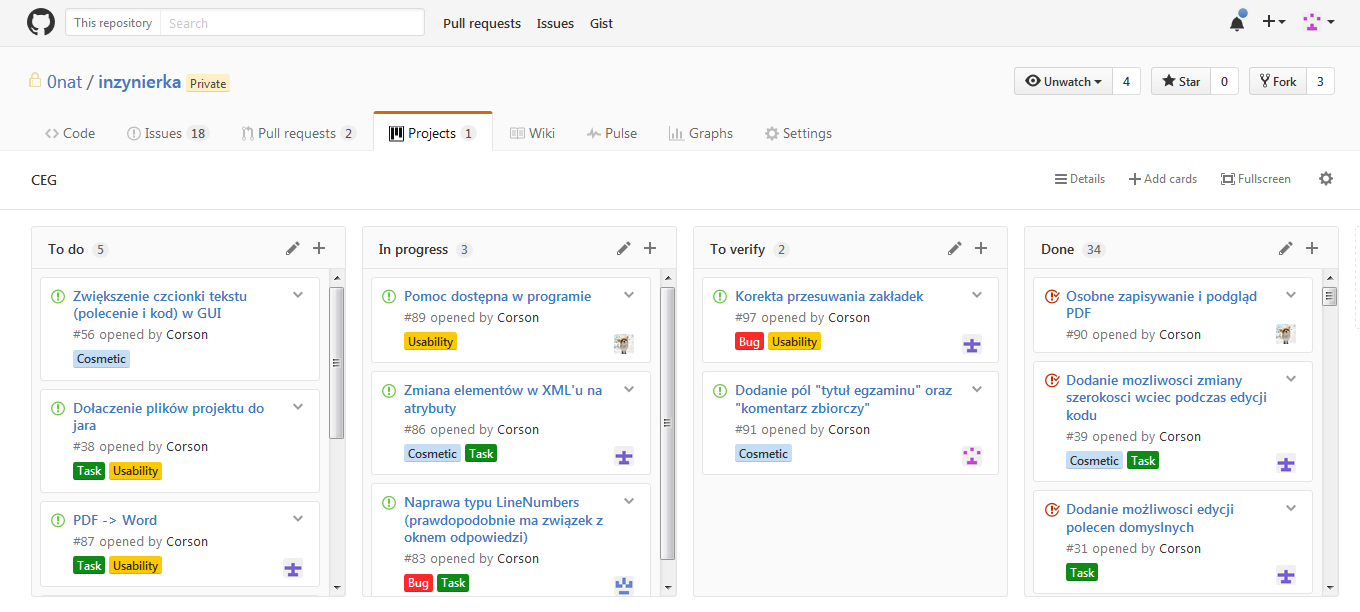
### 5.3.6 Narzędzia wspomagające organizację projektu

Do zaplanowania i śledzenia zadań początkowo używaliśmy darmowego, internetowego i konfigurowalnego narzędzia YouTrack (Rys 5.4.). Utworzone zostały przez nas zadania podobnej wielkości, które były dzielone pomiędzy członków zespołu na bieżąco. Na poniższym zrzucie ekranu widać, że strona jest przejrzysta, intuicyjna i wygodna – zadania można dowolnie przeciągać z jednego statusu do innego.



Rys. 5.4. Zrzut ekranu przedstawiający narzędzie YouTrack

Pomimo zalet oprogramowania YouTrack, zdecydowaliśmy się zmienić je na rzecz funkcjonalności dostępnej w serwisie GitHub. Powodem była wygoda przechowywania całości projektu w jednym miejscu.



Rys. 5.5. Zrzut ekranu przedstawiający zarządzanie zadaniami

na platformie GitHub

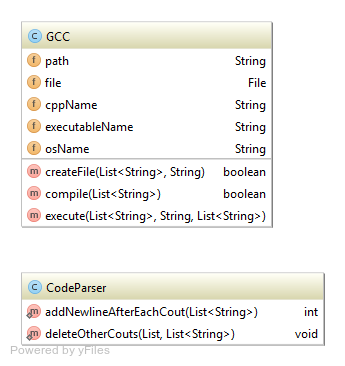
### 5.3.7 Narzędzia wspomagające testowanie

Aby przetestować aplikację na różnych systemach operacyjnych potrzebowaliśmy maszyn wirtualnych. Korzystaliśmy z menedżerów maszyn wirtualnych: VMWare Workstation Player, Workstation Pro oraz VirtualBox.

## 5.4 Struktura kodu

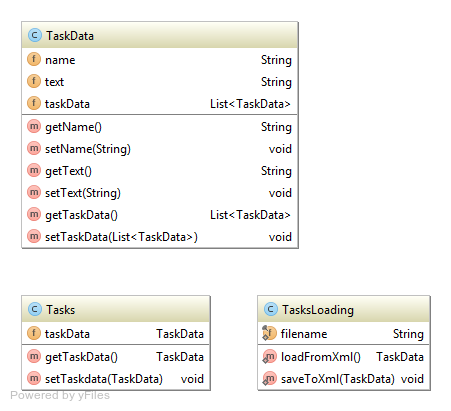
Projekt podzielony został na paczki, co ułatwiło organizację kodu i wspólną, równoległą pracę nad nim. Wyróżnione zostały klasy, obsługujące kompilator g++, które znalazły się w paczce o nazwie *compiler*. Kolejną paczką jest *examContent*, w której umieściliśmy klasy związane z zarządzaniem egzaminem i jego zadaniami. Własne wyjątki połączyliśmy w paczkę *exceptions,* natomiast alerty dla użytkownika, rodzaje czcionek, klasy statyczne i typy enum w paczkę *utils*. Ostatnimi paczkami były: *gui*, zawierająca kontrolery do widoków FXML, *pdf* do obsługi generowania arkusza w formacie PDF i jego wyglądu, *xml*, w której znalazły się klasy korzystające z frameworku JAXB, *fxml* z widokami, *properties* oraz *css*.

Poniższe diagramy (Rys. 5.6. i 5.7.) wygenerowane zostały za pomocą narzędzia w środowisku programistycznym IntelliJ IDEA. Pierwszy z nich pokazuje klasy obsługujące kompilację kodu źródłowego w zadaniach wraz z polami i metodami.



Rys. 5.6. Diagram klas dla paczki compiler

Drugi z diagramów prezentuje klasy, których zadaniem jest obsługa zapisu zadań do pliku w formacie XML, a także odczytu ze stworzonych plików.



Rys. 5.7. Diagram klas dla paczki xml

# 6. INSTRUKCJA UŻYTKOWNIKA

## 6.1 Instalacja i uruchomienie w systemie Windows

Upewnij się, że w systemie jest zainstalowane środowisko JRE 8. Rozpakuj folder z plikami programu w wybranym miejscu na dysku. Do zmiennej systemowej Path (System Properties > Advanced > Environment Variables) dodaj lokalizację folderu MinGW\bin, np. “C:\CExamGenerator\MinGW\bin;”. Przejdź w konsoli do folderu z programem i wywołaj polecenie *examGenerator* lub w przeglądarce plików kliknij dwa razy na plik ExamGenerator.jar.

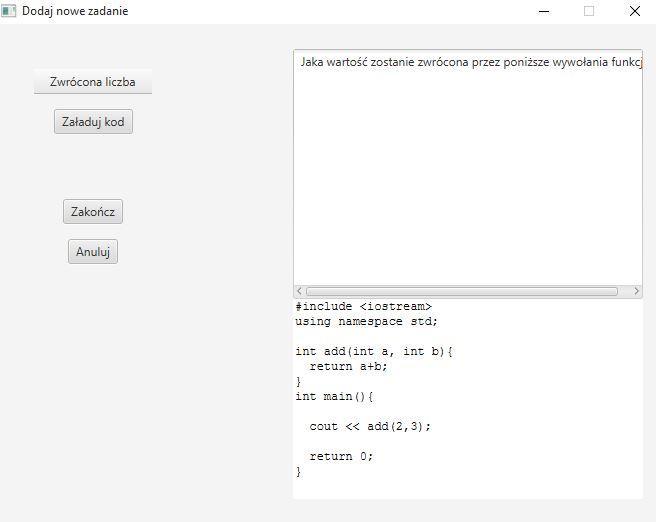
## 6.2 Instalacja i uruchomienie w systemie Linux

Upewnij się, że w systemie jest zainstalowane środowisko JRE 8 i kompilator g++ (dostępny m.in. dla systemu Fedora, CentOs czy OpenSUSE w pakiecie *gcc-c++,* a dla systemów Debian i Ubuntu w pakiecie *build-essential*). Rozpakuj katalog z plikami programu w wybranym miejscu na dysku. Przejdź w konsoli do katalogu z programem i wywołaj polecenie *./examGenerator.sh* lub w przeglądarce plików kliknij dwa razy na plik examGenerator.sh.

## 6.3. Zarządzanie zadaniami

### 6.3.1. Tworzenie zadania

Wybierz w menu okna głównego opcję Zadanie > Dodaj. Zostanie otwarte okno dodawania zadania. Z rozwijanej listy wybierz typ zadania. W oknie po prawej stronie pojawi się domyślne polecenie dla danego typu. Po kliknięciu przycisku „Wczytaj kod” możesz wybrać plik znajdujący się na dysku, z którego chcesz wczytać kod do zadania. Po kliknięciu przycisku „Zakończ” okno dodawania zadania zostanie zamknięte, a utworzone zadanie zostanie dodane do aktualnie otwartego egzaminu.



Rys. 6.1. Okno dodawania zadania

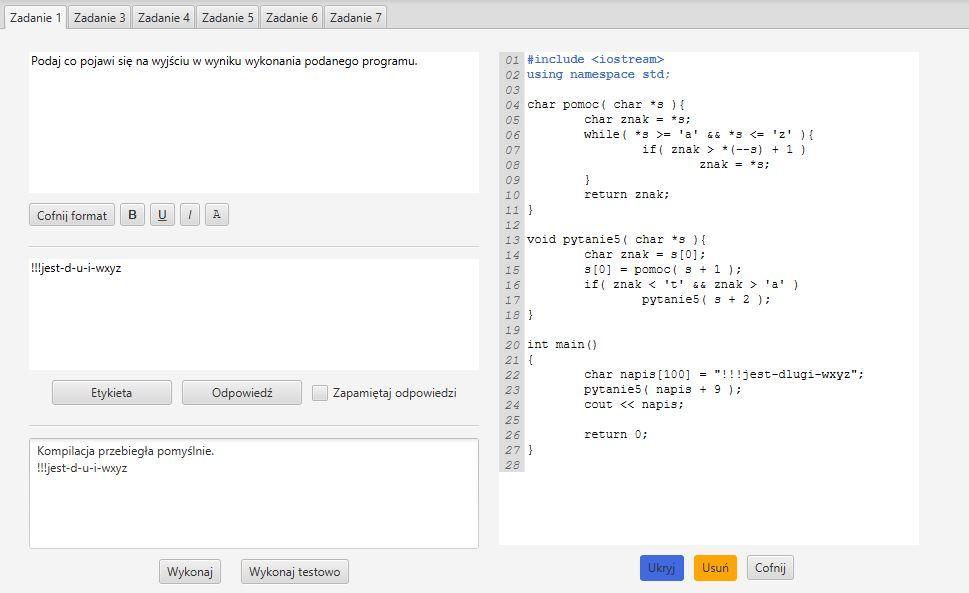
### 6.3.2. Edycja zadania

Po wybraniu z menu okna głównego opcji Zadanie > Edytuj zostanie otwarte okno edycji zadania, identyczne z oknem dodawania zadania (patrz: Tworzenie zadania).

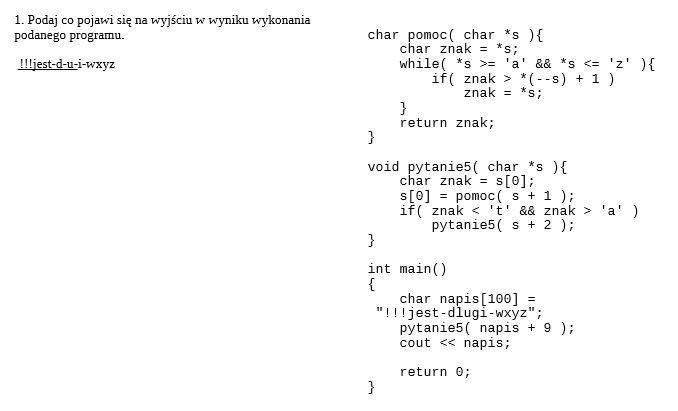
## 6.4. Dostępne typy zadań

### 6.4.1. Wyjście programu

W tym typie zadania wszystko, co pojawi się na wyjściu programu do znaku nowej linii jest traktowane jako jedna odpowiedź. Kolejne linie wyjścia będą ignorowane (stąd liczba odpowiedzi do tego typu zadania zawsze wynosi 1).



Rys. 6.2. Edycja zadania typu „Wyjście programu”

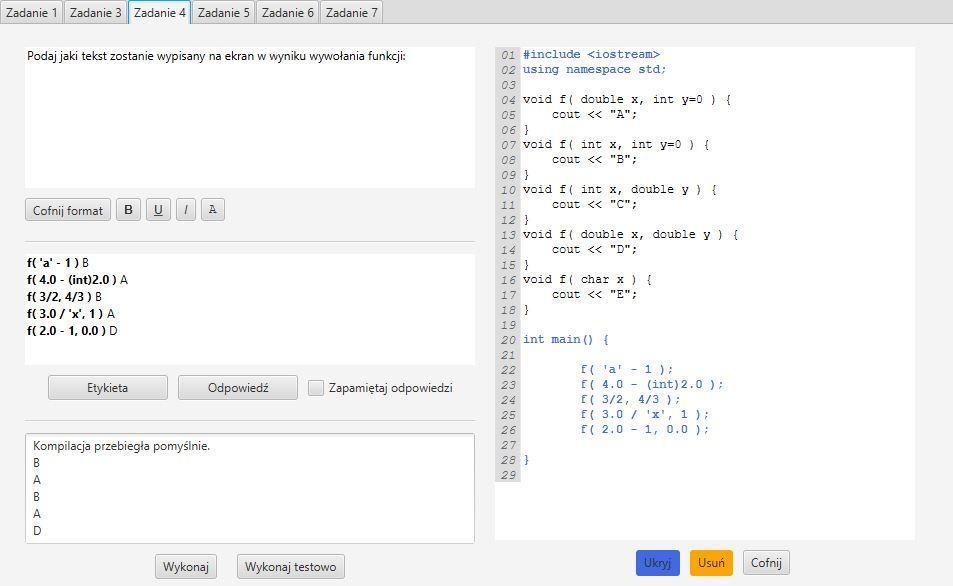


Rys. 6.3. Zadanie typu „Wyjście programu” w arkuszu

Część kodu niepotrzebną do zrozumienia treści polecenia zaznaczamy niebieskim markerem „Usuń”. Nie będzie ona widoczna w arkuszu PDF. Domyślne polecenie pozostawiamy bez zmian. Pole „Zapamiętaj odpowiedzi” pozostawiamy odznaczone, co pozwoli na automatyczne wygenerowanie poprawnych odpowiedzi po wykonaniu kodu dołączonego do zadania.

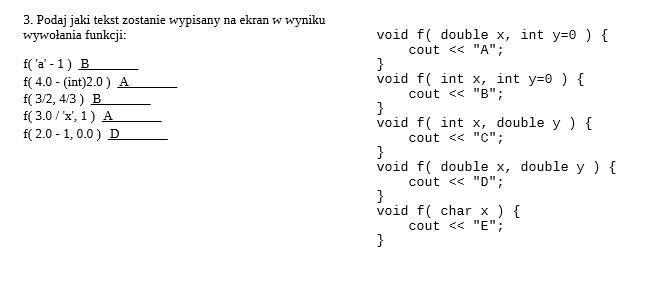
### 6.4.2. Wyjście funkcji

W tym typie zadania poszczególne wywołania funkcji printf / cout są rozdzielane znakami nowej linii. Każda linia wyjścia jest traktowana jako osobna odpowiedź w egzaminie. Domyślna liczba odpowiedzi jest równa liczbie linii wyjścia.



Rys. 6.4. Edycja zadania typu „Wyjście funkcji”

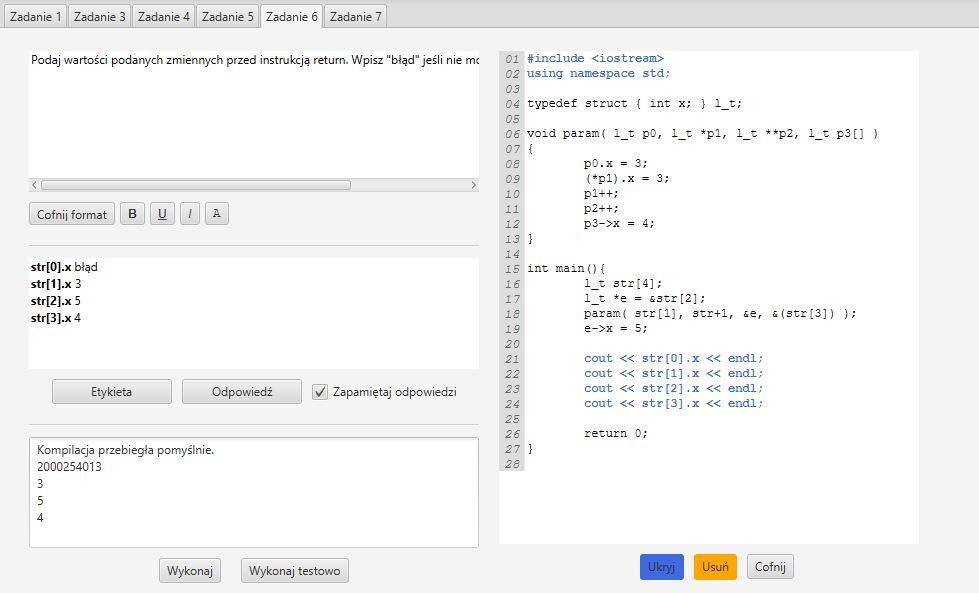
Część kodu niepotrzebną do zrozumienia treści zadania ukrywamy niebieskim markerem „Usuń”. Nie będzie ona widoczna w arkuszu PDF. W polu z odpowiedziami (środkowe pole po lewej stronie) pogrubione fragmenty tekstu pojawią się w arkuszu PDF. Pole „Zapamiętaj odpowiedzi” pozostaje odznaczone, dzięki czemu poprawne odpowiedzi zostaną wygenerowane po wykonaniu kodu. Etykiety pozostaną niezmienione.



Rys. 6.5. Zadanie typu „Wyjście funkcji” w arkuszu

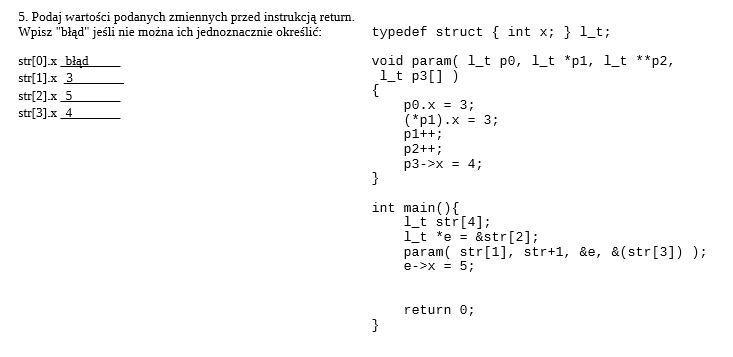
### 6.4.3. Wartość zmiennej

W kodzie dołączonym do zadania tego typu należy umieścić przynajmniej jedno wypisanie wartości zmiennej (np. std::cout << x;). Każda linia wyjścia będzie traktowana jako osobna odpowiedź (tak jak dla typu „Wyjście funkcji”). Aby instrukcja wypisania wartości zmiennej nie była widoczna w arkuszu PDF, należy zaznaczyć ją markerem „Ukryj” (patrz przykład).



Rys. 6.6. Edycja zadania typu "Wartość zmiennej"

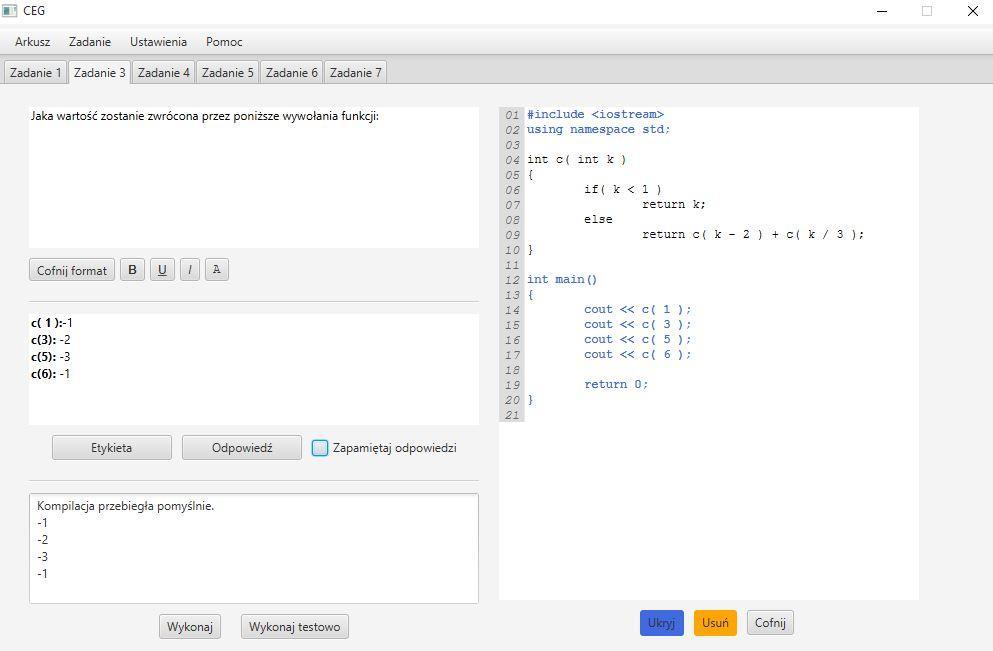
W kodzie dołączonym do zadania zostają umieszczone instrukcje *cout,* aby możliwe było pobranie przez program wyjścia i wygenerowanie poprawnych odpowiedzi. W polu z kodem „ukrywamy” za pomocą niebieskiego markera fragmenty, które nie muszą znaleźć się w arkuszu PDF. Domyślne polecenie zostaje zmienione tak, aby pasowało do dołączonego kodu (wartości zmiennych są sprawdzane przed instrukcją *return*). Zmieniona zostaje również pierwsza z wygenerowanych odpowiedzi. Żeby zmiana została uwzględniona, należy zaznaczyć pole „Zapamiętaj odpowiedzi”. Pogrubione fragmenty tekstu w polu z odpowiedziami zostaną wydrukowane w arkuszu PDF jako etykiety opisujące odpowiedzi.



Rys. 6.7. Zadanie typu "Wartość zmiennej" w arkuszu

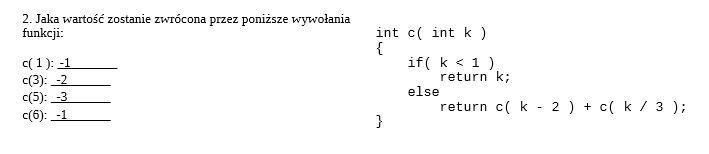
### 6.4.4. Zwrócona wartość

W kodzie dołączonym do zadania tego typu należy umieścić wypisanie wartości zwróconej przez funkcję (np. std::cout << f(100);). Każda linia wyjścia będzie traktowana jako osobna odpowiedź (tak jak dla typu „Wyjście funkcji”). Aby instrukcja wypisania wartości zwróconej przez funkcję nie była widoczna w arkuszu PDF, należy zaznaczyć ją markerem „Ukryj” (patrz przykład).



Rys. 6.8. Edycja zadania typu "Zwrócona wartość"

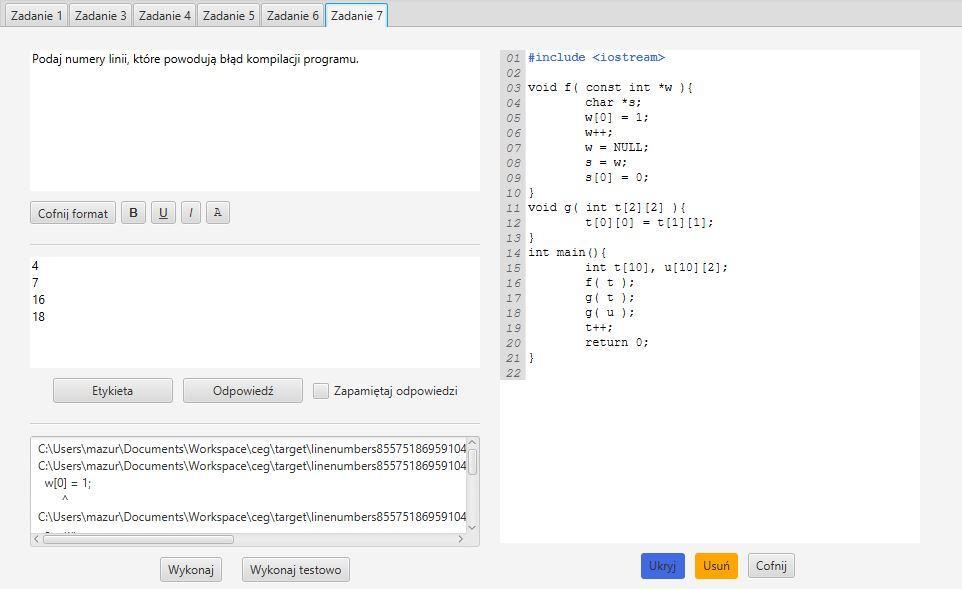
W kodzie dołączonym do zadania zostają umieszczone instrukcje *cout,* aby możliwe było pobranie przez program wyjścia i wygenerowanie poprawnych odpowiedzi. W polu z kodem „ukrywamy” za pomocą niebieskiego markera fragmenty, które nie muszą znaleźć się w arkuszu PDF. Domyślne polecenie pozostaje niezmienione. W polu z odpowiedziami (środkowe pole po lewej stronie) pogrubione fragmenty tekstu pojawią się w arkuszu PDF. Pole „Zapamiętaj odpowiedzi” pozostaje odznaczone, dzięki czemu poprawne odpowiedzi zostaną wygenerowane po wykonaniu kodu. Etykiety pozostaną niezmienione.



Rys. 6.9. Zadanie typu "Zwrócona wartość" w arkuszu

### 6.4.5. Numery linii

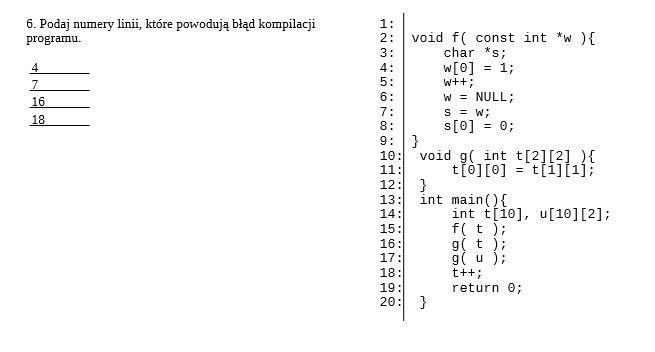
Kod dołączony do zadania tego typu powinien powodować błędy kompilacji. Odpowiedziami do tego zadania są numery linii powodujących błędy kompilacji (odpowiadające tym z arkusza PDF, nie z pola z kodem w programie).



Rys. 6.10. Edycja zadania typu "Numery linii"

Kod dołączony do tego zadania powoduje błędy. Domyślne polecenie pozostało niezmienione. Fragment kodu, który nie musi znaleźć się w arkuszu PDF został zaznaczony niebieskim markerem „Ukryj”.

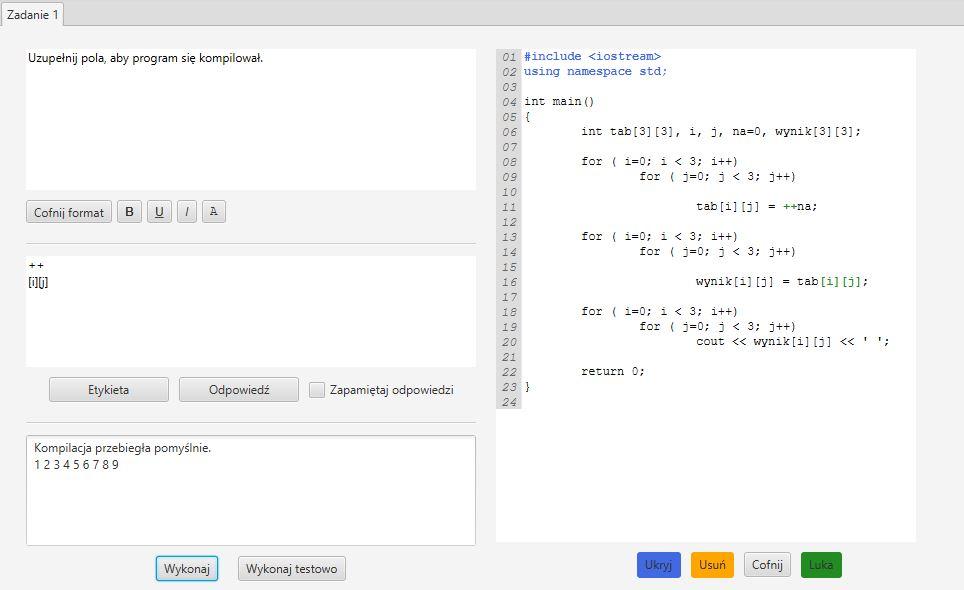
Uwaga - przy tym typie zadania należy pamiętać, aby zawsze sprawdzić poprawność wygenerowanych odpowiedzi i w razie błędów poprawić je i zaznaczyć pole „Zapamiętaj odpowiedzi”.  Generowanie odpowiedzi polega na interpretacji wyjścia kompilatora, który czasami może wskazać linię, w której został wykryty błąd, a nie linię będącą przyczyną błędu.



Rys. 6.11. Zadanie typu "Numery linii" w arkuszu

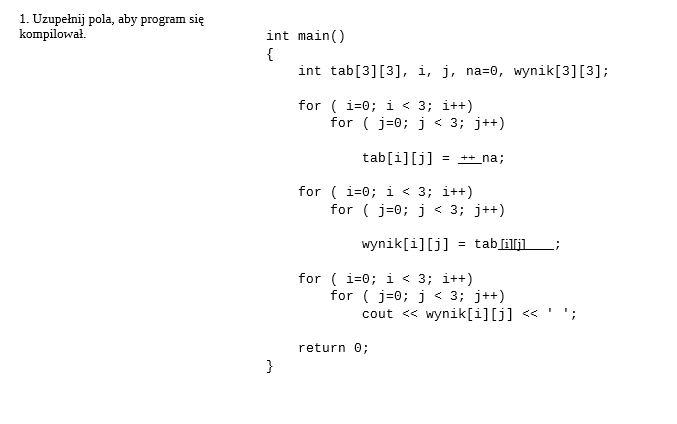
### 6.4.6. Uzupełnianie luk

Dla tego typu zadania dostępny jest dodatkowy marker. Zaznaczone fragmenty kodu będą stanowiły odpowiedzi do zadania (każde zaznaczenie to jedna odpowiedź). W arkuszu dla studenta w miejsce zaznaczonych fragmentów pojawią się luki (poziome linie).



Rys. 6.12. Edycja zadania typu „Uzupełnianie luk”

W polu z kodem „ukrywamy” za pomocą niebieskiego markera fragmenty, które nie muszą znaleźć się w arkuszu PDF, a za pomocą zielonego zaznaczamy fragmenty, które mają zostać zamienione w luki do uzupełnienia. Polecenie i odpowiedzi pozostawiamy bez zmian.



Rys. 6.13. Zadanie typu „Uzupełnianie luk” w arkuszu

### 6.4.7. Typ własny

Dla tego typu zadania nie są generowane odpowiedzi, nie ma też domyślnego polecenia. Użytkownik może stworzyć dowolne odpowiedzi, gdyż nie są one w żaden sposób zależne od wprowadzonego kodu.

## 6.5. Markery

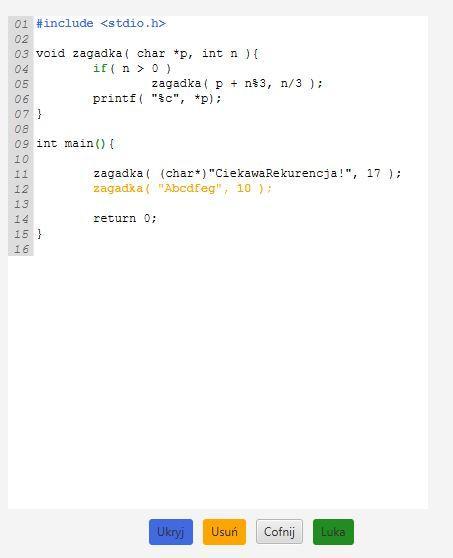
Dla wszystkich typów zadań dostępne są dwa markery: „Usuń” i „Ukryj”.

„Usunięty” kod nie pojawi się w arkuszu PDF i nie zostanie skompilowany podczas zwykłego wykonania. Zostanie skompilowany w przypadku wykonania testowego. Tego markera można użyć np., gdy potrzebne są dodatkowe „testowe” komunikaty wypisywane na ekran, których nie należy brać pod uwagę podczas generowania odpowiedzi.

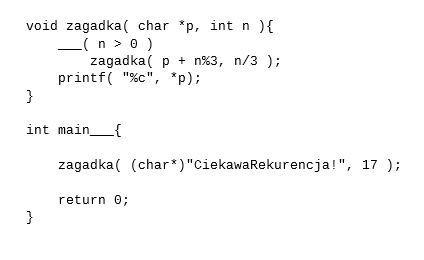
„Ukryty” kod nie pojawi się w arkuszu PDF, natomiast zostanie skompilowany zarówno dla wykonania zwykłego, jak i testowego. Tego markera można użyć np. gdy fragment kodu jest konieczny do poprawnej kompilacji, ale nie powinien być zamieszczony w arkuszu egzaminacyjnym, gdyż pogarszałby czytelność (np. dyrektywy *#include*).

Dla typu „Uzupełnianie luk” dostępny jest dodatkowy marker „Luka” (patrz: Dostępne typy zadań > Uzupełnianie luk).

Przycisk „Cofnij” powoduje wyczyszczenie markerów dla zaznaczonego fragmentu kodu.



Rys. 6.14. Edycja kodu z użyciem markerów



Rys. 6.15. Wynik edycji kodu z użyciem markerów

## 6.6. Okno z odpowiedziami

Po wykonaniu kodu dołączonego do zadania w tym oknie wyświetlane są wygenerowane przez program odpowiedzi (każda w osobnej linii), które można edytować. Aby w arkuszu pojawiły się zmienione odpowiedzi, należy zaznaczyć checkbox „Zapamiętaj odpowiedzi”, w przeciwnym wypadku wprowadzone przez użytkownika zmiany zostaną nadpisane przez odpowiedzi wygenerowane automatycznie. Dodatkowo dostępny jest marker „Etykieta”. Fragment tekstu w oknie odpowiedzi oznaczony tym markerem (pogrubiony) zostanie wydrukowany w arkuszu PDF jako etykieta opisująca odpowiedź – nie będzie częścią odpowiedzi, natomiast pojawi się bezpośrednio przed nią zarówno w wersji dla nauczyciela, jak i studenta. Przycisk „Odpowiedź” służy do cofania zaznaczenia.

## 6.7. Wykonanie i wykonanie testowe

Po kliknięciu przycisku „Wykonaj” nastąpi kompilacja kodu z wykluczeniem kodu „usuniętego” za pomocą pomarańczowego markera. Przycisk „Wykonaj testowo” uruchamia kompilację całego kodu.

Następnie w obu przypadkach skompilowany program zostanie uruchomiony, a na podstawie jego wyjścia zostaną wygenerowane i wypisane poprawne odpowiedzi. Wyjście programu lub ewentualne błędy kompilacji czy wykonania zostaną wypisane w przeznaczonym do tego polu.

## 6.8. Zarządzanie egzaminem

### 6.8.1. Menu główne

W menu w oknie głównym programu dostępne są takie opcje zarządzania egzaminem jak: dodanie zadania, usunięcie zadania, wczytanie / zapis zadania i egzaminu do pliku XML.

Zmiana szerokości zadania w PDF – po wybraniu w menu okna głównego opcji Zadanie > Zmień szerokość w PDF otwiera się nowe okno z suwakiem, dzięki któremu można ustawić, ile procent szerokości arkusza zajmie polecenie, a ile kod dołączony do zadania.

Dodanie numerowania linii kodu – po zaznaczeniu tej opcji w menu głównym (Zadanie > Dodaj numerowanie linii) w arkuszu PDF zostaną wydrukowane numery linii kodu. Opcja ta jest domyślnie zaznaczona dla typu zadania “Numery linii”.

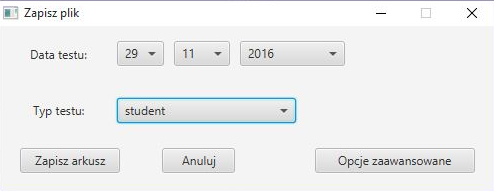
### 6.8.2. Menu kontekstowe

W menu kontekstowym dostępne jest dodawanie nowego zadania, usuwanie aktywnego zadania, zmiana nazwy aktywnego zadania i zmiana kolejności zadań (przesuwanie aktywnego zadania o jedną pozycję w lewo lub w prawo).

## 6.9. Generowanie PDF

### 6.9.1. Ustawienia arkusza

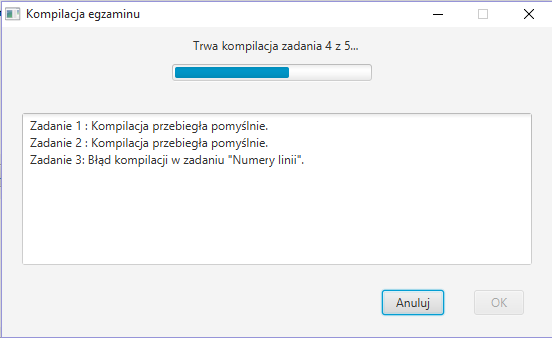
Po wybraniu w menu opcji Arkusz > Generuj PDF dostępne są podstawowe ustawienia arkusza - data, która pojawi się w nagłówku arkusza oraz typ arkusza (wersja dla nauczyciela lub dla studenta). Pozostałe ustawienia dostępne są w oknie opcji zaawansowanych.



Rys. 6.16. Okno ustawień zapisu arkusza PDF

### 6.9.2. Okno kompilacji egzaminu

Po wybraniu ustawień arkusza i miejsca zapisu pliku PDF uruchomiona zostanie kompilacja wszystkich programów dołączonych do zadań. Informacja o poprawnej kompilacji lub ewentualnych błędach pojawi się w oknie kompilacji egzaminu.



Rys. 6.17. Okno kompilacji egzaminu

### 6.9.3. Warunki poprawnej generacji egzaminu

Aby możliwe było poprawne wygenerowanie arkusza egzaminacyjnego, egzamin musi spełniać następujące warunki:

* Nie może być pusty (musi posiadać co najmniej jedno zadanie).
* Wszystkie zadania muszą posiadać niepuste pola z kodem i poleceniem.
* Kod dołączony do wszystkich zadań musi być kompilowany (z wyjątkiem zadań typu  „Numery linii” i typu własnego).
* Liczba odpowiedzi do każdego zadania musi być większa od 0, stąd programy dołączone do zadań muszą wypisywać na wyjście co najmniej jedną linię ( z wyjątkiem zadań typu „Numery linii” i typu własnego).

## 6.10. Opcje zaawansowane

### 6.10.1. Wygląd PDF

W tej zakładce możliwa jest zmiana czcionki dla polecenia i kodu w arkuszu PDF. Dostępnych jest kilka standardowych czcionek. Można również zmienić typ, kolor i formatowanie czcionki dla poprawnych odpowiedzi w arkuszu dla nauczyciela.

### 6.10.2. Dodatkowe opcje PDF

W tej zakładce można dodać lub edytować tytuł i komentarz zbiorczy do egzaminu, które pojawią się na pierwszej stronie arkusza PDF. Tutaj znajduje się też opcja “Dodaj separatory po każdym zadaniu”, po zaznaczeniu której zadania w wygenerowanym arkuszu PDF będą rozdzielone poziomymi liniami.

### 6.10.3. Limit czasu wykonania

Za pomocą suwaka można ustawić limit czasu wykonania jednakowy dla wszystkich programów dołączanych do zadań. Limit ten może wynosić od 1 do 10 sekund. Obejmuje on jedynie wykonanie programu, czas kompilacji i generowania odpowiedzi nie jest ograniczony.

### 6.10.4. Polecenia domyślne

W tej zakładce można edytować domyślne polecenia dla poszczególnych typów zadań, wybieranych z rozwijanej listy.

# 7. TESTY

Testowanie to jeden z bardzo istotnych elementów wytwarzania oprogramowania. Pozwala na sprawdzenie czy aplikacja działa dokładnie tak, jak zostało to określone w wymaganiach funkcjonalnych i wydajnościowych. Inaczej można powiedzieć, że jest to proces, który sprawdza czy została zachowana jakość oprogramowania. Generator testów egzaminacyjnych z języka C++ od samego początku miał być nastawiony na jak najmniejszą liczbę błędów w funkcjonowaniu, dlatego też był to dość istotny etap tworzenia aplikacji. Ze względu na specyfikę produktu i częstotliwość jego użytkowania przez prowadzącego przedmiot nie zostały określone wymagania wydajnościowe, dlatego też nie było potrzeby przeprowadzania takich testów.

Testowanie generatora zostało podzielone na kilka etapów. Najpierw przeprowadzone zostały testy weryfikujące wymagania użytkownika na podstawie prototypu. Ich celem było wstępne określenie czy zaproponowany interfejs użytkownika odpowiada prowadzącemu przedmiot Podstawy Programowania. Kolejna część testów dotyczyła działania aplikacji i jej wyglądu na wielu systemach operacyjnych sprawdzone zostało wymaganie przenośności oprogramowania pomiędzy systemami z rodziny Windows i Linux. W czasie trwania pracy programistycznej przeprowadzane były testy bieżące, zarówno przez programistę wykonującego dane zadanie, jak i dwóch niezwiązanych z danym zadaniem (tzw. testy alfa), zgodne z przyjętą metodyką zarządzania projektem. Na koniec klient przeprowadził testy akceptacyjne mające na celu weryfikację zgodności programu z wymaganiami (tzw. testy beta).

## 7.1. Testy precyzujące wymagania

Na wstępie, aby poprawnie zdefiniować wymagania prowadzącego, zaimplementowany został prototyp, który zawierał projekt interfejsu użytkownika, wstępnie działające niektóre typy zadań, wbudowaną obsługą zewnętrznego kompilatora oraz projektem szkieletu wyglądu dokumentu PDF. Trafił on do testów wykonywanych przez użytkownika, w celu weryfikacji, czy wstępna wersja zgadza się z jego oczekiwaniami i wymaganiami dotyczącymi aplikacji oraz wymaganiami dotyczącymi wygody korzystania z interfejsu użytkownika, co jeszcze powinno zostać według niego zmienione i jakie funkcjonalności dodatkowo uważa za niezbędne.

Wynikiem tych testów była wstępna akceptacja interfejsu użytkownika, projektu podziału typów zadań, wyglądu dokumentu a także wypisywania wyniku kompilacji i wykonania do interfejsu aplikacji, jednak dołączone zostały dodatkowe wymagania, które zostały zdefiniowane dopiero na tym etapie.

Wśród tych wymagań można wymienić możliwość dodania formatowania polecenia w postaci pogrubienia, kursywy i podkreślenia istotnych treści w dokumencie PDF w celu zwiększenia czytelności egzaminu dla studenta.

Użytkownik uważał również za konieczne dodanie markerów, które wskażą, która część kodu ma znaleźć się na dokumencie PDF, która część nie będzie brana pod uwagę w czasie automatycznego generowania odpowiedzi przez kompilator, jednak nie precyzował wykonania tej funkcjonalności, można było wygrać najlepszą według programistów.

Następnym zdefiniowanym na tym etapie wymaganiem była możliwości dodania etykiet dla każdej odpowiedzi w miejscu na odpowiedź w dokumencie PDF, w celu możliwości identyfikacji, do której linii kodu dana odpowiedź się odnosi.

Z tego samego powodu użytkownik prosił również o dodanie numerów linii do typu zadania opierającego się na numerach linii, w których występuje błąd kompilacji. Według wcześniejszej implementacji, student musiałby liczyć, w której linii występuje błąd, co mogłoby prowadzić do wielu błędów popełnianych przez uzupełniających test, pomimo że znali oni poprawną odpowiedź.

Podsumowując ten rodzaj testowania, można uznać, że testy wypadły pomyślnie i pomogły już na tym wczesnym etapie zweryfikować wymagania funkcjonalne i dodać elementy, które nasunęły się użytkownikowi jako konieczne w tym testowaniu.

## 7.2. Testy wstępne dotyczące działania aplikacji na różnych systemach operacyjnych

Jednym z najważniejszych wymagań, które należało spełnić, to przenośność aplikacji pomiędzy systemami z rodziny Linux  i systemami z rodziny Windows. Aby spełnić to wymaganie konieczne było przeprowadzenie testów działania programu na wczesnym etapie produkcji oprogramowania, w celu szybkiej identyfikacji ewentualnych problemów z niezgodnością. Zatem ta część testów została przeprowadzona, gdy aplikacja posiadała tylko możliwość dodawania i usuwania zadań, generowania dokumentu z możliwością wyboru czcionki i jej rozmiaru a także kompilacji zadań. Testy zostały przeprowadzone przy pomocy maszyn wirtualnych i zainstalowanych na nich systemach operacyjnych nie zawierających żadnego dodatkowego oprogramowania a także narzędzia umożliwiającego wirtualizację – VMWare Player lub VirtualBox. Systemy, które zostały wybrane do testów to systemy operacyjne takie jak: Linux Ubuntu 16.04 LTS 64-bit., Linux Fedora 24 64 bit., Windows 7 Professional 64 bit., Windows 8.1 Professional 64 bit., Windows 10 64 bit., Windows 8.1 32-bit., Windows 7 Proffessional, Service Pack 1, 64 bit. Przed rozpoczęciem testowania przygotowany został szablon, na podstawie którego sprawdzane było czy aktualnie zaimplementowane funkcjonalności działają na każdym z systemów. Funkcjonalność w szablonie została również podzielona na tę, która jest niezależna od systemu operacyjnego oraz tę, która jest od niego zależna. Testowanie trwało tydzień. Wyniki wraz z zaznaczeniem, kto wykonywał dany test, jakiego systemu dotyczył i jaką konfigurację zastosował zostały dokładnie przeanalizowane a błędy poprawione.

W ramach tych testów przeprowadzone zostały również testy interfejsu graficznego. Dotyczyły one konieczności zablokowania czy zmiany widoczności niektórych przycisków w zależności od działań użytkownika w celu uniknięcia prób wykonania przez użytkownika akcji, które w danym stanie tworzenia egzaminu są niedostępne. Przykładowo, opcja edycji czy zapisu zadania jest dostępna tylko w przypadku, gdy istnieje jakieś zadanie.

Wynik przeprowadzonego testu spowodował konieczność wprowadzenia wielu zmian w systemie a także konieczność stworzenia instrukcji w formie kroków, która pokaże jak poprawnie zainstalować wymaganą wersję Javy na systemach z rodziny Linux, gdyż nie była to konfiguracja oczywista.

Wśród błędów, które zostały wykryte w czasie testowania można wyróżnić błędy związane z kodowaniem polskich znaków w plikach tekstowych, z których wczytywane były polecenia domyślne, gdyż jest ono różne na systemach z rodziny Windows (kodowanie Windows-1252) i Linux (kodowanie UTF-8). Naprawienie tego błędu wiązało się ze zmianą sposoby przechowywania tychże poleceń na zapisywanie ich w plikach xml. Jednocześnie ułatwiło to implementację edycji poleceń domyślnych z poziomu aplikacji użytkownika.

Kolejnym wykrytym błędem były błędy przy usuwaniu zakładek. Błąd ten występował na obu rodzinach systemów operacyjnych. Przykładowo, przy usuwaniu zadań od pierwszego, gdy było ich więcej niż trzy, ostatnie zadanie nie zostawało usuwane - chowała się za to jego treść, a zakładka zostawała. Błąd ten wynikał z błędów implementacyjnych.

Dość istotnym błędem wykrytym w czasie testów był brak komunikatów o wystąpieniu błędów, ich obsługi, brak informacji o braku możliwości wygenerowania dokumentu z powodu braku którejś z jego części. Podobnie jak poprzedni, był to błąd występujący na obu systemach operacyjnych. Problem został naprawiony poprzez stworzenie klasy Alerts, która ujednoliciła wyświetlanie użytkownikowi występujących błędów oraz stworzone zostały jednolite komunikaty w konsoli w języku angielskim.

Wystąpił również problem z generowaniem odpowiedzi w wersji dla nauczyciela z powodu wprowadzenia do interfejsu użytkownika komunikatu o poprawnej kompilacji zadania przed wyświetleniem wyniku wykonania kodu pobranego z kompilatora. Zostało to naprawione poprzez pominięcie linii o poprawnej kompilacji w momencie ustawiania odpowiedzi dla zadania po jego kompilacji.

Do błędów wynikających z testów interfejsu użytkownika aplikacji w przypadku systemów z rodziny Linux, okazywało się, że często tekst widoczny na przyciskach, etykietach czy opcjach w menu nie mieści się w wyznaczonym miejscu z powodu stosowania innej czcionki domyślnej (w systemie Linux - Sans) niż na systemach z rodziny Windows (Segou UI). Tutaj wprowadzone zostało rozszerzanie przycisków zależnie od tekstu w nich się znajdujących.

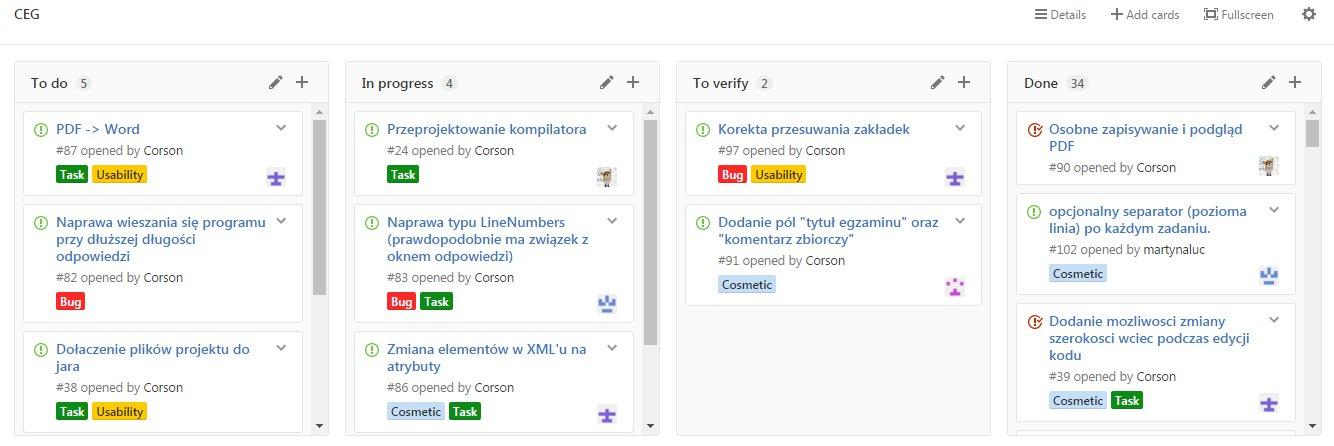
Pozostałe znalezione błędy były na tyle nieduże, że poprawione zostały praktycznie od razu i wynikały z drobnych pomyłek programisty. Można zatem uznać, że testy wypadły pomyślnie, nie było więc podstaw do zatrzymania się na tym etapie, więc rozpoczęte zostało programowanie zaawansowanych okien użytkownika bez obaw, że aplikacja testowana po napisaniu na różnych systemach operacyjnych będzie działała znacząco różnie lub nie zadziała w ogóle na innym systemie niż ten, na którym pracowali i testowali programiści (głównie system Windows).

## 7.3. Testy bieżące (testowanie alfa)

Z czasem, gdy kodu pojawiało się coraz więcej, coraz trudniej było go utrzymywać tak, aby nie występowała duża liczba błędów. Wcześniej już zapadła decyzja o korzystaniu z prywatnego repozytorium studenckiego GitHub, jednak nie były wykorzystywane wszystkie jego możliwości. Postanowiliśmy jednak wykorzystać wszystkie jego możliwości również do testowania aplikacji na bieżąco. Utworzyliśmy własną tablicę z czterema statusami do wyboru: do zrobienia, w czasie pracy, do zweryfikowania, gotowe. Testowanie alfa rozpoczynało się w momencie, gdy programista piszący dane zadanie zmienił status zadania na: do zweryfikowania. W tym momencie, dwóch innych uczestników zespołu, którzy nie programowali  danego zadania, sprawdzało czy kod jest napisany zgodnie z dobry praktykami programowania, czy obsługuje wszystkie możliwe do wystąpienia błędy oraz testowało czy kod realizuje funkcję dokładnie opisaną w zadaniu. Każde zadanie tworzone było na odrębnej gałęzi, przez to, jeżeli któryś z programistów sprawdzających stwierdził, że zadanie zawiera błędy, pisał o występującym problemie w celu jego poprawy i zostawało ono zwrócone do poprawki programiście, który je pisał. Wiązało się to również z tym, że dopóki zadanie nie zostało zatwierdzone przez dwóch programistów to nie miało prawa zostać dołączone do głównej gałęzi. To spowodowało znaczne podniesienie jakości kodu oraz testowanie aplikacji na bieżąco, po każdym zadaniu i  nawet, jeśli błąd nie został wykryty w czasie bezpośredniego sprawdzania tego zadania, to zostawał znaleziony w momencie testowania innych zadań, dzięki czemu można było ustrzec się większości błędów.

Ponieważ w ten sposób pozbyliśmy się praktycznie większości błędów aplikacji to uznaliśmy, że nie ma konieczności wprowadzania testów automatycznych, gdyż ich stworzenie byłoby czasochłonne i nie prowadziłoby w rezultacie do znacznego zmniejszenia liczby błędów, gdyż aplikacja nie posiadała ich aż tylu. Poza tym, testy automatyczne stworzone przez programistę mogą również zawierać błędy i można by stracić bardzo wiele czasu próbując dowiedzieć się czy błąd pochodzi faktycznie z aplikacji, czy jednak jest to błąd testów. Poza tym, testowanie automatyczne wymaga również utrzymywania testów w zależności od zmian wprowadzonych w aplikacji, dlatego w przypadku Generatora testów egzaminacyjnych z języka C++ nie był to element konieczny.

Poniższy rysunek (Rys. 7.1.) przedstawia tablicę zadań na GitHub. Jak można zauważyć, aktualnie widać dwa zadania, które należy przetestować. Mimo zabezpieczenia w postaci podwójnego testowania przez programistów może się zdarzyć, że nadal występują błędy. W momencie ich znalezienia, znajdujący tworzy nowe zadanie z etykietą „Bug”, oznaczającą błąd. Na poniższym rysunku możemy zauważyć trzy właśnie w ten sposób oznaczone błędy. Jeden w statusie do zrobienia, drugi w trakcie pracy i trzeci do zweryfikowania przez innych użytkowników.

Rys. 7.1. Wygląd tablicy zadań na GitHub

Zadanie przechodzi do testów w momencie ukończenia jego programowania i metodą „przeciągnij i upuść” przeniesienia do statusu do zweryfikowania a także jednocześnie utworzenia „Pull Request”, czyli żądania dołączenia kodu do kodu głównego. Wtedy programiści, którzy sprawdzają poprawność rozwiązania danego zadania mogą w prosty sposób zweryfikować kod na podstawie zmian widocznych za zakładce *File changed* na poniższym rysunku. Znajduje się tam spis linii, które zostały zmienione (dodane lub usunięte) przez programistę w celu realizacji danego zadania. Zmiany niezaakceptowane wracają do programisty w celu ich poprawy. Po zrobieniu inspekcji kodu programista testujący zadanie uruchamia kod zmieniony na własnym komputerze i za pomocą testowania manualnego sprawdza możliwie jak największą liczbę przypadków, w których zadanie mogłoby nie zadziałać. Jeśli nie uda mu się znaleźć błędów i jest pierwszym sprawdzającym, pod zadaniem umieszcza komentarz. Jeśli jest drugim sprawdzającym umieszcza komentarz oraz łączy kod z kodem głównym. W tym właśnie momencie zadanie oznaczane jest jako wykonane i jego kod staje się częścią programu głównego. W przypadku znalezienia błędu przez któregoś z programistów sprawdzających zadanie wraca do programisty w celu implementacji ubytków.

Poniższy rysunek (Rys. 7.2.) przedstawia wygląd przykładowego *Pull Request* w trakcie testowania zadania. Łatwo zauważyć, że na górze występuje nazwa zadania. Poniżej możemy znaleźć status danego *Pull Request* (gotowy do dołączenia, dołączony do kodu głównego), ostatnią wykonaną akcję oraz nazwę *brancha*, na którym znajduje się gotowe aktualne zadanie. Następnie możemy zobaczyć trzy zakładki: Konwersacja, *commit*, i zmienione pliki. W pierwszej widzimy opis zadania dodany przez implementującego zadanie, następnie komentarze innych programistów dotyczące testów wykonanych na danych zadaniu i ich błędów. Druga zakładka dotyczy liczby zatwierdzeń części kodu wykonanych przez programistę wraz z ich opisem, datą wstawienia i numerem. Ostatnia zakładka zawiera zmienione pliki, ich największą zaletę można przeczytać wyżej.

Rys. 7.2. Wygląd *Pull Request*

W ten sposób bardzo szybko wychwycona została bardzo duża liczba błędów. Jednak z  powodu znacznej ich liczby przytoczone zostanie tylko kilka, większych, ciekawszych i bardziej interesujących z perspektywy testowania programu.

Pierwszym przykładem błędu, który został naprawiony zanim zadanie zostało dołączone do kodu głównego był błąd związany z obsługą sytuacji wyjątkowych w programie. Jednemu programiście było bardzo trudno znaleźć wszystkie sytuacje wyjątkowe, nadać im odpowiednie nazwy oraz utworzyć informację w postaci alertu w interfejsie użytkownika. Zatem w czasie weryfikacji pozostali programiści pomogli znaleźć wszystkie miejsca w kodzie, w których mogą wystąpić sytuacje wyjątkowe. Zostały wykryte oraz naprawione, to w zadaniu z obsługą błędów, okazało się, że do obsługi jest więcej błędów niż programista zakładał, więc pozostali znaleźli miejsca, gdzie błędy także powinny zostać obsłużone.

Innym błędem znalezionym przez osoby testujące jest pojawianie się dodatkowych spacji w dokumencie PDF przy zmianie stylu części wyrazu (pogrubienie, podkreślenie, kursywa lub czcionka o stałej szerokości znaków). Był to błąd niezauważony przez programistę a jednak jest to błąd, który potencjalny użytkownik mógłby popełnić.

Dość istotnym błędem znalezionym w czasie testów bieżących na systemie z rodziny Linux był problem z możliwością zmiany kolejności zadań. Zostało to zaimplementowane na wzór przesuwania zakładek realizowanego w przeglądarkach internetowych. Niestety, mechanizm, który dobrze działał na systemach z rodziny Windows, nie działał poprawnie na systemach Linux. Ze względu na konieczność zmiany tej funkcjonalności, postanowione zostało pozostawienie opcji w systemie Windows i dodatkowo dodanie możliwości zmiany kolejności zadania z menu kontekstowego dla obu rodzin systemów.

## 7.4. Testy akceptacyjne (testowanie beta)

Warunkiem koniecznym do oficjalnej akceptacji aplikacji przez końcowego użytkownika było pozytywne przejście testów akceptacyjnych. Zostały one przeprowadzone po zakończeniu implementacji programu. Polegały na tworzeniu przykładowych zadań i egzaminów w formie gotowej do przekazania studentom na egzaminie. Aplikacja sprawdzana była pod kątem intuicyjności interfejsu użytkownika oraz funkcjonalności opisanych w wymaganiach funkcjonalnych. Służyły również weryfikacji czy wszystkie wymagania zostały zrozumiane poprawnie i ewentualnym ich doprecyzowaniu w przypadku ich braku. Wynik został dostarczony do programistów w formie opisu zmian, które powinny zostać wprowadzone, aby program mógł zostać zaakceptowany. Testy przeprowadzane były metodą manualną.

Jednym z wymagań dodatkowych było wprowadzenie możliwości dodanie numerów linii do kodu zadania dla każdego typu, nie tylko dla typu opierającego się na numerach linii. Mogłoby to zwiększyć czytelność w przypadku, gdy wykładowca chciałby w poleceniu zwrócić uwagę studentów na kawałek kodu a typem zadania byłby niekoniecznie typ numerowania linii.

Następnym wymaganiem dodatkowym było wprowadzenie możliwości wyróżnienia inną czcionką oraz kolorem poprawnych odpowiedzi w arkuszu dla nauczyciela, aby wyróżniały się one bardziej na tle całego egzaminu. Miało to na celu szybsze znalezienie przez prowadzących odpowiedniej odpowiedzi do danego zadania. Kolejnym wymaganiem dodatkowym dotyczącym dokumentu PDF było wprowadzenie możliwości dodania separatorów w postaci poziomych linii po każdym zadaniu.

Użytkownik poprosił również o wprowadzenie pomocy, z racji faktu, że z programu będzie korzystał jedynie kilka razy w roku, w momencie tworzenia egzaminu i egzaminu poprawkowego. A szukanie dokumentacji i konieczność odnalezienia w niej znaczenia odpowiedniej opcji powoduje, że znacznie wydłuża się czas pracy nad tworzeniem testu.

## 7.5. Podsumowanie

Testowanie jest ważną częścią procesu wytwarzania oprogramowania. W przypadku Generatora testów egzaminacyjnych z języka C++, źle przetestowana aplikacja niosłaby ze sobą szkody w postaci nawet konieczności powtórzenia egzaminu z Podstaw Programowania. Jednak patrząc na niewielką liczbę wymagań dodatkowych, jak i brak błędów związanych z wymaganiami funkcjonalnymi, można wnioskować, że wybrany sposób testowania aplikacji w przypadku Generatora testów egzaminacyjnych z języka C++ sprawdził się.

W Generatorze testów egzaminacyjnych nie zostało zastosowane testowanie jednostkowe. Spowodowane było to niewystarczającą ilością czasu na jego implementację jak i brakiem wystarczającej wiedzy na temat testowania. Ich brak zaczął doskwierać w przypadku, gdy działające wcześniej funkcjonalności, po drobnych zmianach w innych, zaczęły zwracać błędne wartości. Rozpoczynając pracę od początku najprawdopodobniej zapadłaby decyzja wprowadzeniu tego rodzaju testowania.

# 8. PODSUMOWANIE I WNIOSKI

## 8.1. Zrealizowana funkcjonalność

Podczas naszej pracy nad projektem udało nam się zrealizować niemalże wszystkie ustalone wcześniej wymagania, również te, które zostały sformułowane na późnym etapie implementacji. Poniżej znajduje się szczegółowa lista zrealizowanych funkcjonalności:

* zarządzanie arkuszem egzaminacyjnym:
  + odczyt/zapis arkusza z/do pliku,
  + zmiana rodzaju arkusza – wersja dla nauczyciela i studenta,
  + zmiana czcionki i jej rozmiaru osobno dla kodu, poleceń i odpowiedzi,
  + zmiana limitu czasu wykonania dla programów w arkuszu,
  + możliwość dodania poziomych linii (separatorów) między zadaniami – efekt widoczny po generacji arkusza,
  + usuwanie arkusza,
* zarządzanie zadaniami:
  + dodawanie, edycja, usuwanie zadania,
  + wybór typu zadania – zadania zostały z góry zdefiniowane w programie, różnią się od siebie m.in. sposobem wykonywania, markerami oraz poleceniami,
  + edycja domyślnych poleceń dla każdego typu zadania,
  + zmiana szerokości zadania w arkuszu egzaminacyjnym – istnieje możliwość dopasowania szerokości polecenia i kodu dla danego zadania,
  + ładowanie kodu do zadania z pliku,
  + zmiana nazw i kolejności zakładek z zadaniami,
  + formatowanie polecenia – istnieje możliwość zmiany odmiany czcionki dla wybranych fragmentów polecenia (pogrubienie, kursywa, podkreślenie, pismo maszynowe),
  + generowanie i edycja odpowiedzi do zadania – program automatycznie generuje odpowiedź do zadania po wykonaniu wprowadzonego kodu. Możliwa jest późniejsza edycja tej odpowiedzi jak i również jej zapisanie tak, by kolejne wykonanie programu nie skutkowało jej nadpisaniem,
  + oznaczanie kodu markerami:
    - „Ukryj” – ukrywa kod na arkuszu, uwzględniając go przy każdej kompilacji,
    - „Usuń” – usuwa kod z arkusza, uwzględniając go tylko przy kompilacji testowej,
    - „Luka” – tworzy lukę z zaznaczonego tekstu, zastępując go polem na odpowiedź w wersji egzaminu dla studenta,
  + dodawanie paska z numerami linii dla kodu dołączonego do zadania,
* generowanie arkusza w formacie .pdf:
  + w wersji dla nauczyciela: z uzupełnionymi polami na odpowiedź,
  + w wersji dla studenta: z pustym miejscem na odpowiedź,
  + zmiana daty testu,
  + zmiana typu testu:
    - wersja dla nauczyciela,
    - wersja dla studenta,
* kompilacja i wykonywanie wprowadzonego kodu:
  + kompilacja z uwzględnieniem stanu markerów,
  + kompilacja bez uwzględniania stanu markerów,
  + automatyczna kompilacja i generowanie odpowiedzi do zadań podczas generowania pliku .pdf,
  + wykrywanie błędów kompilacji i wykonania.

## 8.2. Niezrealizowana funkcjonalność

### 8.2.1 Arkusz interaktywny

W trakcie ustalania wymagań do projektu pojawił się pomysł generacji arkusza dla studenta dodatkowo w wersji interaktywnej, która umożliwiłaby mu wypełnianie egzaminu w domu (w ramach przygotowania) i automatycznie sprawdzała poprawność wprowadzonych odpowiedzi. Wymaganie to było na tyle interesujące, że postanowiliśmy uzależnić od niego wybór biblioteki do generowania plików .pdf. Niestety, funkcjonalność ta nie została przez nas zrealizowana z uwagi na jej złożoność i jednoczesny brak dostatecznej ilości czasu na jej realizację.

### 8.2.2 Zapis arkusza w formacie edytowalnym

Jednym z istotnych wymagań, które nie zostało sformułowane w początkowym etapie projektu było umożliwienie zapisu arkusza w formacie nadającym się do edycji. Oferowany przez nas sposób umożliwia jedynie przechowanie informacji o treści poszczególnych zadań (poleceniu, kodzie, odpowiedzi itp.), natomiast nie umożliwia późniejszej wizualnej edycji tak przygotowanego egzaminu. Jest to problematyczne z uwagi na fakt, że użytkownik mógłby potrzebować zmienić układ pewnych elementów egzaminu i w obecnej formie nie jest w stanie tego zrobić. Użyta przy generacji pliku .pdf biblioteka nie umożliwia jednak zapisu danych w innym formacie, a co za tym idzie konwersja taka musiałaby być przeprowadzana bezpośrednio z formatu .pdf. Gdyby arkusz składał się jedynie z tekstu, jego treść byłaby możliwa do przeniesienia bezpośrednio do np. formatu docx. Generowany przez nas plik posiada jednak elementy graficzne, które nie mogą być w prosty sposób przekonwertowane na postać tekstową, a co za tym idzie, funkcjonalność ta nie jest prawdopodobnie możliwa do zrealizowania przy wykorzystywanej bibliotece.

## 8.3. Napotkane trudności

### 8.3.1. Zmiana kolejności zadań

Zmiana kolejności zadań to wymaganie, które pojawiło się w późniejszym etapie projektu. Zakładało ono możliwość układania zakładek z zadaniami w kolejności, w której miały pojawić się na arkuszu. Zadanie to było problematyczne z dwóch powodów. Po pierwsze, JavaFX nie posiada wbudowanej możliwości przesuwania zakładek, funkcjonalność ta została więc przez nas zaimplementowana. Własne rozwiązanie sprawiło jednak, że uwidocznił się drugi problem, mianowicie JavaFX ma trudności z obsługą animacji w systemie Linux. Tego problemu nie udało nam się ominąć i w rezultacie przesuwanie zakładek dostępne jest tylko w systemie Windows, natomiast uniwersalnie istnieje także możliwość zmiany położenia zadania przy pomocy menu kontekstowego.

### 8.3.2. Ubogość biblioteki pdfBox

Wybrana z uwagi na możliwość generacji arkusza interaktywnego biblioteka pdfBox okazała się być bardzo okrojona, co sprawiło, że znaczna część funkcjonalności musiała być przez nas pisana własnoręcznie. Metody, które udostępnia z pewnością mają zastosowanie w przypadku prostych dokumentów tekstowych, jednak z uwagi na występowanie w generowanych przez nas arkuszach np. kolumn z tekstem, jej API nie było dla nas wystarczające. W rezultacie musieliśmy zająć się generacją arkusza znak po znaku, co przekłada się negatywnie na czas  jego generacji.

### 8.3.3. Narzędzie do projektowania interfejsu

Do projektowania interfejsu użyliśmy narzędzia SceneBuilder (w środowisku Intellij Idea), które spełniało swoje zadanie do czasu pojawienia się komponentów z biblioteki RichTextFX dostarczających m.in. gotowe pola do edycji kodu. Mimo prawidłowego dołączenia biblioteki, środowisko przy każdym uruchomieniu narzędzia SceneBuilder usuwało importy związane z ww. biblioteką, powodując tym samym błędy kompilacji. Było to o tyle uciążliwe, że ścieżka do importu nie była podpowiadana przez środowisko, przez co niejednokrotnie byliśmy zmuszeni wpisywać ją ręcznie, co spowalniało pracę nad implementacją.

### 8.3.4. Powolne tworzenie się procesów w systemie Windows

Kompilacja całego egzaminu wymaga wywołania kompilatora osobno dla każdego zadania i podania mu na wejście kodu zawartego w zadaniu. O ile w systemie Linux operacja ta jest wykonywana błyskawicznie, o tyle w systemie Windows trwa ona dłuższą chwilę, przez co kompilacja całego egzaminu znacznie się wydłuża. Rozwiązaniem byłoby uruchamianie kompilatora wielokrotnie w jednym procesie, jednak mimo wielu prób, nie udało nam się osiągnąć żądanego efektu.

### 8.3.5. Brak testów jednostkowych i integracyjnych

Tworząc kod do naszego projektu nie posiadaliśmy dostatecznej wiedzy na temat testów i korzyści płynących z ich wykorzystywania. W rezultacie jedynym sposobem testowania było ręczne weryfikowanie poprawności poszczególnych funkcjonalności. O ile w początkowej fazie projektu było to w zupełności wystarczającą metodą, o tyle ostatnie poprawki często niosły ze sobą problemy w innych częściach projektu, nierzadko będące trudne do zauważenia. Pokrycie chociaż części funkcjonalności testami było jednym z wyznaczonych przez nas zadań pobocznych, niestety ilość pracy, jaką musielibyśmy włożyć w końcowej fazie projektu w dodanie testów, nie umożliwiłaby nam zrealizować w pełni pozostałych zadań, dlatego też postanowiliśmy zrezygnować z tego pomysłu.

## 8.4. Zadania na przyszłość

### 8.4.1. Umożliwienie zapisu arkusza w formacie edytowalnym

Zapis arkusza w formacie innym niż .pdf znacznie ułatwiłby późniejszą pracę z nim. Aby to osiągnąć należałoby wykorzystać bibliotekę wykorzystującą do generacji pliku szablony np. JasperReports. Takie podejście umożliwiałoby zapis arkusza najpierw w formie pliku HTML, a następnie szybkie wygenerowanie na jego podstawie pliku .pdf lub np. docx. Zmiana biblioteki wiązałaby się jednak z całkowitą zmianą dotychczas używanej metody generacji arkusza w formacie .pdf.

### 8.4.2. Umożliwienie generacji arkusza interaktywnego

Arkusz interaktywny to rozwiązanie, które umożliwiłoby użytkownikowi generowanie arkuszy dla studentów w celu przygotowania do rzeczywistego egzaminu. Pozwalałoby na sprawdzanie arkusza już na etapie jego wypełniania. To z kolei mogłoby znaleźć dodatkowe zastosowanie w formie wejściówek przeprowadzanych na komputerach laboratoryjnych. Odpowiedni skrypt mógłby sprawdzać pola uzupełnione przez studenta, a wynik zapisywać w przeznaczonym do tego celu systemie. Arkusz interaktywny prawdopodobnie mógłby powstać przy wykorzystaniu tej samej biblioteki, co do generowania zwykłych arkuszy, gdyż umożliwia ona zapis w formacie HTML, a więc i bezproblemowe osadzenie w nim skryptów.

### 8.4.3. Obsługa błędów wykonania dla dostarczonego kodu

Aktualnie nasz program jedynie wykrywa błędy wykonania i wyświetla komunikat o jego wystąpieniu. Aby w całości zastąpić funkcjonalność popularnych edytorów kodów takich jak np. Visual Studio, należałoby dodać możliwość sygnalizacji błędu komunikatem opisującym rodzaj zaistniałego błędu.

### 8.4.4. Pokrycie kluczowych funkcjonalności testami

Z uwagi na rozbudowanie naszego programu, przed jego dalszym rozwijaniem należałoby pokryć chociaż kluczowe funkcjonalności testami jednostkowymi i integracyjnymi. Pozwoliłoby to na znacznie szybsze wykrywanie błędów podczas implementacji i zabezpieczyłoby program przed przypadkową utratą części funkcjonalności w czasie jego rozwijania. Ponadto, zastosowanie testów umożliwiłoby lepsze spełnianie postawionych wymagań, gdyż miałyby one w nich bezpośrednie odwzorowanie.

### 8.4.5. Proste podpowiedzi i kolorowanie składni

Częściowo zastępując naszą aplikacją programy aktualnie używane do edycji kodu, powinniśmy zadbać o zapewnienie zbliżonego środowiska pracy. Niestety nasza aplikacja nie udostępnia podpowiedzi do wprowadzanego kodu oraz nie koloruje składni, co może spowalniać pracę mniej doświadczonych użytkowników. Wprowadzenie tej funkcjonalności mogłoby także zredukować liczbę błędów wynikających z popełnienia kompilujących się literówek.

## 8.5. Podsumowanie

Żyjemy w czasach nieustannego rozwoju oprogramowania, nowe aplikacje realizujące za użytkownika coraz więcej czynności sprawiają, że popełnia on w swojej pracy coraz mniej błędów. To niewątpliwie krok w dobrą stronę, biorąc pod uwagę jakie konsekwencje mogą one nieść za sobą. Pomyłka w arkuszu egzaminacyjnym niejednokrotnie unieważnia błędne zadanie sprawiając, że test nie jest kompletny, a tym samym nie sprawdza wszystkich umiejętności studenta. Tworząc naszą aplikację kierowaliśmy się przede wszystkim właśnie koniecznością wyeliminowania potencjalnych błędów w arkuszu. Chociaż nie możemy zapobiec literówkom w poleceniu, nasza aplikacja pozwala uchronić się przed pomyłką w kodzie, która mogłaby doprowadzić do całkowitej zmiany odpowiedzi do zadania.

Czy nasza aplikacja mogłaby działać lepiej? Z pewnością tak, o czym świadczą chociażby wyznaczone zadania na przyszłość. To nie oznacza jednak, że jest ona niekompletna. Zrealizowana przez nas w projekcie funkcjonalność ułatwia pracę nad tworzeniem zadań i pozwala uniknąć wielu błędów, wciąż jednak pozostawia na nie miejsce. Realizacja zadań na przyszłość mogłaby przyczynić się do powstania aplikacji, która niemalże w całości wyeliminowałaby możliwość zaistnienia błędu w tego typu egzaminie.

Projekt realizowaliśmy w czteroosobowym zespole, w którym każdy miał możliwość zajmować się zarówno interfejsem użytkownika, jego obsługą, tworzeniem arkusza, a także implementacją logiki aplikacji. Podzieliliśmy jednak zadania tak, by za każde z nich była odpowiedzialna tylko jedna osoba. Interfejsem graficznym użytkownika zajmowała się Natalia Kowalik, jego obsługą – Paweł Lal, generowaniem pliku PDF – Martyna Łuczkowska, natomiast obsługą logiki aplikacji – Marta Mazur.

Podczas przygotowywania projektu mieliśmy okazję poznać bliżej zarówno metodyki jak i narzędzia wspomagające pracę zespołową. Pozwoliło nam to również docenić doświadczenie nabyte na studiach oraz praktykach i wykorzystać je w naszym własnym projekcie.

# WYKAZ LITERATURY

1. Materiały dydaktyczne z przedmiotu "Podstawy Programowania" (kier. Informatyka)
2. Dokumentacja Java SE 8, <https://docs.oracle.com/javase/8/docs/>, (data dostępu 07.12.2016 r.)
3. Richard Warburton: *Java 8 Lambdas*, O’Reilly 2014, Sebastopol, CA, USA
4. Dokumentacja JavaFX, <http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>, (data dostępu 07.12.2016 r.)
5. Dokumentacja PDFBox, <https://pdfbox.apache.org/docs/2.0.0/javadocs/>, (data dostępu 07.12.2016 r.)
6. Dokumentacja JAXB, <http://www.oracle.com/technetwork/articles/javase/index-140168.html>, (data dostępu 07.12.2016 r.)
7. Dokumentacja Apache Maven, <https://maven.apache.org/guides/>, (data dostępu 07.12.2016 r.)
8. Dokumentacja i źródło narzędzia do tworzenia zaawansowanych pól teksowych RichTextFX <https://github.com/TomasMikula/RichTextFX/blob/master/README.md>, (data dostępu 07.12.2016 r.)
9. Jenifer Tidwell: *Designing Interfaces*, O’Reilly 2010, Sebastopol, CA, USA

# WYKAZ RYSUNKÓW

Rys. 2.1. Wygląd aplikacji testowej stworzonej przez IndiaBIX 12

Rys. 2.2. Wygląd aplikacji CppDroid 13

Rys. 3.1. Skrócony diagram przypadków użycia 18

Rys. 4.1. Diagram komponentów systemu 25

Rys. 4.2. Diagram klas 29

Rys. 5.1. Zrzut ekranu przedstawiający widok narzędzia JavaFX Scene Builder 36

Rys. 5.2. Zrzut ekranu przedstawiający zdalne repozytorium projektu inżynierskiego

na platformie GitHub 39

Rys. 5.3. Zrzut ekranu przedstawiający dodatek Git Extensions 39

Rys. 5.4. Zrzut ekranu przedstawiający narzędzie YouTrack 41

Rys. 5.5. Zrzut ekranu przedstawiający zarządzanie zadaniami na platformie GitHub 41

Rys. 5.6. Diagram klas dla paczki compiler 42

Rys. 5.7. Diagram klas dla paczki xml 43

Rys. 6.1. Okno dodawania zadania 45

Rys. 6.2. Edycja zadania typu „Wyjście programu” 46

Rys. 6.3. Zadanie typu „Wyjście programu” w arkusz 46

Rys. 6.4. Edycja zadania typu „Wyjście funkcji” 47

Rys. 6.5. Zadanie typu „Wyjście funkcji” w arkuszu 47

Rys. 6.6. Edycja zadania typu "Wartość zmiennej" 48

Rys. 6.7. Zadanie typu "Wartość zmiennej" w arkuszu 49

Rys. 6.8. Edycja zadania typu "Zwrócona wartość" 49

Rys. 6.9. Zadanie typu "Zwrócona wartość" w arkuszu 50

Rys. 6.10. Edycja zadania typu "Numery linii" 50

Rys. 6.11. Zadanie typu "Numery linii" w arkuszu 51

Rys. 6.12. Edycja zadania typu „Uzupełnianie luk” 52

Rys. 6.13. Zadanie typu „Uzupełnianie luk” w arkuszu 52

Rys. 6.14. Edycja kodu z użyciem markerów 53

Rys. 6.15. Wynik edycji kodu z użyciem markerów 54

Rys. 6.16. Okno ustawień zapisu arkusza PDF 55

Rys. 6.17. Okno kompilacji egzaminu 56

Rys. 7.1. Wygląd tablicy zadań na GitHub 62

Rys. 7.2. Wygląd Pull Request 63

# WYKAZ TABEL

Tabela 3.1. Zarządzanie zadaniami 19

Tabela 3.2. Dodawanie zadania 19

Tabela 3.3. Usuwanie zadania 20

Tabela 3.4. Edycja zadania 20

Tabela 3.5. Edycja polecenia domyślnego 21

Tabela 3.6. Wykonywanie kodu dołączonego do zadania 21

Tabela 3.7. Zmiana kolejności zadania 22

Tabela 3.8. Zarządzanie arkuszem do tworzenia egzaminu 22

Tabela 3.9. Odczyt arkusza 23

Tabela 3.10. Zapis arkusza 23

Tabela 3.11. Generacja arkusza egzaminacyjnego 24

Tabela 4.1. Charakterystyka warstwy prezentacji 26

Tabela 4.2. Charakterystyka warstwy usług 27

Tabela 4.3. Charakterystyka warstwy danych 27

Tabela 4.4. Charakterystyka komponentu GenerateExamController 27

Tabela 4.5. Charakterystyka komponentu MainController 27

Tabela 4.6. Charakterystyka komponentu ManageTaskController 27

Tabela 4.7. Charakterystyka komponentu Exam 27

Tabela 4.8. Charakterystyka komponentu Task 27

Tabela 4.9. Charakterystyka komponentu GCC 28

Tabela 4.10. Charakterystyka komponentu XML Manager 28

Tabela 4.11. Charakterystyka komponentu PDF Manager 28

Tabela 4.12. Charakterystyka komponentu XML 28

Tabela 4.13. Charakterystyka klasy MainController 29

Tabela 4.14. Charakterystyka klasy ManageTaskController 30

Tabela 4.15. Charakterystyka klasy GenerateExamController 30

Tabela 4.16. Charakterystyka klasy GCC 30

Tabela 4.17. Charakterystyka klasy Task 31

Tabela 4.18. Charakterystyka klasy TaskType 31

Tabela 4.19. Charakterystyka klasy Exam 31

Tabela 4.20. Charakterystyka klasy PDF 32

Tabela 4.21. Schemat układu okna głównego 32

Tabela 4.22. Charakterystyka panelu górnego 32

Tabela 4.23. Charakterystyka panelu lewego górnego 32

Tabela 4.24. Charakterystyka panelu lewego środkowego 33

Tabela 4.25. Charakterystyka panelu lewego dolnego 33

Tabela 4.26. Charakterystyka panelu prawego 33

Tabela 4.27. Charakterystyka formularza zapisu PDF 33

Tabela 4.28. Charakterystyka formularza zarządzania zadaniem 33

Tabela 4.29. Charakterystyka formularza zmiany wyglądu dokumentu 33

Tabela 4.30. Charakterystyka formularza zmiany limitu czasu wykonania 34

Tabela 4.31. Charakterystyka formularza zmiany szerokości elementów zadania 34

Tabela 4.32. Charakterystyka pliku zadania 34

Tabela 4.33. Charakterystyka arkusza 35

Tabela 4.34. Charakterystyka pliku polecenia 35

# DODATEK A: Diagram przypadków użycia

Na rys. 1.1. znajduje się pełny diagram przypadków użycia.

C:\Users\Pawel\Downloads\Diagram przypadków użycia (1).png

Rys. 1.1. Diagram przypadków użycia

# DODATEK B: Opis przypadków użycia

W poniższych tabelach 1.1. – 1.24. znajdują się opisy przypadków użycia przedstawionych na rysunku 1.1.

**Tabela 1.1.** Zarządzanie zadaniami

|  |  |
| --- | --- |
| Nazwa przypadku: | Zarządzanie zadaniami |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie z menu opcji „Zadanie” |
| Warunki początkowe: | Brak |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie zawartości menu „Zadanie”. |

**Tabela 1.2.** Dodawanie zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Dodawanie zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Dodaj” lub „Wczytaj zadanie” |
| Warunki początkowe: | Otwarte menu „Zadanie” |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie okna dodawania zadania lub wczytania pliku |

**Tabela 1.3.** Tworzenie nowego zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Tworzenie nowego zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Dodaj” |
| Warunki początkowe: | Otwarte menu „Zadanie” |
| Opis przebiegu interakcji: | 1. Program otwiera okno kreatora zadania i blokuje przycisk dodawania zadania.  2. Użytkownik wybiera typ zadania.  3. Program odblokowuje przycisk dodawania zadania.  4. Użytkownik wybiera plik z kodem do zadania.  5. Użytkownik wybiera opcję „Zapisz”. |
| Przebiegi alternatywne: | 2a.1 Użytkownik pomija krok 2 i przechodzi do pkt. 5.  2a.3 Program nie pozwala na dodanie nowego zadania.  2a.4 Użytkownik wraca do pkt. 2.  4a Użytkownik pomija krok 4. |
| Sytuacje wyjątkowe: | 4a.1 Użytkownik wpisuje niepoprawną nazwę pliku (nieistniejący plik).  4a.2 Program komunikuje brak pliku o podanej nazwie.  4b.1 Użytkownik wpisuje niepoprawną nazwę pliku (zły format pliku).  4b.2 Program ignoruje wprowadzony plik. |
| Warunki końcowe: | Wyświetlenie zawartości menu „Zadanie”. |

**Tabela 1.4.** Wybór typu zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Wybór typu zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Typ zadania” |
| Warunki początkowe: | Otwarte okno dodawania zadania |
| Opis przebiegu interakcji: | 1. Program wyświetla możliwe do wyboru zadania.  2. Użytkownik wybiera zadanie z listy. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Załadowanie domyślnej zawartości wybranego typu zadania do pola na polecenie. |

**Tabela 1.5.** Wczytanie kodu z pliku

|  |  |
| --- | --- |
| Nazwa przypadku: | Wczytanie kodu z pliku |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Załaduj kod” |
| Warunki początkowe: | Otwarte okno dodawania zadania |
| Opis przebiegu interakcji: | 1. Program wyświetla okno wyboru pliku.  2. Użytkownik wybiera plik. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 2a.1 Użytkownik wpisuje niepoprawną nazwę pliku (nieistniejący plik).  2a.2 Program komunikuje brak pliku o podanej nazwie.  2b.1 Użytkownik wpisuje niepoprawną nazwę pliku (zły format pliku).  2b.2 Program ignoruje wprowadzony plik. |
| Warunki końcowe: | Załadowanie kodu z pliku do pola na kod. |

**Tabela 1.6.** Ładowanie zadania z pliku

|  |  |
| --- | --- |
| Nazwa przypadku: | Ładowanie zadania z pliku |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Wczytaj zadanie” |
| Warunki początkowe: | Brak |
| Opis przebiegu interakcji: | 1. Program wyświetla okno wyboru pliku.  2. Użytkownik wybiera zadanie z listy. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 2a.1 Użytkownik wpisuje niepoprawną nazwę pliku (nieistniejący plik).  2a.2 Program komunikuje brak pliku o podanej nazwie.  2b.1 Użytkownik wpisuje niepoprawną nazwę pliku (zły format pliku).  2b.2 Program ignoruje wprowadzony plik. |
| Warunki końcowe: | Utworzenie zadania na podstawie informacji zawartych w wybranym pliku. |

**Tabela 1.7.** Usuwanie zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Usuwanie zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Usuń” |
| Warunki początkowe: | Otwarte menu „Zadanie”, niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Usunięcie aktualnego zadania z arkusza. |

**Tabela 1.8.** Edycja zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Edycja zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Edytuj” |
| Warunki początkowe: | Otwarte menu „Zadanie”, niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | 1. Program otwiera okno kreatora zadania, uzupełnia je danymi o aktualnym zadaniu i odblokowuje przycisk dodania zadania.  2. Użytkownik wybiera typ zadania.  3. Użytkownik wybiera plik z kodem do zadania.  4. Użytkownik wybiera opcję „Zapisz”. |
| Przebiegi alternatywne: | 2a Użytkownik pomija krok 2.  3a Użytkownik pomija krok 3. |
| Sytuacje wyjątkowe: | 3a.1 Użytkownik wpisuje niepoprawną nazwę pliku (nieistniejący plik).  3a.2 Program komunikuje brak pliku o podanej nazwie.  3b.1 Użytkownik wpisuje niepoprawną nazwę pliku (zły format pliku).  3b.2 Program ignoruje wprowadzony plik. |
| Warunki końcowe: | Zastąpienie treści aktualnego zadania nowo wprowadzonymi danymi. |

**Tabela 1.9.** Usuwanie kodu do kompilacji

|  |  |
| --- | --- |
| Nazwa przypadku: | Usuwanie kodu do kompilacji |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Usuń” poniżej pola z kodem |
| Warunki początkowe: | Niepusty arkusz z zadaniami, zaznaczony kawałek kodu |
| Opis przebiegu interakcji: | 1. Program usuwa zaznaczony fragment kodu z pliku wejściowego dla kompilatora. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Zmiana koloru zaznaczonego kodu na pomarańczowy. |

**Tabela 1.10.** Ukrywanie kodu do kompilacji

|  |  |
| --- | --- |
| Nazwa przypadku: | Ukrywanie kodu do kompilacji |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Ukryj” poniżej pola z kodem |
| Warunki początkowe: | Niepusty arkusz z zadaniami, zaznaczony kawałek kodu |
| Opis przebiegu interakcji: | 1. Program ukrywa zaznaczony fragment kodu na arkuszu w formacie .pdf. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Zmiana koloru zaznaczonego kodu na niebieski. |

**Tabela 1.11.** Edycja polecenia domyślnego

|  |  |
| --- | --- |
| Nazwa przypadku: | Edycja polecenia domyślnego |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie zakładki „Polecenia domyślne” |
| Warunki początkowe: | Otwarte okno „Ustawienia” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno edycji poleceń domyślnych.  2. Użytkownik wybiera typ zadania.  3. Program wyświetla polecenie domyślne dla wybranego typu zadania w polu do edycji.  4. Użytkownik edytuje treść polecenia.  5. Użytkownik wybiera opcję „Zastosuj”. |
| Przebiegi alternatywne: | 2a.1 Użytkownik pomija krok 2 i przechodzi do pkt. 5.  2a.2 Program nie podejmuje żadnego działania.  2a.3 Użytkownik powraca do pkt. 2.  4a Użytkownik pomija krok 4. |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Zastąpienie treści wybranego polecenia domyślnego zawartością pola do edycji. |

**Tabela 1.12.** Wykonywanie kodu dołączonego do zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Wykonywanie kodu dołączonego do zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Wykonaj” lub „Wykonaj testowo” |
| Warunki początkowe: | Niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | 1. Program kompiluje kod dołączony do zadania.  2. Wynik kompilacji zostaje wyświetlony na ekranie.  3. Program wykonuje program powstały po kompilacji.  4. Wynik programu zostaje wyświetlony na ekranie. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1 Dołączony kod nie jest kompilowalny.  1a.2 Wynik nieudanej kompilacji zostaje wyświetlony na ekranie.  1b.1 Zadanie nie posiada kodu.  1b.2 Program wyświetla informację o pustym pliku wejściowym dla kompilatora.  3a.1 Dołączony kod posiada błąd wykonania.  3a.2 Program wyświetla komunikat o błędzie wykonania. |
| Warunki końcowe: | Wyświetlenie wyniku kompilacji i wykonania w oknie kompilacji. |

**Tabela 1.13.** Kompilacja kodu dołączonego do zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Kompilacja kodu dołączonego do zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Wykonaj” lub „Wykonaj testowo” |
| Warunki początkowe: | Niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | 1. Program kompiluje kod dostarczony do zadania. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1. Dostarczony kod nie jest kompilowalny.  1a.2. Program nie tworzy pliku wykonywalnego. |
| Warunki końcowe: | Utworzenie pliku wykonywalnego. |

**Tabela 1.14.** Wyświetlanie wyniku kompilacji

|  |  |
| --- | --- |
| Nazwa przypadku: | Wyświetlanie wyniku kompilacji |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Pomyślna kompilacja kodu dołączonego do zadania |
| Warunki początkowe: | Niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie wyniku kompilacji w oknie kompilacji. |

**Tabela 1.15.** Pobieranie wyjścia programu

|  |  |
| --- | --- |
| Nazwa przypadku: | Pobieranie wyjścia programu |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wyświetlenie pomyślnego wyniku kompilacji kodu dołączonego do zadania |
| Warunki początkowe: | Niepusty arkusz z zadaniami, kompilowalny kod wykonywanego zadania |
| Opis przebiegu interakcji: | 1. Program wykonuje plik powstały w wyniku pomyslnej kompilacji. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1. Wykonywany kod powoduje błąd wykonania.  1a.2. Program wyświetla komunikat o błędzie wykonania. |
| Warunki końcowe: | Wyświetlenie wyjścia programu w oknie kompilacji. |

**Tabela 1.16.** Wykrywanie błędów wykonania

|  |  |
| --- | --- |
| Nazwa przypadku: | Wykrywanie błędów wykonania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wykonanie kodu zawierającego błąd wykonania |
| Warunki początkowe: | Niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie okna z informacją o błędzie wykonania. |

**Tabela 1.17.** Zmiana kolejności zadania

|  |  |
| --- | --- |
| Nazwa przypadku: | Zmiana kolejności zadania |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie z menu kontekstowego zadania opcji „Przesuń w …” |
| Warunki początkowe: | Niepusty arkusz z zadaniami |
| Opis przebiegu interakcji: | 1. Program zamienia kolejnością aktualnie zadanie z zadaniem znajdującym się po wybranej stronie. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1 W arkuszu znajduje się tylko jedno zadanie.  1a.2 Pozycja zadania zostaje odświeżona.  1b.1 Została wybrana opcja „Przesuń w lewo” dla zadania znajdującego się skrajnie z lewej strony.  1b.2 Pozycja zadania zostaje odświeżona.  1c.1 Została wybrana opcja „Przesuń w prawo” dla zadania znajdującego się skrajnie z prawej strony.  1c.2 Pozycja zadania zostaje odświeżona. |
| Warunki końcowe: | Zmiana kolejności zadań w arkuszu. |

**Tabela 1.18.** Zarządzanie arkuszem do tworzenia egzaminu

|  |  |
| --- | --- |
| Nazwa przypadku: | Zarządzanie arkuszem do tworzenia egzaminu |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie z menu opcji „Arkusz” |
| Warunki początkowe: | Brak |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie zawartości menu „Arkusz”. |

**Tabela 1.19.** Odczyt arkusza

|  |  |
| --- | --- |
| Nazwa przypadku: | Odczyt arkusza |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Wczytaj arkusz” |
| Warunki początkowe: | Otwarte menu „Arkusz” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno wyboru pliku.  2. Użytkownik wybiera żądany plik.  3. Użytkownik potwierdza wybór przyciskiem „Otwórz”.  4. Program tworzy nowy arkusz i ładuje do niego dane z pliku. |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 2a.1 Użytkownik wybiera plik niebędący arkuszem  2a.2 Program komunikuje błąd odczytu danych z pliku i przerywa proces ładowania arkusza. |
| Warunki końcowe: | Załadowanie danych z pliku do nowego arkusza. |

**Tabela 1.20.** Zapis arkusza

|  |  |
| --- | --- |
| Nazwa przypadku: | Zapis arkusza |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Zapisz” |
| Warunki początkowe: | Otwarte menu „Arkusz” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno wyboru miejsca zapisu.  2. Użytkownik wybiera żądaną lokalizację (wraz z nazwą pliku).  3. Program zapisuje zawartość arkusza we wskazanym pliku. |
| Przebiegi alternatywne: | 1a.1 W danej instancji programu został już raz zapisany arkusz.  1a.2 Program zapisuje zawartość arkusza we wcześniej wskazanym pliku. |
| Sytuacje wyjątkowe: | 3a.1 Użytkownik nie ma praw do zapisu pliku w danej lokacji.  3a.2 Program wyświetla komunikat o niemożności zapisu pliku w podanej lokacji. |
| Warunki końcowe: | Zapisanie danych zawartych w arkuszu, we wskazanym pliku. |

**Tabela 1.21.** Generacja arkusza egzaminacyjnego

|  |  |
| --- | --- |
| Nazwa przypadku: | Generacja arkusza egzaminacyjnego |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Generuj PDF” |
| Warunki początkowe: | Otwarte menu „Arkusz” |
| Opis przebiegu interakcji: | 1. Program wyświetla okno generacji arkusza egzaminacyjnego |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | 1a.1 Nie jest otworzony żaden arkusz.  1a.2 Program wyświetla komunikat o niemożności wygenerowania pustego arkusza. |
| Warunki końcowe: | Wyświetlenie okna generacji arkusza egzaminacyjnego. |

**Tabela 1.22.** Podgląd arkusza egzaminacyjnego

|  |  |
| --- | --- |
| Nazwa przypadku: | Podgląd arkusza egzaminacyjnego |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Otwórz ostatni PDF” |
| Warunki początkowe: | Otwarte menu „Arkusz”, wcześniejsze wygenerowanie w danej instancji programu co najmniej jednego arkusza |
| Opis przebiegu interakcji: | Brak |
| Przebiegi alternatywne: | Brak |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Wyświetlenie pliku arkusza w dostępnej przeglądarce plików .pdf. |

**Tabela 1.23.** Generacja arkusza egzaminacyjnego w wersji dla studenta

|  |  |
| --- | --- |
| Nazwa przypadku: | Generacja arkusza egzaminacyjnego w wersji dla studenta |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Zapisz arkusz” |
| Warunki początkowe: | Otwarte okno zapisu arkusza w formacie .pdf |
| Opis przebiegu interakcji: | 1. Użytkownik wybiera żądaną datę testu.  2. Użytkownik wybiera z listy rozwijanej typ testu „student”.  3. Użytkownik wybiera opcję „Zapisz arkusz”. |
| Przebiegi alternatywne: | 1a.1. Użytkownik pomija wybór daty.  1a.2. Następuje przejście do kroku 2. |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Przejście do okna kompilacji egzaminu. |

**Tabela 1.24.** Generacja arkusza egzaminacyjnego w wersji dla nauczyciela

|  |  |
| --- | --- |
| Nazwa przypadku: | Generacja arkusza egzaminacyjnego w wersji dla nauczyciela |
| Aktorzy: | Użytkownik |
| Zdarzenie inicjujące: | Wybranie opcji „Zapisz arkusz” |
| Warunki początkowe: | Otwarte okno zapisu arkusza w formacie .pdf |
| Opis przebiegu interakcji: | 1. Użytkownik wybiera żądaną datę testu.  2. Użytkownik wybiera z listy rozwijanej typ testu „nauczyciel”.  3. Użytkownik wybiera opcję „Zapisz arkusz”. |
| Przebiegi alternatywne: | 1a.1. Użytkownik pomija wybór daty.  1a.2. Następuje przejście do kroku 2. |
| Sytuacje wyjątkowe: | Brak |
| Warunki końcowe: | Przejście do okna kompilacji egzaminu. |