WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: IOT- Internet Rzeczy

Jakub Czarnota
Nr albumu studenta w69772

Aplikacja konsolowa Elektroniczny system oceniania

Prowadzący: mgr inż. Ewa Żesławska

Praca projektowa programowanie obiektowe C#

Rzeszów 2025

Spis treści

Wstęp

Nowoczesne systemy oceniania znacznie ułatwiają życie zarówno nauczycielom jak i uczniom. Dzięki nim oceny mogą być szybko wprowadzane, przechowywane i przeglądane w formie elektronicznej, co eliminuje potrzebę korzystania z tradycyjnych, papierowych dzienników. Wystarczy kilka kliknięć, aby uzyskać dostęp do informacji o postępach w nauce, bez konieczności żmudnego przeglądania zapisów czy odwiedzania szkoły. Jest to także rozwiązanie ekologiczne pozwalające na redukcję wycinki drzew potrzebnych do stworzenia papierowych dzienników.

Nauczyciele zyskują także możliwość generowania różnorodnych raportów i statystyk, które pomagają lepiej analizować wyniki uczniów, na przykład według przedmiotów, okresów nauki czy wyników poszczególnych klas. To nie tylko oszczędność czasu, ale także większa przejrzystość i mniejsze ryzyko błędów. Również zwiększa to kontrolę dyrekcji nad kształceniem uczniów.

W ramach tego projektu moim celem będzie stworzenie prostego elektronicznego systemu oceniania. System ten pozwoli nauczycielom na wprowadzanie ocen cząstkowych, a uczniom da dostęp do przeglądania swoich wyników. Dodatkowo zostaną wprowadzone funkcje generowania raportów i statystyk, które pomogą lepiej zrozumieć osiągnięcia edukacyjne i wspierać proces nauczania.

Rozdział 1

Opis założeń projektu

1.1 Cele projektu

Celem projektu jest stworzenie prostego i intuicyjnego elektronicznego systemu oceniania w języku C#, który ułatwi zarządzanie ocenami z różnych przedmiotów. System będzie wspierał zarówno nauczycieli, którzy będą mogli wprowadzać oceny, jak i uczniów, którzy zyskają łatwy dostęp do swoich wyników.

Projekt ma także na celu lepsze zrozumienie, jak działają systemy do zarządzania informacjami w szkołach oraz jakie elementy są kluczowe, aby takie rozwiązania były wygodne w użyciu i niezawodne. Poza podstawowymi funkcjami planujemy dodać takie opcje, jak generowanie raportów i statystyk. Dzięki temu system pomoże w lepszej analizie wyników uczniów i wsparciu procesu nauczania.

Efektem końcowym będzie aplikacja konsolowa, która spełni wszystkie te założenia.

1.2 Wymagania funkcjonalne i niefunkcjonalne

Wymagania funkcjonalne

- **Reguły biznesowe:**
  - Użytkownicy systemu podzieleni na dwie grupy: nauczyciele (uprawnieni do wstawiania ocen) i uczniowie (uprawnieni wyłącznie do przeglądania ocen).
  - Możliwość wstawiania ocen cząstkowych.
  - Możliwość generowania raportów i statystyk przez nauczycieli.
- **Poziomy autoryzacji:**
  - **Nauczyciel:** możliwość logowania, wprowadzania ocen, edycji istniejących ocen, przeglądania ocen uczniów i generowania raportów.
  - **Uczeń:** możliwość logowania i przeglądania własnych ocen.
- **Interfejs użytkownika:**
  - Prostota obsługi interfejsu konsolowego dla obu grup użytkowników.
  - Dostosowany ekran logowania z możliwością weryfikacji użytkownika (login i hasło).
- **Funkcjonalności raportowania:**
  - Generowanie zestawień ocen ucznia .
  - Tworzenie statystyk np. średnich ocen.

- **Obsługa błędów:**
  - Komunikaty o niepoprawnym logowaniu (np. błędny login lub hasło).
  - Informowanie nauczyciela o błędach przy wprowadzaniu ocen (np. wartości spoza skali ocen).

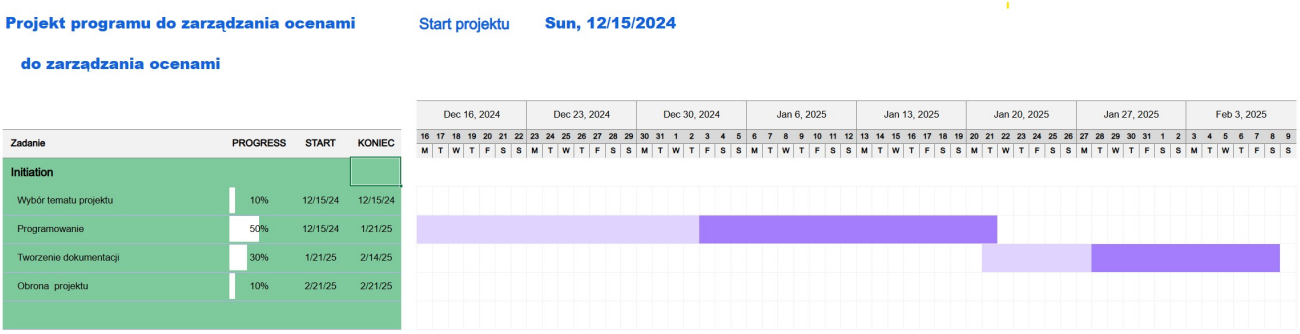
Wymagania niefunkcjonalne

- **Wydajność:**
  - System powinien szybko reagować na polecenia użytkownika).
- **Użyteczność:**
  - Interfejs powinien być intuicyjny, aby użytkownicy mogli bez problemu korzystać z systemu, nawet bez wcześniejszego przeszkolenia.
- **Niezawodność:**
  - System musi być stabilny i odporny na błędy, aby nie dopuścić do utraty danych ocen lub niemożliwości zalogowania się.
- **Utrzymywalność:**
  - Kod źródłowy systemu powinien być napisany w sposób umożliwiający modyfikację i rozwój w przyszłości.
- **Bezpieczeństwo:**
  - Logowanie użytkowników za pomocą loginu i hasła.
  - Ograniczony dostęp do funkcji wprowadzania ocen wyłącznie dla nauczycieli.
  - Ochrona danych uczniów zgodnie z wytycznymi RODO (np. brak możliwości przeglądania ocen innych uczniów).
- **Interoperacyjność:**
  - System powinien być zaprojektowany tak, aby możliwe było jego przyszłe rozszerzenie o wersję z interfejsem graficznym lub integrację z innymi systemami edukacyjnymi.
- **Skalowalność:**
  - System powinien obsługiwać dowolną liczbę użytkowników (uczniów i nauczycieli) bez utraty wydajności.
- **Bezpieczeństwo danych:**
  - Regularne kopie zapasowe ocen i raportów.

1.3 Diagram Ganta

1.3.1 Czym jest diagram ganta?

Diagram Gantta to narzędzie wizualne wykorzystywane w zarządzaniu projektami, które przedstawia harmonogram zadań na osi czasu. Dzięki niemu można zobaczyć, kiedy poszczególne zadania się zaczynają, kończą oraz jakie są między nimi zależności. Umożliwia to efektywne monitorowanie postępów, identyfikowanie potencjalnych opóźnień i optymalizację planu działania.

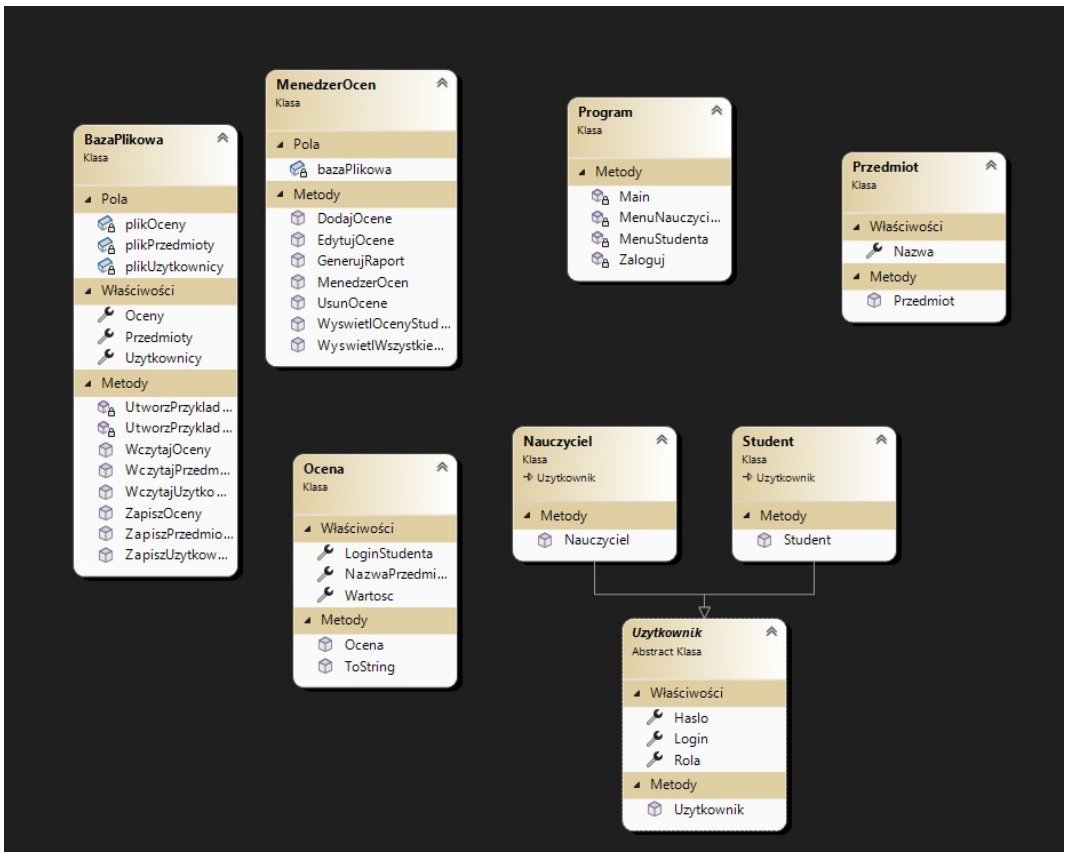


Rysunek 1.1: Diagram Gantta

Rozdział 2

Opis struktury oraz narzędzi użytych w projekcie

2.1 Diagram klas wraz z krótkim opisem



Jest to prosta aplikacja napisana w języku C Platforma NET 8.0 przy użyciu Microsoft Visual Studio Community 2022 (64-bitowy) - Wersja 17.12.1. Aplikacja służąca do zarządzania ocenami studentów. System rozróżnia dwa typy użytkowników: nauczyciela (który może wystawiać, edytować i usuwać oceny) oraz studenta (który może je przeglądać). Dane przechowywane są w plikach tekstowych (za pomocą klasy `BazaPlikowa`), a obsługą operacji na ocenach zajmuje się klasa `MenedzerOcen`. Gówny punkt startowy aplikacji to klasa `Program`, która prezentuje róne menu w zalenoci od zalogowanej roli uytkownika.

1. Struktura klas i ich odpowiedzialności

- **Program**
  - Metody: `Main()`, `MenuNauczyciela()`, `MenuStudenta()`, `Zaloguj()`.
  - Odpowiada za uruchomienie aplikacji oraz wyświetlanie odpowiedniego interfejsu (menu) w zależności od roli użytkownika.
- **Użytkownik (klasa abstrakcyjna)**
  - Pola: `Login`, `Haslo`, `Rola`.
  - Bazowa klasa dla `Student` i `Nauczyciel`. Zawiera wspólne właściwości i ewentualną logikę uwierzytelniania.
- **Student** (dziedziczy po `Użytkownik`)
  - Reprezentuje studenta. Ma dostęp do funkcji wyświetlania własnych ocen.
- **Nauczyciel** (dziedziczy po `Użytkownik`)
  - Reprezentuje nauczyciela. Ma uprawnienia do dodawania, edytowania i usuwania ocen.
- **Ocena**
  - Pola: `LoginStudenta`, `NazwaPrzedmiotu`, `Wartosc`.
  - Reprezentuje pojedynczą ocenę studenta, z konkretną wartością (np. 5.0) i powiązanym przedmiotem.
  - Metoda `ToString()` służy do łatwego formatowania wyświetlania oceny.
- **Przedmiot**
  - Pole: `Nazwa`.
  - Reprezentuje przedmiot, z którym można powiązać oceny.
- **BazaPlikowa**
  - Pola: `plikOceny`, `plikPrzedmioty`, `plikUzytkownicy`.
  - Właściwości: `Oceny`, `Przedmioty`, `Uzytkownicy` (listy/kolekcje).
  - Metody: `WczytajOceny()`, `ZapiszOceny()`, `WczytajPrzedmioty()`, `ZapiszPrzedmioty()`, `WczytajUzytkownikow()`, `ZapiszUzytkownikow()`.
  - Zapewnia komunikację z warstwą plików – wczytuje i zapisuje dane o ocenach, przedmiotach i użytkownikach.
- **MenedzerOcen**
  - Pole: `bazaPlikowa`.
  - Metody: `DodajOcene()`,`EdytujOcene()`,`UsunOcene()`,`WyswietlOcenyStudenta()` `WyswietlWszystkieOceny()`.
  - Korzysta z `BazaPlikowa` do zapisu/odczytu i obsługuje logikę biznesową związaną z ocenami (np. walidacją danych).

2.1.1 Narzędzie użyte do wykonania aplikacji:

Microsoft Visual Studio Community 2022 (64-bitowy) - Wersja 17.12.1.

2.1.2 Minimalne wymagania:

Microsoft nie podaje wprost szczegółowych (sztywnych) parametrów minimalnych dotyczących procesora czy pamięci RAM wyłącznie dla .NET. W praktyce wymogi .NET pokrywają się z minimalnymi wymaganiami wspieranych systemów operacyjnych. Najczęściej oznacza to:

Procesor:

- Minimum 1 GHz (lub szybszy)
- Zwykle wymaga się obsługi instrukcji SSE2 (dotyczy x86/x64)
- Architektura:
- x64 lub x86 (na Windows 10/11 64-bit najczęściej używa się x64)

Pamięć RAM:

- System operacyjny Windows 10/11 wymaga co najmniej 2 GB RAM do komfortowego działania (4 GB jest silnie zalecane).
- Miejsce na dysku:
- Około 2–5 GB na .NET SDK (w zależności od wersji, pakietów i narzędzi).
- Sam .NET Runtime zajmuje zdecydowanie mniej (kilkaset MB).

Z punktu widzenia Microsoft, **minimalne** wymagania są w dużej mierze równe **minimalnym wymaganiom systemu Windows**, na którym ma zostać zainstalowany .NET. Dla przykładu:

- Windows 10 (64-bit) wymaga procesora co najmniej 1 GHz, z obsługą PAE, NX i SSE2.
- Windows 11 (64-bit) również wymaga minimum 1 GHz, 2 rdzeni, 4 GB RAM itd.

2.2 Zarządzanie danymi oraz baza danych

Aplikacja, zgodnie z przedstawionym diagramem i opisem, korzysta z warstwy dostępu do danych realizowanej przez klasę `BazaPlikowa`. W jej obecnej implementacji wszystkie informacje przechowywane są w plikach tekstowych. Dzięki temu rozwiązanie jest proste we wdrożeniu i nie wymaga dodatkowych usług (np. serwera bazy danych). Poniżej przedstawiono obecną implementację opartą o pliki.

2.2.1 Implementacja (pliki tekstowe)

- **Klasa BazaPlikowa**:
  - Odpowiada za wczytywanie i zapisywanie danych z/do plików.
  - Posiada osobne metody do obsługi użytkowników (`WczytajUzytkownikow()`, `ZapiszUzytkownikow()`), przedmiotów (`WczytajPrzedmioty()`, `ZapiszPrzedmioty()`) oraz ocen (`WczytajOceny()` `ZapiszOceny()`).
  - Dane wczytane z plików są przechowywane w polach/ właściwościach klasy, np. `Oceny`, `Uzytkownicy`, `Przedmioty` (najczęściej jako listy lub kolekcje).
- **Zapisywanie danych**:
  - Przy każdej istotnej operacji (np. dodawanie nowej oceny, modyfikacja istniejącej) `MenedzerOcen` korzysta z metod `BazaPlikowa`, by zaktualizować odpowiedni plik tekstowy.

- **Struktura danych**:
  - `Ocena` (zawiera np. `LoginStudenta`, `NazwaPrzedmiotu`, `Wartosc`).
  - `Uzytkownik` (baza dla `Student` i `Nauczyciel`, zawiera `Login`, `Haslo`, `Rola`).
  - `Przedmiot` (np. `Nazwa`).
- **Zalety i wady podejścia plikowego**:
  - *Zalety*: łatwa konfiguracja, brak konieczności instalacji dodatkowych serwerów, prosta migracja między stanowiskami.
  - *Wady*: brak zaawansowanych mechanizmów transakcyjności i jednoczesnego dostępu (możliwe problemy z kolizjami przy równoczesnym zapisie), ograniczona wydajność przy dużej liczbie danych.

Rozdział 3

System kontroli wersji

3.1 Github

Wszystkie pliki projektowe, w tym dokumentacja w formacie LaTeX, zostały umieszczone w repozytorium GitHub. GitHub to popularna platforma hostingowa, która opiera się na systemie kontroli wersji Git i umożliwia efektywne zarządzanie kodem oraz dokumentacją projektową. Dzięki GitHub możliwa jest transparentna współpraca, monitorowanie zmian i zgłaszanie problemów przez zespół, co znacznie usprawnia proces rozwoju projektu.

link do repozytorium

Rozdział 4

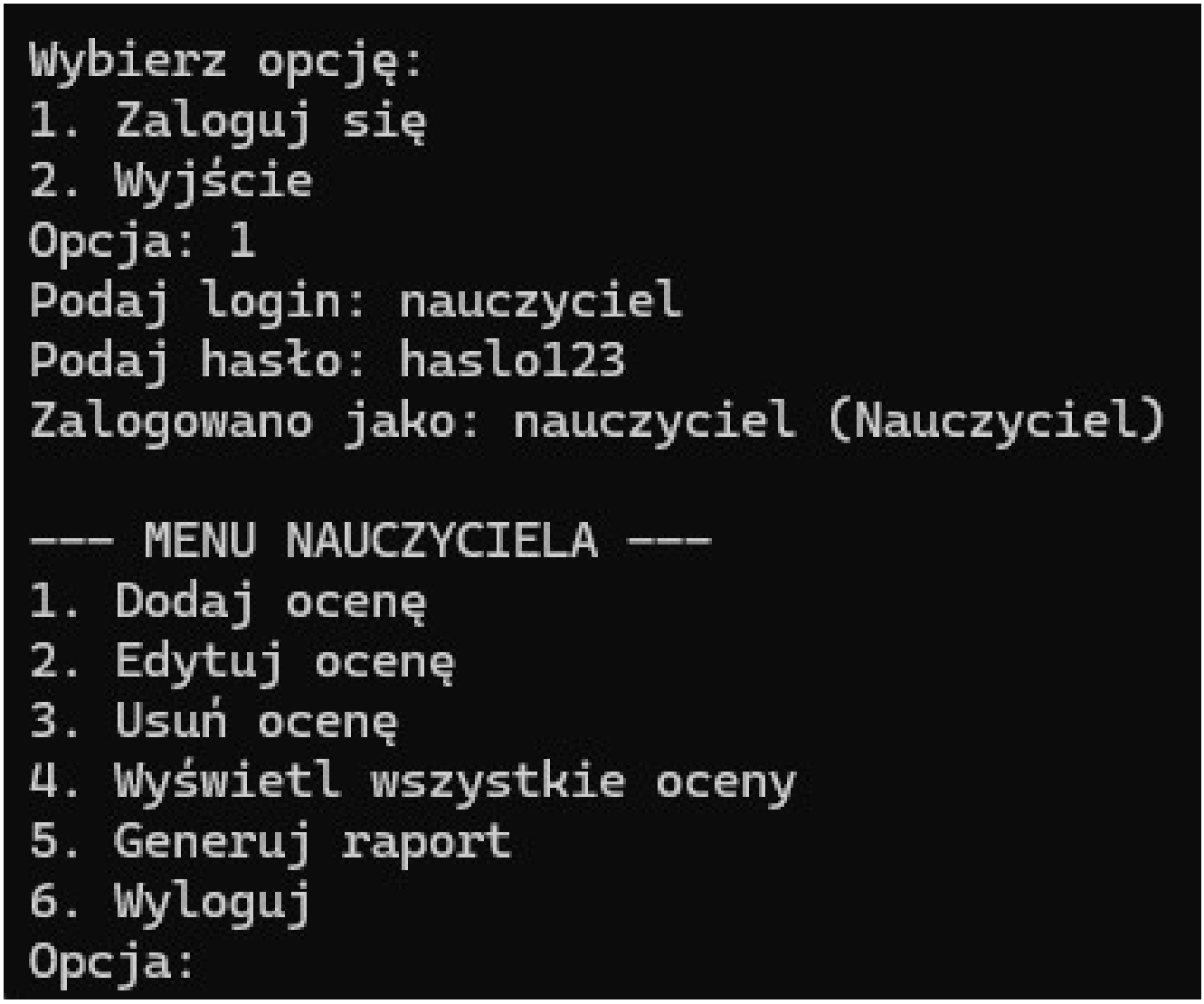
Prezentacja warstwy użytkowej projektu

4.1 Menu główne logowania

```
1
2  Console.WriteLine("Elektroniczny System Oceniania");
3  bool wyjscie = false;
4
5  while (!wyjscie)
6  {
7      Console.WriteLine("\nWybierz opcja:");
8      Console.WriteLine("1. Zaloguj sie");
9      Console.WriteLine("2. Wyjscie");
10     Console.Write("Opcja: ");
11
12     string opcja = Console.ReadLine();
13     switch (opcja)
14     {
15         case "1":
16             Uzytkownik zalogowany = Zaloguj(bazaPlikowa.Uzytkownicy);
17             if (zalogowany != null)
18             {
19                 Console.WriteLine("Zalogowano jako: " + zalogowany.Login +
                       " (" + zalogowany.Rola + ")");
20                 if (zalogowany is Nauczyciel nauczyciel)
21                 {
22                     MenuNauczyciela(nauczyciel, menedzerOcen);
23                 }
24                 else if (zalogowany is Student student)
25                 {
26                     MenuStudenta(student, menedzerOcen);
27                 }
28             }
29             else
30             {
31                 Console.WriteLine("Niepoprawny login lub haslo.");
32             }
33             break;
34         case "2":
35             wyjscie = true;
36             break;
37         default:
38             Console.WriteLine("Nieznana opcja.");
39             break;
40     }
41 }
42
43 Console.WriteLine("Zamykam program...");
```

Listing 4.1: Menu główne kod C #

4.2 Menu nauczyciel



1. Metoda `DodajOcene()`

Opis:

- **Cel:** Umożliwia dodanie nowej oceny ucznia.
- **Kroki działania:**
  1. Pobiera z konsoli login ucznia i wyszukuje go w bazie.
  2. Jeśli uczeń zostanie znaleziony, wyświetla dostępne przedmioty.
  3. Użytkownik podaje nazwę przedmiotu oraz wartość oceny.
  4. Wartość oceny jest walidowana (musi być w zakresie 1-6).
  5. Po poprawnej walidacji tworzy nową ocenę, dodaje ją do bazy i zapisuje zmiany.

```
public void DodajOcene()
{
    Console.Write("Podaj login ucznia: ");
    string loginStudenta = Console.ReadLine();
    var uczen = bazaPlikowa.Uzytkownicy
        .FirstOrDefault(u => u.Rola == "Student" && u.Login ==
           loginStudenta);
    if (uczen == null)
    {
        Console.WriteLine("Nie znaleziono ucznia.");
        return;
    }
    Console.WriteLine("Dostepne przedmioty:");
    foreach (var przedm in bazaPlikowa.Przedmioty)
    {
        Console.WriteLine("- " + przedm.Nazwa);
    }

    Console.Write("Podaj nazwe przedmiotu: ");
    string nazwaPrzedmiotu = Console.ReadLine();

    var przedmiot = bazaPlikowa.Przedmioty
        .FirstOrDefault(p => p.Nazwa.Equals(nazwaPrzedmiotu,
           StringComparison.OrdinalIgnoreCase));

    if (przedmiot == null)
    {
        Console.WriteLine("Nie znaleziono przedmiotu.");
        return;
    }

    Console.Write("Podaj wartosc oceny (1-6): ");
    string wartoscString = Console.ReadLine();

    if (double.TryParse(wartoscString, out double wartosc))
    {
        if (wartosc < 1 || wartosc > 6)
        {
            Console.WriteLine("Bledny zakres oceny (1-6).");
            return;
        }

        Ocena nowa = new Ocena(loginStudenta, przedmiot.Nazwa, wartosc);
        bazaPlikowa.Oceny.Add(nowa);
        bazaPlikowa.ZapiszOceny();
        Console.WriteLine("Ocena dodana.");
    }
    else
    {
        Console.WriteLine("Niepoprawna wartosc oceny.");
    }
}
```

Listing 4.2: Metoda dodaj ocene

2. Metoda `EdytujOcene()`

Opis:

- **Cel:** Umożliwia edycję (modyfikację) istniejącej oceny ucznia.
- **Kroki działania:**
  1. Pobiera z konsoli login ucznia i wyszukuje wszystkie oceny przypisane temu uczniowi.
  2. Jeśli oceny istnieją, wyświetla je numerowane.
  3. Użytkownik wybiera numer oceny, którą chce zmodyfikować.
  4. Następnie podaje nową wartość oceny, która podlega walidacji (zakres 1-6).
  5. Po poprawnej walidacji aktualizowana jest wartość oceny, a zmiany są zapisywane.

```
public void EdytujOcene()
{
    Console.Write("Podaj login ucznia: ");
    string loginStudenta = Console.ReadLine();

    var ocenyUcznia = bazaPlikowa.Oceny
        .Where(o => o.LoginStudenta == loginStudenta).ToList();

    if (!ocenyUcznia.Any())
    {
        Console.WriteLine("Brak ocen dla podanego ucznia.");
        return;
    }

    for (int i = 0; i < ocenyUcznia.Count; i++)
    {
        Console.WriteLine((i + 1) + ". " + ocenyUcznia[i]);
    }

    Console.Write("Wybierz numer oceny: ");
    string indeksString = Console.ReadLine();

    if (!int.TryParse(indeksString, out int indeks) || indeks < 1 || indeks
        > ocenyUcznia.Count)
    {
        Console.WriteLine("Niepoprawny numer.");
        return;
    }

    var edytowana = ocenyUcznia[indeks - 1];
    Console.Write("Podaj nowa wartosc (1-6): ");
    string nowaString = Console.ReadLine();

    if (double.TryParse(nowaString, out double nowa))
    {
        if (nowa < 1 || nowa > 6)
        {
            Console.WriteLine("Bledny zakres oceny.");
            return;
        }

        edytowana.Wartosc = nowa;
        bazaPlikowa.ZapiszOceny();
        Console.WriteLine("Ocena zaktualizowana.");
    }
    else
    {
        Console.WriteLine("Niepoprawna wartosc oceny.");
    }
}
```

Listing 4.3: Metoda Edytuj Ocene

3. Metoda `UsunOcene()`

Opis:

- **Cel:** Umożliwia usunięcie wybranej oceny ucznia z systemu.
- **Kroki działania:**
  1. Pobiera login ucznia, aby wyszukać wszystkie oceny przypisane temu uczniowi.
  2. Jeśli oceny istnieją, wyświetla je numerowane.
  3. Użytkownik wybiera numer oceny, którą chce usunąć.
  4. Po potwierdzeniu usunięcia ocena jest usuwana z bazy, a zmiany są zapisywane.

```
public void UsunOcene()
{
    Console.Write("Podaj login ucznia: ");
    string loginStudenta = Console.ReadLine();

    var ocenyUcznia = bazaPlikowa.Oceny
        .Where(o => o.LoginStudenta == loginStudenta).ToList();

    if (!ocenyUcznia.Any())
    {
        Console.WriteLine("Brak ocen dla podanego ucznia.");
        return;
    }

    for (int i = 0; i < ocenyUcznia.Count; i++)
    {
        Console.WriteLine((i + 1) + ". " + ocenyUcznia[i]);
    }

    Console.Write("Wybierz numer oceny: ");
    string indeksString = Console.ReadLine();

    if (!int.TryParse(indeksString, out int indeks) || indeks < 1 || indeks
        > ocenyUcznia.Count)
    {
        Console.WriteLine("Niepoprawny numer.");
        return;
    }

    var doUsuniecia = ocenyUcznia[indeks - 1];
    bazaPlikowa.Oceny.Remove(doUsuniecia);
    bazaPlikowa.ZapiszOceny();
    Console.WriteLine("Ocena usunieta.");
}
```

Listing 4.4: Metoda Usuń ocene

4. Metoda `WyswietlWszystkieOceny()`

Opis:

- **Cel:** Wyświetlenie wszystkich ocen zapisanych w systemie.
- **Kroki działania:**
  1. Sprawdza, czy w systemie znajdują się jakiekolwiek oceny.
  2. Jeśli nie, informuje o braku ocen.
  3. W przeciwnym przypadku iteruje przez listę ocen i wyświetla każdą z nich.

```
public void WyswietlWszystkieOceny()
{
if (!bazaPlikowa.Oceny.Any())
    {
        Console.WriteLine("Brak ocen w systemie.");
        return;
    }
    foreach (var ocena in bazaPlikowa.Oceny)
    {
        Console.WriteLine(ocena);
    }
}
```

Listing 4.5: Metoda WyswietlWszystkieOceny

5. Metoda `WyswietlOcenyStudenta(Student student)`

Opis:

- **Cel:** Wyświetlenie ocen konkretnego studenta.
- **Kroki działania:**
  1. Wyszukuje oceny przypisane do podanego studenta (na podstawie loginu).
  2. Jeśli student nie ma żadnych ocen, informuje o tym.
  3. W przeciwnym razie iteruje przez oceny i wyświetla przedmiot oraz wartość oceny.

```
public void WyswietlOcenyStudenta(Student student)
{
    var ocenyUcznia = bazaPlikowa.Oceny
        .Where(o => o.LoginStudenta == student.Login).ToList();
    if (!ocenyUcznia.Any())
    {
        Console.WriteLine("Brak ocen.");
        return;
    }
    foreach (var ocena in ocenyUcznia)
    {
        Console.WriteLine("Przedmiot: " + ocena.NazwaPrzedmiotu + ", Ocena:
            " + ocena.Wartosc);
    }
}
```

Listing 4.6: Metoda wyświetl oceny studenta

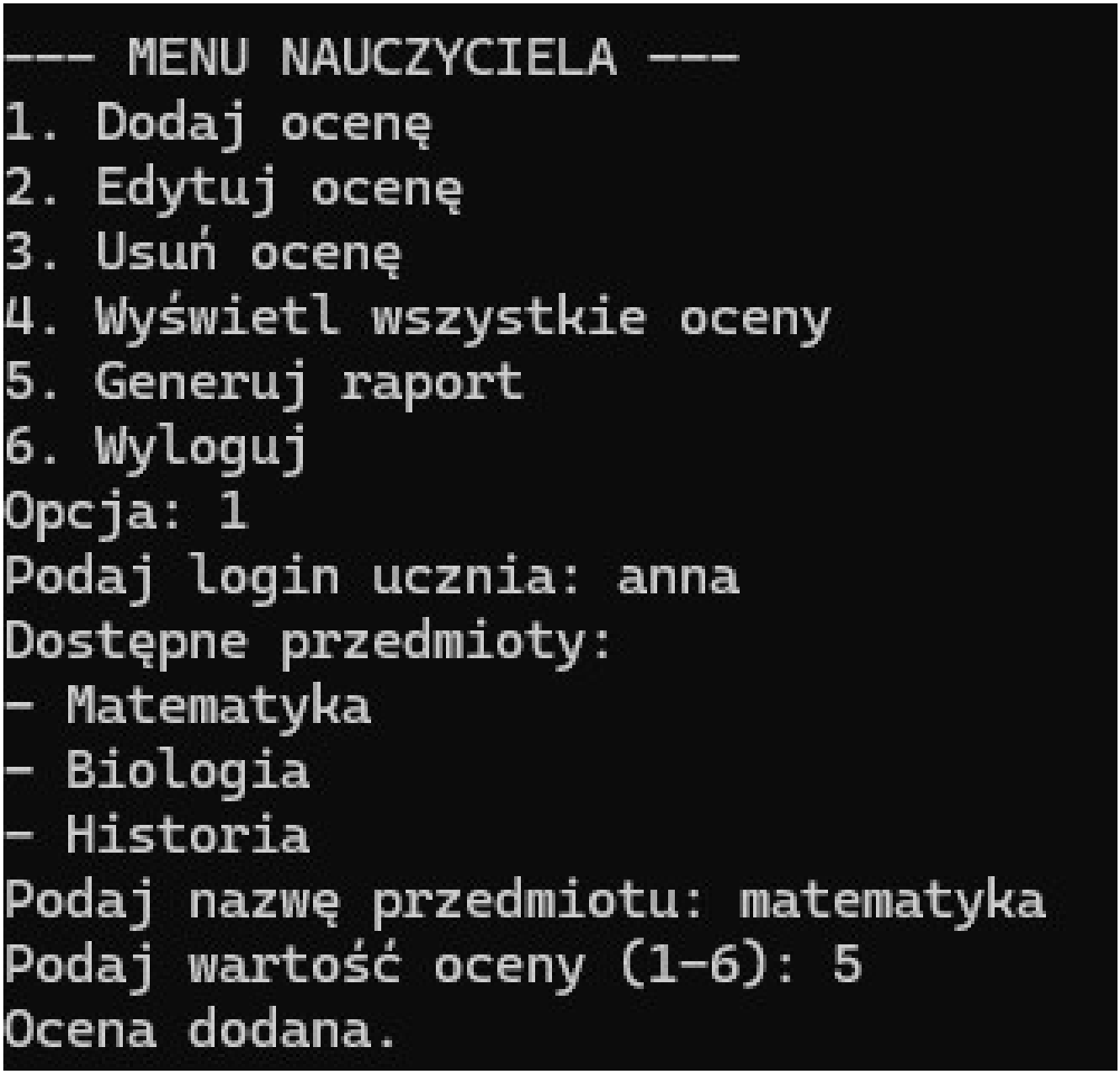
6. Metoda `GenerujRaport()`

Opis:

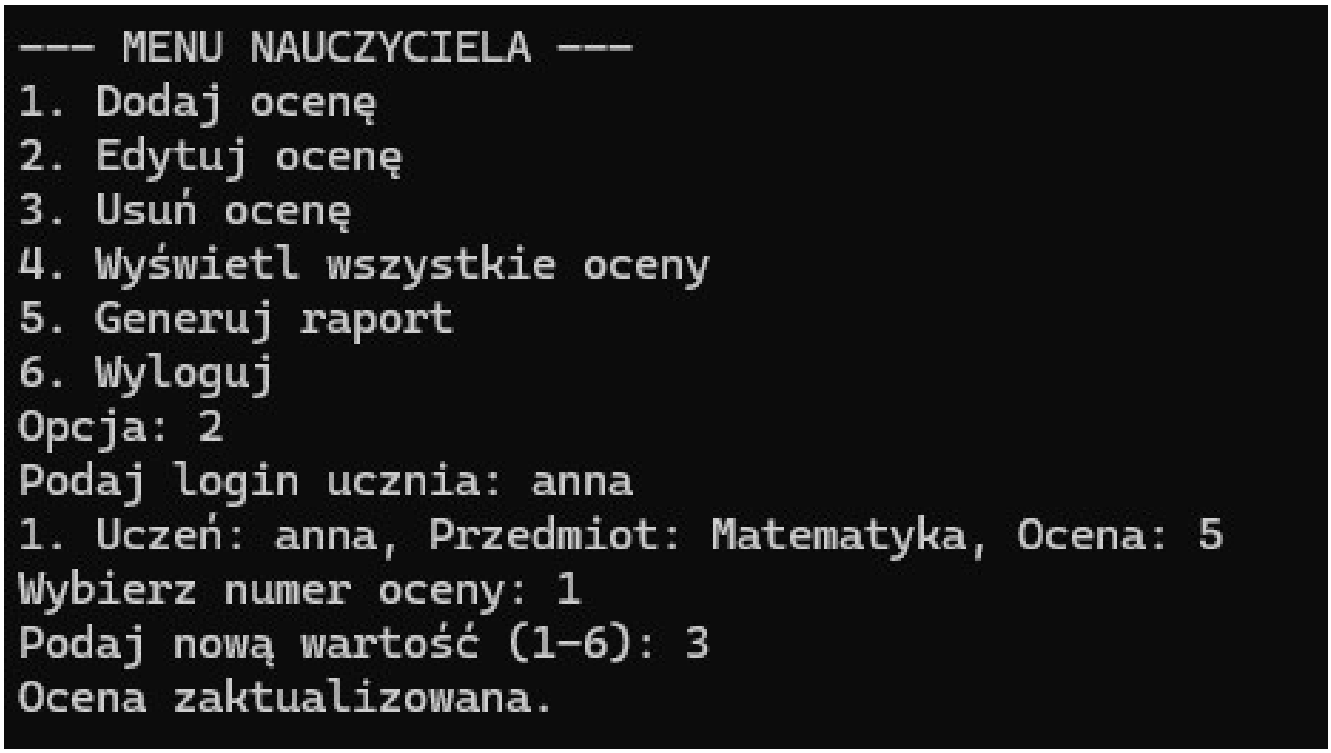
- **Cel:** Generowanie raportu ocen w systemie.
- **Kroki działania:**
  1. Sprawdza, czy w systemie istnieją oceny.
  2. Oblicza średnią wszystkich ocen.
  3. Grupuje oceny według przedmiotu i dla każdej grupy oblicza średnią ocen.
  4. Wyświetla średnią ocen całkowitą oraz średnią dla każdego przedmiotu.

```
public void GenerujRaport()
{
    if (!bazaPlikowa.Oceny.Any())
    {
        Console.WriteLine("Brak ocen w systemie.");
        return;
    }

    double sredniaWszystkich = bazaPlikowa.Oceny.Average(o => o.Wartosc);
    Console.WriteLine("Srednia wszystkich ocen: " + sredniaWszystkich.
       ToString("F2"));

    var grupy = bazaPlikowa.Oceny
        .GroupBy(o => o.NazwaPrzedmiotu)
        .Select(g => new { Przedmiot = g.Key, Srednia = g.Average(x => x.
           Wartosc) });

    foreach (var grupa in grupy)
    {
        Console.WriteLine("Przedmiot: " + grupa.Przedmiot + ", Srednia: " +
            grupa.Srednia.ToString("F2"));
    }
}
```

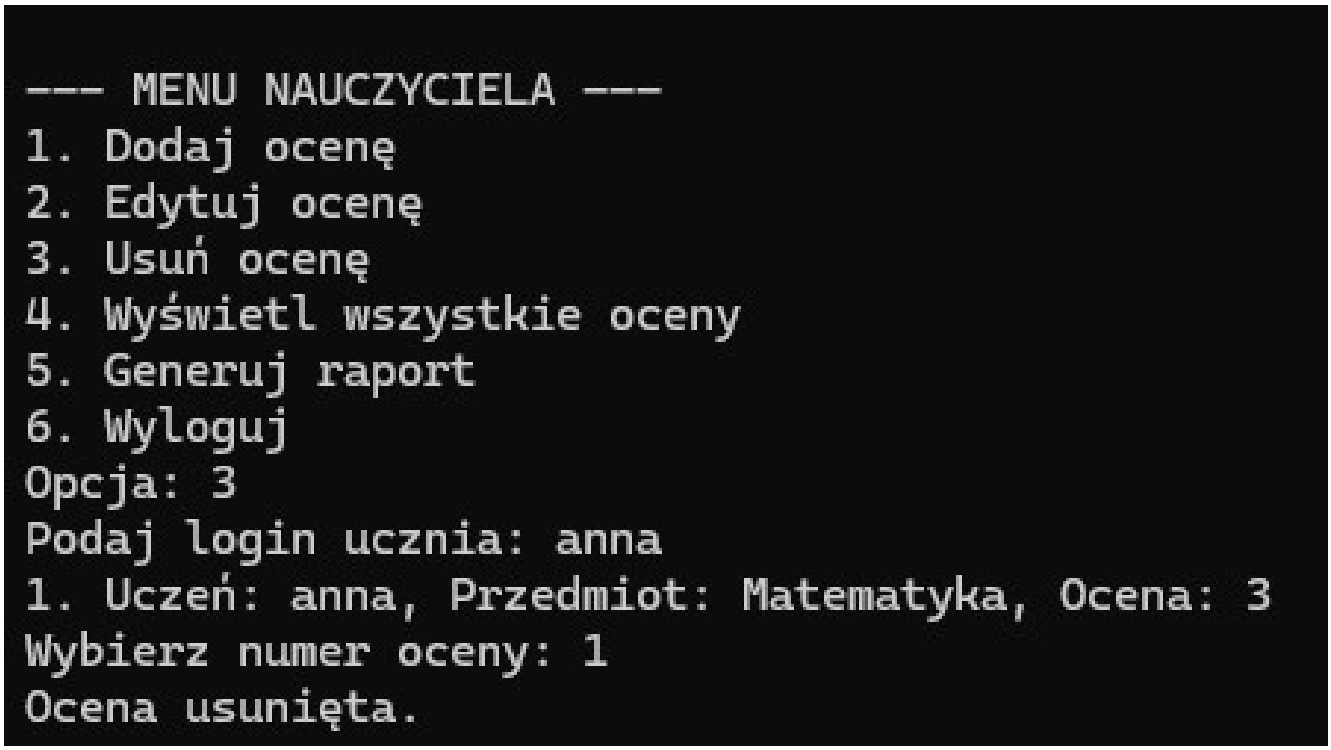
Listing 4.7: Metoda GenerujRaport



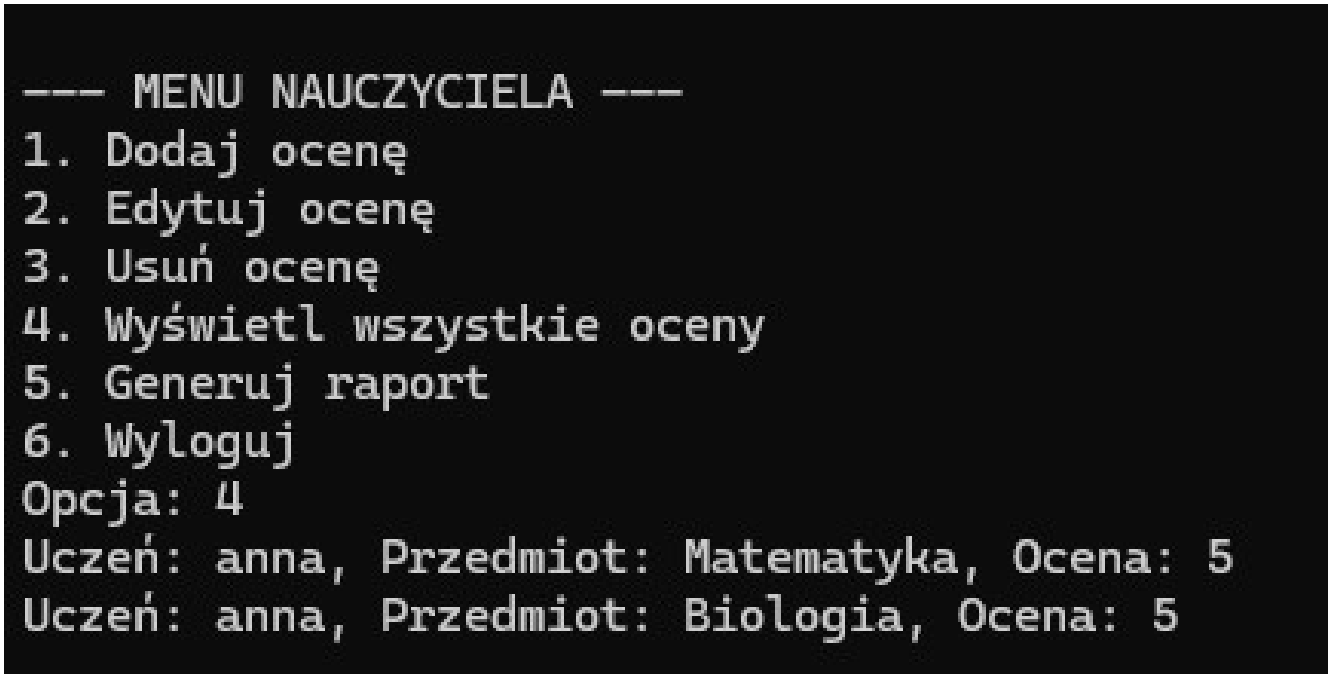
Rysunek 4.1: Interfejs metody DodajOcene()



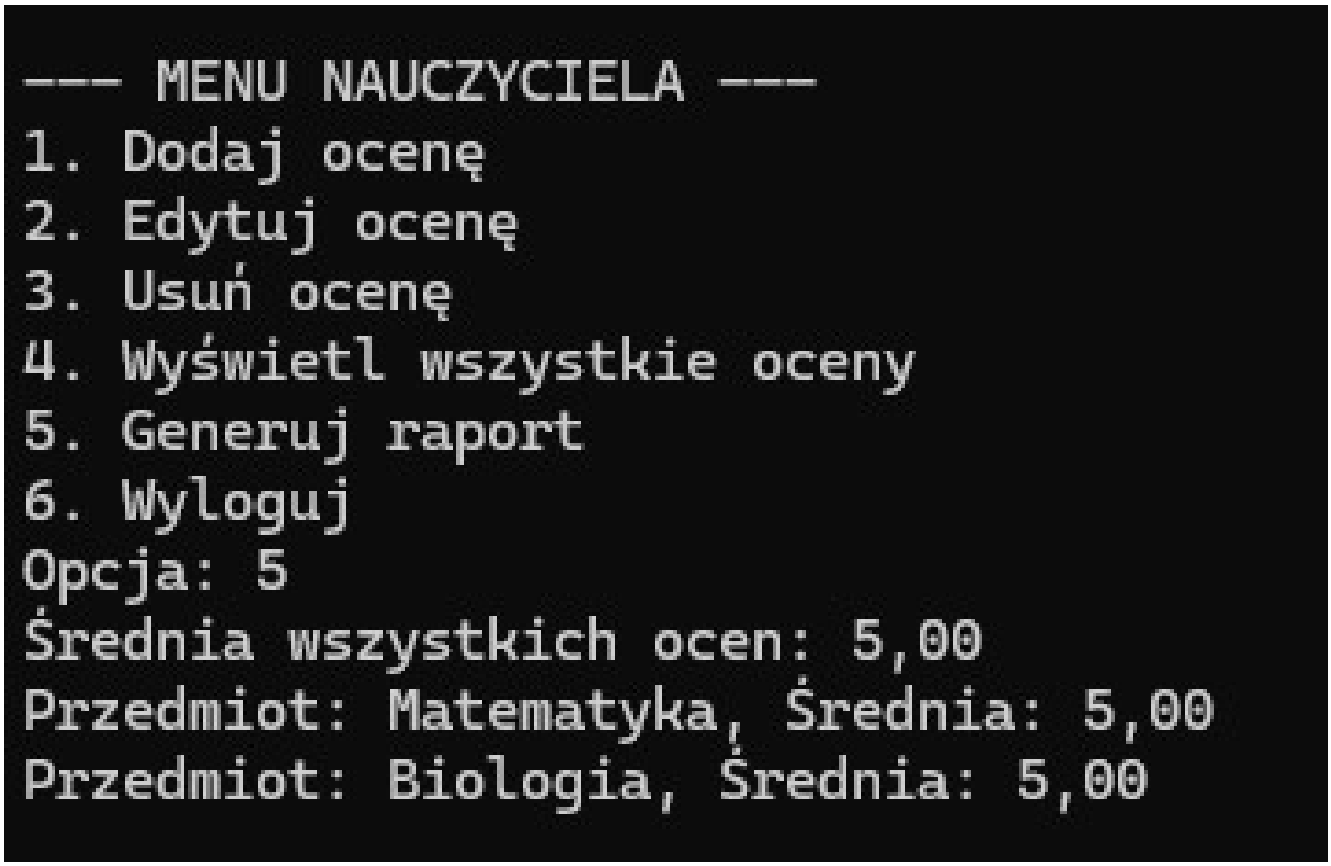
Rysunek 4.2: Interfejs metodyEdytujOcene()



Rysunek 4.3: Interfejs metody UsunOcene()



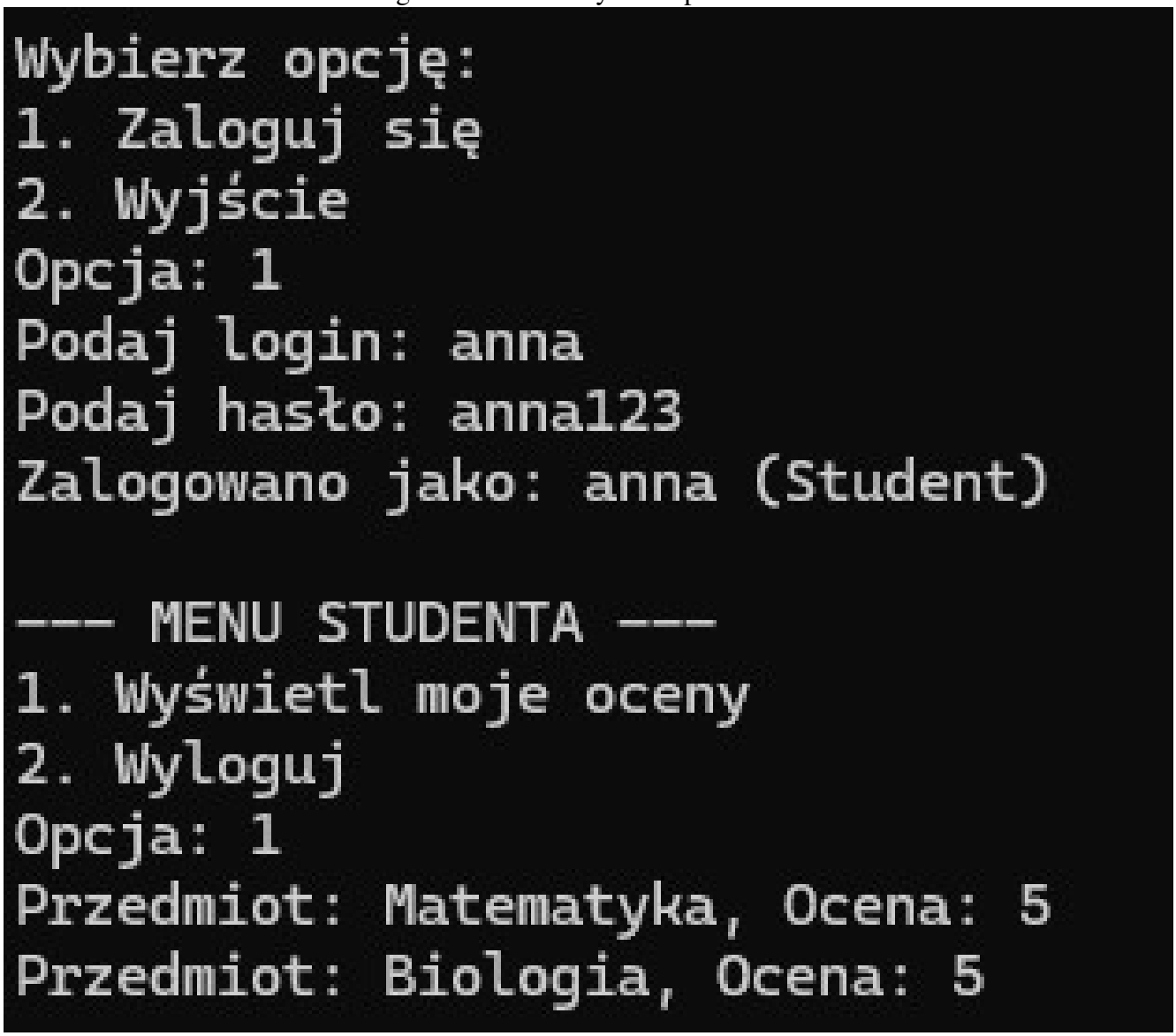
Rysunek 4.4: Interfejs metody WyswietlWszystkieOceny()



Rysunek 4.5: Interfejs metody GenerujRaport()

4.3 Menu Ucznia

W menu ucznia wywołujemy metodę WyswietlOcenyStudenta() która pokazuje wszystkie oceny danego studenta z wszystkich przedmiotów



Rozdział 5

Podsumowanie

W ramach projektu Elektronicznego Systemu Oceniania zrealizowałem szereg kluczowych funkcjonalności, które umożliwiają zarządzanie ocenami uczniów. Opracowany system, napisany w języku C#, wspiera logowanie użytkowników z uwzględnieniem różnych ról (nauczyciel oraz student), co pozwala na dostosowanie interfejsu oraz dostępnych operacji do potrzeb poszczególnych grup użytkowników.

Do najważniejszych zrealizowanych prac należą:

- Dodawanie ocen – umożliwienie nauczycielom wprowadzania ocen dla uczniów poprzez intuicyjny interfejs konsolowy.
- Edycja i usuwanie ocen – wprowadzenie mechanizmów modyfikacji oraz usuwania ocen, co pozwala na korektę ewentualnych błędów.
- Wyświetlanie ocen – prezentacja ocen zarówno w postaci listy wszystkich wpisów, jak i ocen konkretnego ucznia.
- Generowanie raportów – opracowanie funkcjonalności obliczania średnich ocen ogółem oraz dla poszczególnych przedmiotów, co wspiera analizę wyników edukacyjnych.
- W wyniku prac powstał system umożliwiający sprawną obsługę oceniania, który jest łatwy w utrzymaniu i dalszym rozwoju.

Planowane dalsze prace rozwojowe:

- Rozbudowa interfejsu użytkownika – przejście od interfejsu konsolowego do graficznego (GUI), co zwiększy użyteczność i dostępność systemu.
- Integracja z relacyjną bazą danych – zamiast wykorzystywania plików do przechowywania danych, planowana jest implementacja systemu bazodanowego, co poprawi skalowalność oraz bezpieczeństwo danych.
- Rozszerzenie funkcjonalności – dodanie modułów analizy statystycznej wyników, automatycznego generowania szczegółowych raportów oraz funkcji powiadomień dla użytkowników.
- Poprawa bezpieczeństwa i walidacji danych – implementacja dodatkowych mechanizmów zabezpieczających przed błędami oraz nieautoryzowanym dostępem.

Podsumowując, zrealizowany projekt stanowi solidną bazę do dalszego rozwoju systemu oceniania, a planowane usprawnienia pozwolą na jeszcze lepsze dostosowanie aplikacji do potrzeb współczesnych placówek edukacyjnych.

Bibliografia

-Dokumentacja C# : https://learn.microsoft.com/pl-pl/dotnet/csharp/
-Jakub Kozera Kurs języka C# od podstaw z zadaniami praktycznymi, omawiający kluczowe zagadnienia języka C# dla platformy .NET Ostania aktualizacja 3 luty 2025
-Materiały z labolatoriów oraz wykładów

Spis rysunków

Spis tabel