

# System Obsługi Studenta

Norbert Budzyński

Krzysztof Kogut

## Dziedzina problemu

Dziedziną problemu jest System Obsługi Studenta na uczelni. Projekt wykonany jest w procesie iteracyjnym, co pozwala na sprawną refaktoryzację projektu, a także dbanie o poprawność, proces wytwórczy, utrzymanie oraz rozszerzanie projektu o kolejne moduły, jeżeli proces tego wymaga.

## Diagram przypadków użycia dla systemu

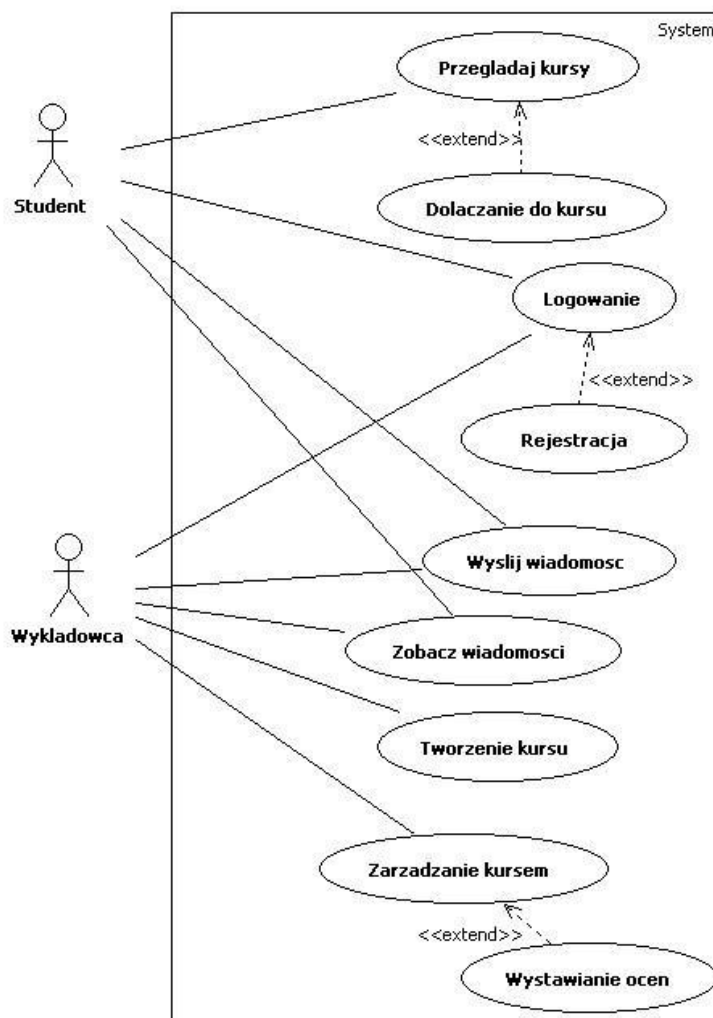


Diagram przypadków użycia przyjmuje dwóch aktorów: Studenta oraz Wykładowcę. Obie te osoby posiadają zarówno specjalne funkcje, takie jak np. przeglądanie kursów czy tworzenie takiego, jednak nie zapomnieć możemy o cechach wspólnych takich jak przeglądanie wiadomości czy logowanie.

## Przypadki użycia dla studenta

Przypadek użycia	Przeglądanie kursów
Scenariusz	Wyświetlenie listy dostępnych kursów dla studenta
Warunki początkowe	Student jest zalogowany do systemu
Opis	Student żąda listę dostępnych kursów. System pobiera kursy na które student może się zapisać. System wyświetla listę kursów.
Warunki końcowe	Wyświetlona jest lista dostępnych kursów

Przypadek użycia	Dołączanie do kursu
Scenariusz	Dołączanie do kursu w oparciu o listę dostępnych kursów
Warunki początkowe	Student jest zalogowany do systemu
Opis	Student wybiera kurs, na który uczęszcza i zapisuje się jako uczestnik kursu
Warunki końcowe	Student jest zapisany do kursu

## Przypadki użycia dla wykładowcy

Przypadek użycia	Tworzenie kursu
Scenariusz	Tworzenie kursu przez wykładowcę
Warunki początkowe	Wykładowca jest zalogowany
Opis	Wykładowca może stworzyć kurs dotyczący swojego przedmiotu
Warunki końcowe	Wykładowca tworzy kurs

Przypadek użycia	Zarządzanie kursem
Scenariusz	System zarządzania kursem przez wykładowcę
Warunki początkowe	Wykładowca jest zalogowany
Opis	Wykładowca zarządza stworzonym przez siebie kursem, zmienia informacje o nim i dokonuje zmian w jego obrębie
Warunki końcowe	Wykładowca zarządza w pełni kursem

Przypadek użycia	Wystawianie ocen
Scenariusz	Wykładowca wystawia ocenę końcową z kursu
Warunki początkowe	Wykładowca jest zalogowany
Opis	Wykładowca wysyła żądanie wystawienia oceny dla studenta w ramach kursu. System sprawdza czy wykładowca jest prowadzącym wybranego kursu. System sprawdza czy student jest zapisany do wybranego kursu. System dodaje ocenę.
Warunki końcowe	Nowa ocena jest wystawiona

## Przypadki użycia wspólne

Przypadek użycia	Logowanie
Scenariusz	Pomyślne zalogowanie się do systemu
Warunki początkowe	Użytkownik nie jest zalogowany do systemu
Opis	Użytkownik wysyła żądanie zalogowania się do systemu używając podanych poświadczeń. Jeśli poświadczenia są prawidłowe, użytkownik zostaje zalogowany do systemu.
Warunki końcowe	Użytkownik jest zalogowany w systemie

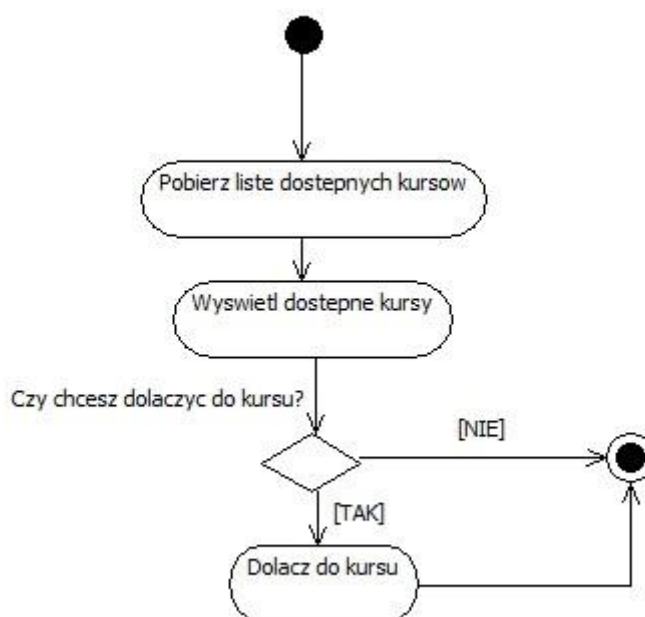
Przypadek użycia	Rejestracja
Scenariusz	Pomyślne zarejestrowanie się w systemie
Warunki początkowe	Użytkownik nie jest zalogowany do systemu
Opis	Użytkownik wysyła żądanie zarejestrowania się w systemie. Użytkownik o podanych poświadczeniach nie istnieje. System tworzy nowego użytkownika o podanych poświadczeniach.
Warunki końcowe	Użytkownik został zarejestrowany w systemie

Przypadek użycia	Wysyłanie wiadomości
Scenariusz	Wysłanie wiadomości do innego użytkownika systemu
Warunki początkowe	Użytkownik systemu jest zalogowany
Opis	Użytkownik systemu żąda wysłania wiadomości. System sprawdza czy odbiorca istnieje. System zapisuje wiadomość w bazie.
Warunki końcowe	Wiadomość została wysłana

Przypadek użycia	Przeglądanie wiadomości
Scenariusz	Użytkownik systemu przegląda wiadomości wysłane do niego
Warunki początkowe	Użytkownik systemu jest zalogowany
Opis	Użytkownik systemu żąda swoich wiadomości. System pobiera wiadomości których adresatem jest użytkownik. System wyświetla pobrane wiadomości.
Warunki końcowe	Wiadomość została wysłana

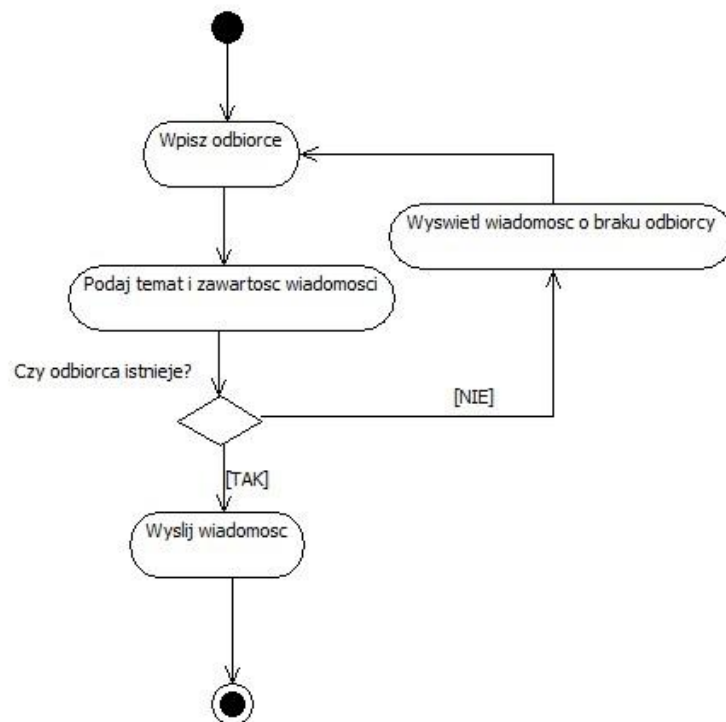
## Wybrane diagramy aktywności

### Wyświetlanie kursów



Wyświetlanie kursów zdefiniowane przez nas działa w sposób następujący: Uczelnia przechowuje informacje o dostępnych kursach, która zostaje pobrana, a następnie wyświetlana użytkownikowi. Następnie pojawia się decyzja, czy student chce dołączyć do kursu. W przypadku, kiedy student nie chce dołączać do kursu algorytm przechodzi do stanu końcowego, a w przeciwnym wypadku uruchamiana jest metoda która służy zapisywaniu się na dany kurs wybrany później przez użytkownika.

## Wysyłanie wiadomości



Wysyłanie wiadomości działa poprzez prosty schemat: Wpisujemy odbiorcę, a następnie temat i zawartość wiadomości, którą użytkownik chce wysłać. Następnie sprawdzamy, czy dany odbiorca istnieje. Jeżeli takowy istnieje, to wiadomość zostaje wysłana do odbiorcy. W przeciwnym wypadku jednakże wyświetlana jest wiadomość o braku odbiorcy, a program “cofa się” do momentu wpisywania odbiorcy wiadomości.

## Zarządzanie kursem

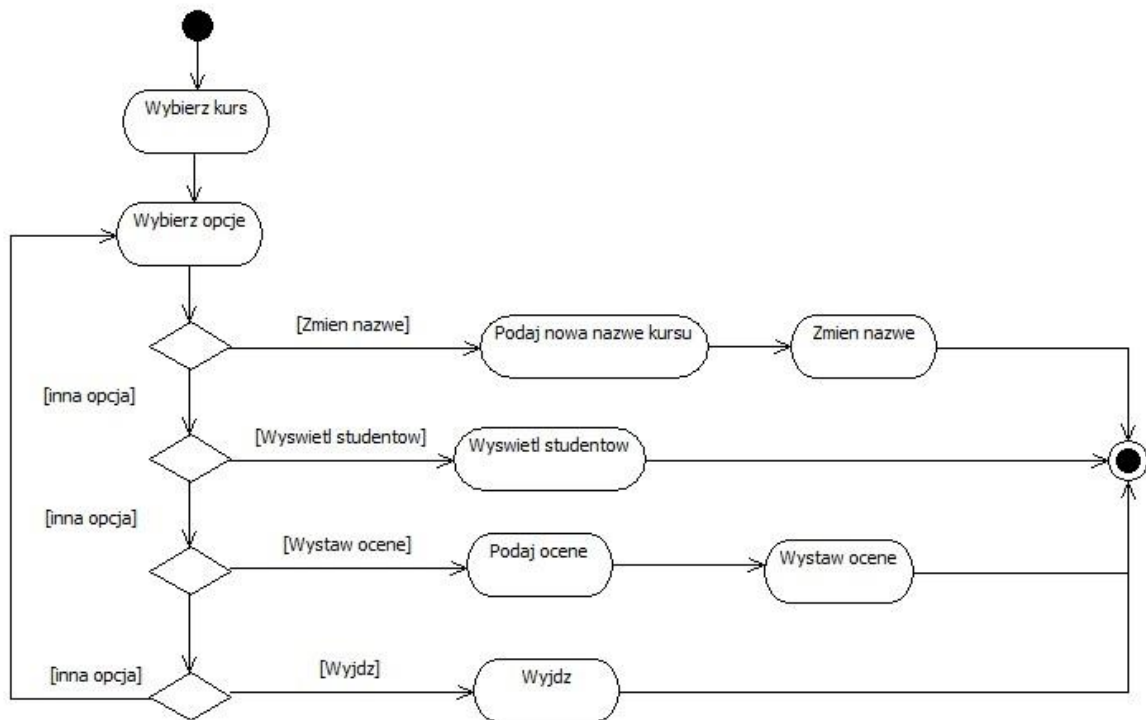
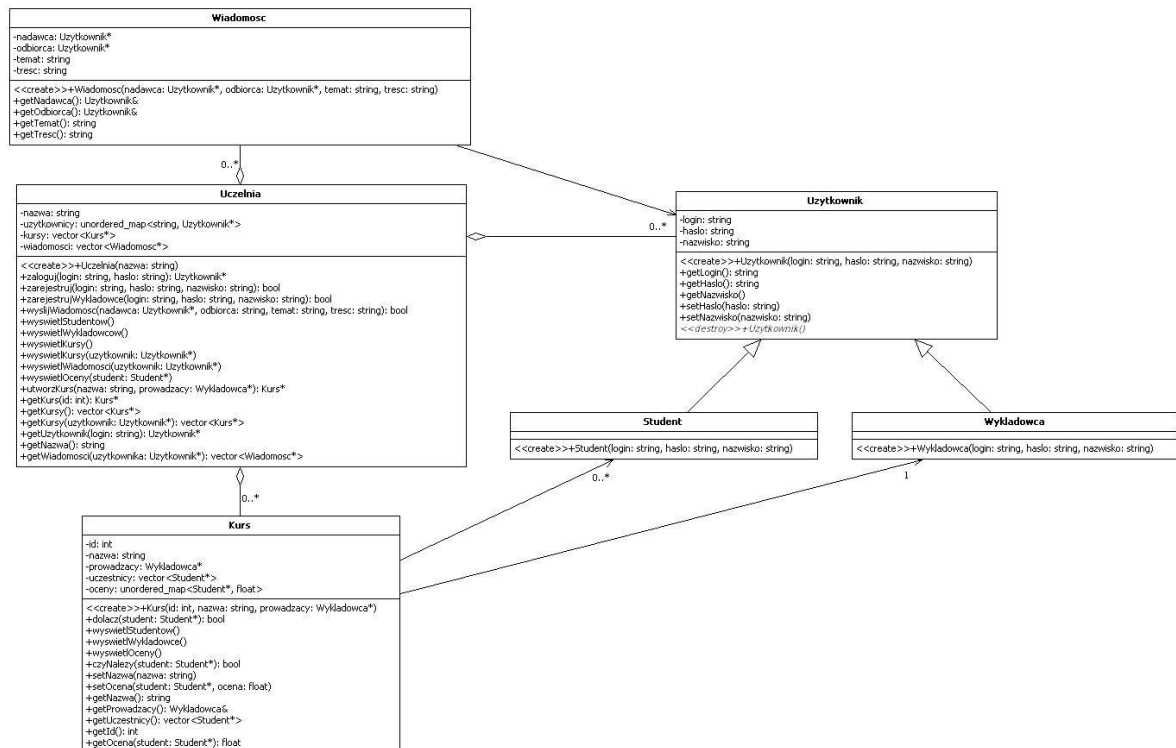


Diagram aktywności dla zarządzania kursem działa poprzez wybór użytkownika (w tym przypadku wykładowcy) odpowiedniej opcji z menu w systemie. Następnie sprawdzane są opcje, które dla odpowiednich wyborów wywołują odpowiednie metody dla kursu stworzonego przez wykładowcę. Po wykonaniu jakichkolwiek zmian, program zmienia swój stan na końcowy. W przeciwnym wypadku, program wraca do momentu bloku wybrania opcji przez użytkownika.



# Struktura logiczna projektu



Struktura logiczna projektu jest zbudowana w sposób następujący:

“Sercem” całego systemu jest klasa Uczelnia, której zadaniem jest dbanie o dostarczenie usług takich, jak: system logowania, rejestracja, wyświetlanie dostępnych kursów oraz wiadomości użytkownika itp. Uczelnia jest zagregowana z klasą Użytkownik (klasą, która posiada dwie klasy podrzędne: Studenta i Wykładowcę), Kurs (gdzie znajduje się np. system wystawiania ocen studentom) oraz Wiadomość (który służy jako klasa przechowująca informacje o odbiorcy, nadawcy, temacie oraz treści wiadomości). Asocjacje pozwalają na większą elastyczność naszego modelu, a ponadto mamy pewność, że klasy takie jak “Student” bądź “Wykładowca” posiadają wiedzę o klasie “Kurs”.

## Diagram sekwencji

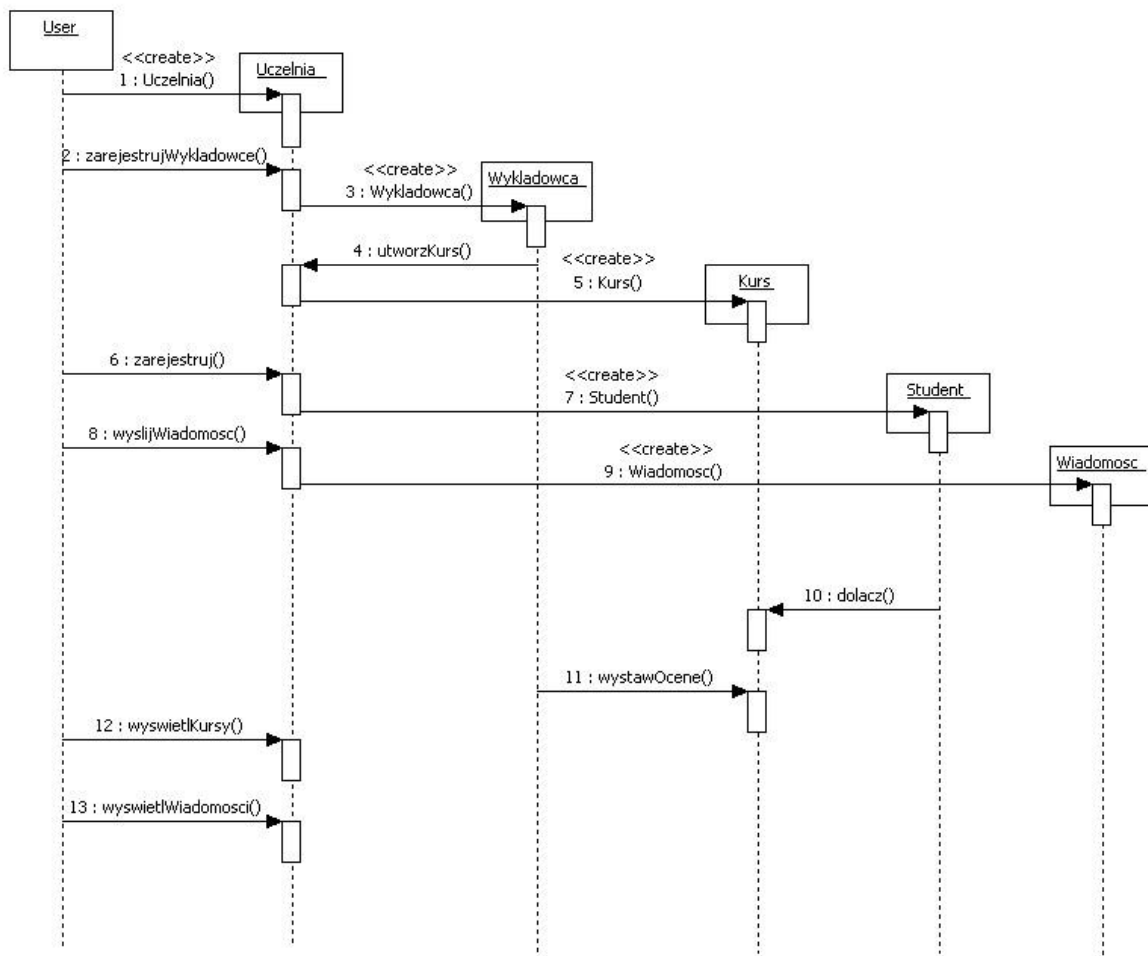


Diagram sekwencji oparty jest na Userze(Administratorze), który tworząc System Obsługi Studenta tworzy instancję klasy Uczelnia, która pozwala mu na sterowanie całym systemem, a także na tworzenie kont wykładowców oraz studentów. Wykładowca tworzy przykładowy kurs, a po zarejestrowaniu studenta administrator wysyła wiadomość powitalną do tego studenta. Student dołącza do kursu, a wykładowca wystawia ocenę na kursie tego studenta. Następnie sprawdza wszystkie kursy oraz wiadomości w systemie.

## Implementacja Systemu Obsługi Studentów

### Uczelnia.h

```
#if !defined(_UCZELNIA_H)
#define _UCZELNIA_H
```

```

#include <string>
#include <vector>
#include <unordered_map>

class Kurs;
class Student;
class Uzytkownik;
class Wiadomosc;
class Wykladowca;

using namespace std;

class Uczelnia {
public:
    Uczelnia(string nazwa);
    Uzytkownik* zaloguj(string login, string haslo);
    bool zarejestruj(string login, string haslo, string nazwisko);
    bool zarejestrujWykladowce(string login, string haslo, string
nazwisko);
    bool wyslijWiadomosc(Uzytkownik* nadawca, string odbiorca, string
temat, string tresc);
    void wyswietlStudentow();
    void wyswietlWykladowcow();
    void wyswietlKursy();
    void wyswietlKursy(Uzytkownik* uzytkownik);
    void wyswietlOceny(Student* student);
    void wyswietlWiadomosci(Uzytkownik* uzytkownik);
    Kurs* utworzKurs(string nazwa, Wykladowca* prowadzacy);
    Kurs* getKurs(int id);
    vector<Kurs*> getKursy();
    vector<Kurs*> getKursy(Uzytkownik* uzytkownik);
    Uzytkownik* getUzytkownik(string login);
    string getNazwa();
    vector<Wiadomosc*> getWiadomosci(Uzytkownik* uzytkownik);

private:
    string nazwa;
    unordered_map<string, Uzytkownik*> uzytkownicy;
    vector<Kurs*> kursy;
    vector<Wiadomosc*> wiadomosci;
};

#endif // _UCZELNIA_H

```

Plik nagłówkowy Uczelnia.h zawiera deklaracje wszystkich pól(atrybutów) i metod(operacji) zawartych w klasie Uczelnia.

## Uczelnia.cpp

```
#include <stdexcept>
#include "Uczelnia.h"
#include "Kurs.h"
#include "Uzytkownik.h"
#include "Wiadomosc.h"
#include "Wykladowca.h"
#include "Student.h"

Uczelnia::Uczelnia(string nazwa) {
    this->nazwa = nazwa;
}

Uzytkownik* Uczelnia::zaloguj(string login, string haslo) {
    try {
        Uzytkownik* u = this->uzytkownicy.at(login);
        if (u->getHaslo() == haslo) {
            return u;
        }
        return nullptr;
    } catch (std::out_of_range const& _) {
        return nullptr;
    }
}

bool Uczelnia::zarejestruj(string login, string haslo, string nazwisko)
{
    if (this->uzytkownicy.find(login) != this->uzytkownicy.end()) {
        // Uzytkownik juz istnieje!
        return false;
    }

    this->uzytkownicy[login] = new Student(login, haslo, nazwisko);
    return true;
}

bool Uczelnia::zarejestrujWykladowce(string login, string haslo, string
nazwisko) {
    if (this->uzytkownicy.find(login) != this->uzytkownicy.end()) {
        // Uzytkownik juz istnieje!
        return false;
    }
    this->uzytkownicy[login] = new Wykladowca(login, haslo, nazwisko);
    return true;
}
```

```

}
bool Uczelnia::wyslijWiadomosc(Uzytkownik* nadawca, string odbiorca,
string temat, string tresc) {
    auto odb = this->getUzytkownik(odbiorca);

    // Jezeli odbiorca nie istnieje
    if (odb == nullptr) {
        return false;
    }

    this->wiadomosci.push_back(new Wiadomosc(nadawca, odb, temat,
tresc));
    return true;
}

void Uczelnia::wyswietlStudentow() {
    printf("Studenci uczelni '%s':\n", this->getNazwa().c_str());

    for (const std::pair<const std::string, Uzytkownik*>& v :
this->uzytkownicy) {
        Student* s = dynamic_cast<Student*>(v.second);
        if (s != nullptr) {
            printf("\t[%s] %s\n", s->getLogin().c_str(),
s->getNazwisko().c_str());
        }
    }
}

void Uczelnia::wyswietlWykladowcow() {
    printf("Wykladowcy uczelni '%s':\n", this->getNazwa().c_str());

    for (const std::pair<const std::string, Uzytkownik*>& v :
this->uzytkownicy) {
        Wykladowca* w = dynamic_cast<Wykladowca*>(v.second);
        if (w != nullptr) {
            printf("\t[%s] %s\n", w->getLogin().c_str(),
w->getNazwisko().c_str());
        }
    }
}

```

```

void Uczelnia::wyswietlKursy() {
    printf("Dostepne kursy: \n");

    for (Kurs* k : this->kursy) {
        printf("\t[%d] %s\n", k->getId(), k->getNazwa().c_str());
    }
}

void Uczelnia::wyswietlKursy(Uzytkownik* uzytkownik) {
    printf("Kursy uzytkownika '%s': \n",
uzytkownik->getNazwisko().c_str());

    for (Kurs* k : this->getKursy(uzytkownik)) {
        printf("\t[%d] %s\n", k->getId(), k->getNazwa().c_str());
    }
}

void Uczelnia::wyswietlOceny(Student* student) {
    printf("Oceny studenta '%s': \n", student->getNazwisko().c_str());

    for (Kurs* k : this->kursy) {
        // Jezeli uzytkownik jest studentem i nalezy do kursu
        if (k->czyNalezy(student)) {
            float ocena = k->getOcena(student);
            if (ocena == 0.0f) {
                printf("\t%s: Brak oceny\n", k->getNazwa().c_str());
            } else {
                printf("\t%s: %.1f\n", k->getNazwa().c_str(), ocena);
            }
        }
    }
}

void Uczelnia::wyswietlWiadomosci(Uzytkownik* uzytkownik) {
    printf("Wiadomosci uzytkownika '%s': \n",
uzytkownik->getNazwisko().c_str());

    for (Wiadomosc* w : this->getWiadomosci(uzytkownik)) {
        printf("\t Od %s: %s\n\t\t%s\n\n",
w->getNadawca().getNazwisko().c_str(), w->getTemat().c_str(),
w->getTresc().c_str());
    }
}

Kurs* Uczelnia::utworzKurs(string nazwa, Wykladowca* prowadzacy) {
    int id = this->kursy.size();

```

```

        auto kurs = new Kurs(id, nazwa, prowadzacy);
        this->kursy.push_back(kurs);
        return kurs;
    }

    Kurs* Uczelnia::getKurs(int id) {
        try {
            return this->kursy.at(id);
        } catch (std::out_of_range const& _) {
            return nullptr;
        }
    }

    vector<Kurs*> Uczelnia::getKursy() {
        return this->kursy;
    }

    vector<Kurs*> Uczelnia::getKursy(Uzytkownik *uzytkownik) {
        vector<Kurs*> kursy;

        for (Kurs* kurs : this->kursy) {
            Student* s = dynamic_cast<Student*>(uzytkownik);

            // Jezeli uzytkownik jest studentem i nalezy do kursu, lub
            // uzytkownik jest prowadzacym kursu
            if ((s != nullptr && kurs->czyNalezy(s)) ||
                kurs->getProwadzacy().getLogin() == uzytkownik->getLogin()) {
                kursy.push_back(kurs);
            }
        }

        return kursy;
    }

    Uzytkownik *Uczelnia::getUzytkownik(string login) {
        try {
            return this->uzytkownicy.at(login);
        } catch (std::out_of_range const& _) {
            return nullptr;
        }
    }

    string Uczelnia::getNazwa() {
        return this->nazwa;
    }

```

```

vector<Wiadomosc*> Uczelnia::getWiadomosci(Uzytkownik *uzytkownik) {
    vector<Wiadomosc*> wiadomosci;

    for (Wiadomosc* w : this->wiadomosci) {
        if (w->getOdbiorca().getLogin() == uzytkownik->getLogin()) {
            wiadomosci.push_back(w);
        }
    }

    return wiadomosci;
}

```

Plik Uczelnia.cpp zawiera pełną implementację operacji zadeklarowanych w pliku nagłówkowym. Klasa ta, będąc “sercem” całego systemu odpowiada za funkcjonowanie i integrację wszystkich elementów struktury logicznej systemu. Posiada ona funkcjonalności pozwalające na tworzenie kont studentów, wykładowców, wyświetlanie i wysyłanie wiadomości, a także pokazywanie dostępnych kursów oraz tworzenie ich (w przypadku wykładowców).

## Kurs.h

```

#ifndef _KURS_H
#define _KURS_H

#include <string>
#include <vector>
#include <unordered_map>

class Student;
class Wykladowca;

using namespace std;

class Kurs {
public:
    Kurs(int id, string nazwa, Wykladowca* prowadzacy);
    bool dolacz(Student* student);
    void wyswietlStudentow();
    void wyswietlWykladowce();
    void wyswietlOceny();
    bool czyNalezy(Student* student);
    void setNazwa(string nazwa);
    bool setOcena(Student* student, float ocena);
    float getOcena(Student* student);
    int getId();

```



```

        string getNazwa();
        Wykladowca& getProwadzacy();
        vector<Student*> getUczestnicy();
private:
        int id;
        string nazwa;
        Wykladowca* prowadzacy;
        vector<Student*> uczestnicy;
        unordered_map<Student*, float> oceny;
};

#endif // _KURS_H

```

Plik Kurs.h posiada deklaracje atrybutów i operacji potrzebnych do prawidłowego działania implementacji kursu.

## Kurs.cpp

```

#include <algorithm>
#include "Kurs.h"
#include "Student.h"
#include "Wykladowca.h"

Kurs::Kurs(int id, string nazwa, Wykladowca* prowadzacy) {
    this->id = id;
    this->nazwa = nazwa;
    this->prowadzacy = prowadzacy;
}

bool Kurs::dolacz(Student* student) {
    // TODO: Sprawdzanie czy rejestracja na kurs jest otwarta, czy
    student jest na odpowiednim roku itd...
    if (this->czyNalezy(student)) {
        return false;
    }

    this->uczestnicy.push_back(student);
    this->oceny[student] = 0.f;
    return true;
}

```

```

void Kurs::wyswietlStudentow() {
    printf("Uczestnicy kursu '%s':\n", this->getNazwa().c_str());

    for (Student* student : this->uczestnicy) {
        printf("\t[%s] %s\n", student->getLogin().c_str(),
student->getNazwisko().c_str());
    }
}

void Kurs::wyswietlWykladowce() {
    printf("Wykladowca kursu '%s':\n", this->getNazwa().c_str());
    printf("\t[%s] %s\n", this->getProwadzacy().getLogin().c_str(),
this->getProwadzacy().getNazwisko().c_str());
}

void Kurs::wyswietlOceny() {
    printf("Oceny uczestnikow kursu '%s':\n", this->getNazwa().c_str());

    for (const std::pair<Student*, float>& v : this->oceny) {
        float ocena = v.second;
        if (ocena == 0.0f) {
            printf("\t[%s] %s: Brak oceny\n",
v.first->getLogin().c_str(), v.first->getNazwisko().c_str());
        } else {
            printf("\t[%s] %s: %.1f\n", v.first->getLogin().c_str(),
v.first->getNazwisko().c_str(), ocena);
        }
    }
}

bool Kurs::czyNalezy(Student* student) {
    return std::find(this->uczestnicy.begin(), this->uczestnicy.end(),
student) != this->uczestnicy.end();
}

void Kurs::setNazwa(string nazwa) {
    // TODO: Sprawdzanie czy użytkownik wysyłający żądanie jest
prowadzącym...
    this->nazwa = nazwa;
}

bool Kurs::setOcena(Student* student, float ocena) {
    // TODO: Sprawdzanie czy użytkownik wysyłający żądanie jest
prowadzącym...
    if (this->czyNalezy(student)) {
        this->oceny[student] = ocena;
        return true;
    }
}

```

```

    } else {
        return false;
    }
}

float Kurs::getOcena(Student* student) {
    if (this->czyNalezy(student)) {
        return this->oceny[student];
    } else {
        return -1.f;
    }
}

int Kurs::getId() {
    return this->id;
}

string Kurs::getNazwa() {
    return this->nazwa;
}

Wykladowca& Kurs::getProwadzacy() {
    return *(this->prowadzacy);
}

vector<Student*> Kurs::getUczestnicy() {
    return this->uczestnicy;
}

```

W pliku Kurs.cpp znajduje się implementacja funkcjonalności, która ma za zadanie dostarczenie podstawowych informacji oraz możliwości obsługi kursu przez Wykładowcę (takich jak wstawianie ocen), jak i również dołączenia studenta do kursu oraz wyświetlania informacji o kursie (oceny, uczestnicy, prowadzącym dany kurs etc.)

## Wiadomosc.h

```

#ifndef _WIADOMOSC_H
#define _WIADOMOSC_H

#include <string>

class Uzytkownik;

using namespace std;

```

```

class Wiadomosc {
public:
    Wiadomosc(Uzytkownik* nadawca, Uzytkownik* odbiorca, string temat,
string tresc);
    Uzytkownik& getNadawca();
    Uzytkownik& getOdbiorca();
    string getTemat();
    string getTresc();
private:
    Uzytkownik* nadawca;
    Uzytkownik* odbiorca;
    string temat;
    string tresc;
};

#endif // _WIADOMOSC_H

```

Powyższy plik zawiera deklarację operacji oraz atrybutów potrzebnych do stworzenia systemu wysyłania wiadomości.

## Wiadomosc.cpp

```

#include "Wiadomosc.h"
#include "Uzytkownik.h"

Wiadomosc::Wiadomosc(Uzytkownik* nadawca, Uzytkownik* odbiorca, string
temat, string tresc) {
    this->nadawca = nadawca;
    this->odbiorca = odbiorca;
    this->temat = temat;
    this->tresc = tresc;
}

Uzytkownik& Wiadomosc::getNadawca() {
    return *(this->nadawca);
}

Uzytkownik& Wiadomosc::getOdbiorca() {
    return *(this->odbiorca);
}

string Wiadomosc::getTemat() {
    return this->temat;
}

string Wiadomosc::getTresc() {
    return this->tresc;
}

```

## Uzytkownik.h

```
#if !defined(_UZYTKOWNIK_H)
#define _UZYTKOWNIK_H

#include <string>

using namespace std;

class Uzytkownik {
public:
    Uzytkownik(string login, string haslo, string nazwisko);
    string getLogin();
    string getHaslo();
    string getNazwisko();
    void setHaslo(string haslo);
    void setNazwisko(string nazwisko);
    virtual ~Uzytkownik() {}
private:
    string login;
    string haslo;
    string nazwisko;
};

#endif // _UZYTKOWNIK_H
```

Plik Uzytkownik.h zawiera deklaracje metod i pól zawartych w klasie.

## Uzytkownik.cpp

```
#include "Uzytkownik.h"

Uzytkownik::Uzytkownik(string login, string haslo, string nazwisko) {
    this->login = login;
    this->haslo = haslo;
    this->nazwisko = nazwisko;
}

string Uzytkownik::getLogin() {
    return this->login;
}

string Uzytkownik::getHaslo() {
    return this->haslo;
}
```

```

string Uzytkownik::getNazwisko() {
    return this->nazwisko;
}

void Uzytkownik::setHaslo(string haslo) {
    this->haslo = haslo;
}

void Uzytkownik::setNazwisko(string nazwisko) {
    this->nazwisko = nazwisko;
}

```

Klasa Użytkownik jest klasą nadrzędną dla klas Student i Wykładowca. Zawiera ona podstawowe operacje i pola, które są wykorzystywane przez klasy podrzędne.

## Student.h

```

#ifndef _STUDENT_H
#define _STUDENT_H

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

using namespace std;

class Student : public Uzytkownik {
public:
    Student(string login, string haslo, string nazwisko);
};

#endif // _STUDENT_H

```

Plik nagłówkowy Student.h zawiera jedynie atrybuty klasy nadrzędnej (Użytkownik).

## Student.cpp

```

#include "Student.h"

Student::Student(string login, string haslo, string nazwisko) :
    Uzytkownik(login, haslo, nazwisko) {}

```

Plik Student.cpp zawiera parametry istotne do zainicjowania użytkownika

## Wykladowca.h

```
#if !defined(_WYKLADOWCA_H)
#define _WYKLADOWCA_H

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

using namespace std;

class Wykladowca : public Uzytkownik {
public:
    Wykladowca(string login, string haslo, string nazwisko);
};

#endif // _WYKLADOWCA_H
```

Podobnie jak w przypadku Studenta, plik ten zawiera deklarację konstruktora klasy Wykladowca z parametrami klasy Uzytkownik.

## Wykladowca.cpp

```
#include "Wykladowca.h"

Wykladowca::Wykladowca(string login, string haslo, string nazwisko) :
Uzytkownik(login, haslo, nazwisko) {}
```

Implementacja klasy Wykladowca, tak samo jak dla Studenta, przyjmuje argumenty niezbędne do stworzenia użytkownika.

## main.cpp

```
#include <iostream>
#include <functional>
#include <string>
#include <random>
#include <map>
#include "Kurs.h"
#include "Uczelnia.h"
#include "Uzytkownik.h"
#include "Wykladowca.h"
#include "Student.h"

void wyswietlKursyZDolaczeniem(Uczelnia* uczelnia, Student* student);
void wyslijWiadomosc(Uczelnia* uczelnia, Uzytkownik* uzytkownik);
```

```

void zarzadzajKursami(Uczelnia* uczelnia, Wykladowca* wykladowca);
void utworzKurs(Uczelnia* uczelnia, Wykladowca* wykladowca);

struct Komenda {
    std::string nazwa;
    std::function<int>(Uczelnia*, Uzytkownik*)> f;
};

std::map<char, Komenda> komendy_student = {
    { 'x', { "Wyloguj sie", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { return -1; } } },
    { '1', { "Wyswietl kursy", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { wyswietlKursyZDolaczeniem(uczelnia,
dynamic_cast<Student*>(uzytkownik)); return 0; } } },
    { '2', { "Wyswietl Twoje kursy", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { uczelnia->wyswietlKursy(uzytkownik); return 0; } } },
    { '3', { "Wyswietl Twoje oceny", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) {
uczelnia->wyswietlOceny(dynamic_cast<Student*>(uzytkownik)); return 0; }
} },
    { '4', { "Wyswietl Twoje wiadomosci", [] (Uczelnia* uczelnia,
Uzytkownik* uzytkownik) { uczelnia->wyswietlWiadomosci(uzytkownik);
return 0; } } },
    { '5', { "Wyswietl wykladowcow", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { uczelnia->wyswietlWykladowcow(); return 0; } } },
    { '6', { "Wyslij wiadomosc", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { wyslijWiadomosc(uczelnia, uzytkownik); return 0; } } }
};

std::map<char, Komenda> komendy_wykladowca = {
    { 'x', { "Wyloguj sie", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { return -1; } } },
    { '1', { "Wyswietl Twoje kursy", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { uczelnia->wyswietlKursy(uzytkownik); return 0; } } },
    { '2', { "Wyswietl Twoje wiadomosci", [] (Uczelnia* uczelnia,
Uzytkownik* uzytkownik) { uczelnia->wyswietlWiadomosci(uzytkownik);
return 0; } } },
    { '3', { "Wyswietl wykladowcow", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { uczelnia->wyswietlWykladowcow(); return 0; } } },
    { '4', { "Zarzadzaj kursami", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { zarzadzajKursami(uczelnia,
dynamic_cast<Wykladowca*>(uzytkownik)); return 0; } } },
    { '5', { "Utworz kurs", [] (Uczelnia* uczelnia, Uzytkownik*
uzytkownik) { utworzKurs(uczelnia,
dynamic_cast<Wykladowca*>(uzytkownik)); return 0; } } },
    { '6', { "Wyslij wiadomosc", [] (Uczelnia* uczelnia, Uzytkownik*

```



```

uzytkownik) { wyslijWiadomosc(uczelnia, uzytkownik); return 0; } } }
};

void clear_screen() {
#ifdef _WIN32
    std::system("cls");
#else
    std::system("clear");
#endif
}

// Wysylanie wiadomosci (wspolne dla wykadowcy i studenta)
void wyslijWiadomosc(Uczelnia* uczelnia, Uzytkownik* uzytkownik) {
    std::cout << "Podaj login adresata: ";
    std::string adresat;
    std::cin >> adresat;

    std::string temat;
    std::string wiadomosc;

    std::cout << "Podaj temat: ";
    std::getline(std::cin >> std::ws, temat);

    std::cout << "Podaj wiadomosc: ";
    std::getline(std::cin >> std::ws, wiadomosc);

    bool result = uczelnia->wyslijWiadomosc(uzytkownik, adresat, temat,
    wiadomosc);

    if (result) {
        std::cout << "Wiadomosc zostala wyslana!\n";
    } else {
        std::cout << "Nie udalo sie wyslac wiadomosc, czy adresat
    istnieje?\n";
    }
}

// Wyszwietlanie wszystkich kursow i pytanie o dolaczenie (dla studenta)
void wyswietlKursyZDolaczeniem(Uczelnia* uczelnia, Student* student) {
    uczelnia->wyswietlKursy();

    std::cout << "Czy chcesz dolaczyc do kursu? [t/N]: ";
    char answer;
    std::cin >> answer;

    if (answer == 'T' || answer == 't') {

```

```

        std::cout << "Podaj id kursu do ktorego chcesz dolaczyc: ";
        int id;
        std::cin >> id;

        auto kurs = uczelnia->getKurs(id);
        if (kurs == nullptr) {
            std::cout << "Kurs nie istnieje!\n";
            return;
        }

        if (kurs->dolacz(student)) {
            std::cout << "Dolaczono do kursu '" << kurs->getNazwa() <<
"'\n";
        } else {
            std::cout << "Nie udalo sie dolaczyc do kursu\n";
        }
    }
}

void wystawOcene(Uczelnia* uczelnia, Wykladowca* wykladowca, Kurs* kurs)
{
    std::string ch;
    do {
        clear_screen();
        std::cout << "Wystawianie ocen w kursie '" << kurs->getNazwa()
<< "'\n";
        std::cout << "-----\n";
        for (auto &student : kurs->getUczestnicy()) {
            std::cout << "[" << student->getLogin() << "]" " <<
student->getNazwisko() << "\n";
        }
        std::cout << "x. Wyjdz\n\n";
        std::cout << "Wybierz studenta: ";
        std::cin >> ch;

        // Znajdz studenta
        for (auto &student : kurs->getUczestnicy()) {
            if (student->getLogin() == ch) {
                float ocena = -1.f;
                do {
                    clear_screen();
                    std::cout << "Wystawianie oceny dla '" <<
student->getNazwisko() << "' w '" << kurs->getNazwa() << "'\n";
                    std::cout << "Podaj ocene <0.0 - 5.0>: ";
                    std::cin >> ch;
                    ocena = std::stof(ch);
                } while (ocena < 0 || ocena > 5);
            }
        }
    } while (ch != 'x');
}

```

```

        } while (ocena < 0.f || ocena > 5.f);

        kurs->setOcena(student, ocena);
        std::cout << "Wystawiono ocene " << ocena << "
studentowi '" << student->getNazwisko() << "' w '" << kurs->getNazwa()
<< "'\n";

        return;
    }
}
} while (ch != "x");
}

void zarzadzajKursem(Uczelnia* uczelnia, Wykladowca* wykladowca, Kurs*
kurs) {
    char ch;
    while (true) {
        clear_screen();
        std::cout << "Zarzadzanie kursem '" << kurs->getNazwa() <<
        "'\n";

        std::cout << "-----\n";
        std::cout << "1. Zmien nazwe\n";
        std::cout << "2. Wyszwietl studentow\n";
        std::cout << "3. Wystaw ocene\n";
        std::cout << "x. Wyjdz\n\n";

        std::cout << "Wybierz opcje: ";
        std::cin >> ch;

        switch (ch) {
            case '1': {
                std::cout << "Podaj nowa nazwe kursu: ";
                std::string nazwa;
                std::getline(std::cin >> std::ws, nazwa);

                std::string staraNazwa = kurs->getNazwa();
                kurs->setNazwa(nazwa);

                std::cout << "Zmieniono nazwe kursu z '" << staraNazwa
<< "' na '" << kurs->getNazwa() << "'!\n";
                break;
            }
            case '2': {
                clear_screen();
                std::cout << "Lista studentow kursu '" <<
kurs->getNazwa() << "'\n";
                for (auto &student : kurs->getUczestnicy()) {

```

```

        char ocena[20];
        if (kurs->getOcena(student) == 0.0f) {
            sprintf(ocena, "Brak oceny");
        } else {
            sprintf(ocena, "%.1f", kurs->getOcena(student));
        }
        std::cout << "[" << student->getLogin() << "]" " <<
student->getNazwisko() << " (ocena " << ocena << ")\n";
    }
    break;
}
case '3':
    wystawOcene(uczelnia, wyklawowca, kurs);
    break;
case 'x':
    return;
}
std::cout << "[ENTER]\n";
std::cin.get(); std::cin.get();
};
}

```

// Wyszwietlanie wlasnych kursow, pytanie o wybor kursu i co chcemy z nim zrobic (dla wyklawowcy)

```

void zarzadzajKursami(Uczelnia* uczelnia, Wyklawowca* wyklawowca) {
    // Wybieranie kursu
    std::string ch;
    while (true) {
        clear_screen();
        std::cout << "Zarzadzanie kursami\n";
        std::cout << "-----\n";
        for (auto &kurs : uczelnia->getKursy(wyklawowca)) {
            std::cout << kurs->getId() << ". " << kurs->getNazwa() <<
"\n";
        }
        std::cout << "x. Wroc\n\n";
        std::cout << "Wybierz kurs: ";
        std::cin >> ch;

        if (ch == "x") {
            return;
        } else {
            auto kurs = uczelnia->getKurs(std::stoi(ch));

            if (kurs != nullptr) {
                zarzadzajKurssem(uczelnia, wyklawowca, kurs);
            }
        }
    }
}

```

```

        return;
    }
}

};

}

// Tworzenie kursu (dla wykładowcy)
void utworzKurs(Uczelnia* uczelnia, Wykladowca* wyklowca) {
    std::cout << "Podaj nazwe kursu: ";
    std::string nazwa;

    std::getline(std::cin >> std::ws, nazwa);

    auto kurs = uczelnia->utworzKurs(nazwa, wyklowca);

    if (kurs != nullptr) {
        std::cout << "Utworzono kurs '" << kurs->getNazwa() << "' o id "
<< kurs->getId() << std::endl;
    } else {
        std::cout << "Nie udalo sie utworzyc kursu!" << std::endl;
    }
}

Uzytkownik* zaloguj(Uczelnia* uczelnia) {
    std::cout << "Przykładowi uzytkownicy:\n\tWykladowca: login
'jkowalski', haslo 'haslo'\n\tStudent: login '1234', haslo 'haslo'\n\n";

    std::string login;
    std::string haslo;
    Uzytkownik* student = nullptr;

    do {
        std::cout << "Podaj login: ";
        std::cin >> login;
        std::cout << "Podaj haslo: ";
        std::cin >> haslo;

        student = uczelnia->zaloguj(login, haslo);
        if (student == nullptr) {
            std::cout << "Nieprawidlowe dane!\n\n";
        }
    } while (student == nullptr);

    std::cout << "Zalogowano pomyslnie!\n\n";
    return student;
}

```

```

void zarejestruj(Uczelnia* uczelnia) {
    std::string haslo;
    std::string nazwisko;
    bool result = false;
    int numer_indeksu;

    do {
        std::cout << "Podaj imie i nazwisko: ";
        std::getline(std::cin >> std::ws, nazwisko);
        std::cout << "Podaj haslo: ";
        std::cin >> haslo;

        // Wygeneruj numer indeksu
        std::random_device dev;
        std::mt19937 rng(dev());
        std::uniform_int_distribution<std::mt19937::result_type>
dist(100000, 999999);

        do {
            numer_indeksu = dist(rng);
        } while (uczelnia->getUzytkownik(std::to_string(numer_indeksu))
!= nullptr);

        result = uczelnia->zarejestruj(std::to_string(numer_indeksu),
haslo, nazwisko);

        if (!result) {
            std::cout << "Nie udalo sie stworzyc uzytkownika!\n";
        }
    } while (result == false);

    std::cout << "\nKonto utworzone pomyslnie, twoj numer indeksu (ktory
jest loginem) to '" << numer_indeksu << "'!\n\n";
}

// Wyszwietla glowne menu po zalogowaniu z dostepnymi opcjami w
zaleznosci od "rangi"
void show_menu(Uczelnia* uczelnia, Uzytkownik* uzytkownik, bool student)
{
    map<char, Komenda> *komendy = student ? &komendy_student :
&komendy_wykladowca;

    char ch;
    int result = 0;
    do {

```

```

        clear_screen();

        std::cout << "System Obslugi Uczelni (zalogowano jako " <<
(student ? "student" : "wykladowca") << " " << uzytkownik->getNazwisko()
<< ")\n";
        std::cout << "-----\n";
        for (auto &v : *komendy) {
            std::cout << v.first << ". " << v.second.nazwa << "\n";
        }
        std::cout << "\nWybierz opcje: ";
        std::cin >> ch;

        if (komendy->find(ch) != komendy->end()) {
// Jezeli wybrana komenda istnieje, wywolujemy ja
            result = komendy->at(ch).f(uczelnia, uzytkownik);
        }

        if (result != -1) {
// Nie czekajmy na enter przy wylogowywaniu (dalej bedziemy czekac w
//show_interface)
            std::cout << "[ENTER]\n";
            std::cin.get(); std::cin.get();
        }
    } while (result == 0);
// Jesli z funkcji dostaniemy inny result code to konczymy menu. Uzywane
// do wylogowywania
}

void show_interface(Uczelnia* uczelnia) {
    // Menu logowania/rejestracji
    char ch;
    while (true) {
        clear_screen();
        std::cout << "System Obslugi Uczelni\n";
        std::cout << "-----\n";
        std::cout << "1. Zaloguj sie\n";
        std::cout << "2. Zarejestruj sie\n";
        std::cout << "x. Wyjdz\n\n";
        std::cout << "Wybierz opcje: ";
        std::cin >> ch;

        switch (ch) {
            case '1': {
                auto uzytkownik = zaloguj(uczelnia);
                bool isStudent = (dynamic_cast<Student*>(uzytkownik) !=
nullptr);

```

```

        show_menu(uczelnia, uzytkownik, isStudent);
        break;
    }
    case '2':
        zarejestruj(uczelnia);
        break;
    case 'x':
        exit(0);
    }

    std::cout << "[ENTER]\n";
    std::cin.get(); std::cin.get();
};
}

int main() {
    auto uczelnia = new Uczelnia("Uniwersytet Chlopskiego Rozumu");
    std::cout << "Utworzono uczelnie '" << uczelnia->getNazwa() <<
    "'\n";

    // Dodajemy testowych uzytkownikow i kurs
    uczelnia->zarejestrujWykladowce("jkowalski", "haslo", "prof. dr hab.
n. med. Jan Kowalski");
    uczelnia->zarejestruj("1234", "haslo", "Piotr Nowak");
    auto w =
dynamic_cast<Wykladowca*>(uczelnia->getUzytkownik("jkowalski"));
    auto s = dynamic_cast<Student*>(uczelnia->getUzytkownik("1234"));
    uczelnia->utworzKurs("Filozofia", w);
    uczelnia->utworzKurs("Ekonomia", w);

    // Wysylamy wiadomosc powitalna
    uczelnia->wyslijWiadomosc(s, s->getLogin(), "Witamy!", "Witamy w
systemie!");

    // Dopisujemy studenta na ekonomie i wystawiamy ocene
    uczelnia->getKurs(1)->dolacz(s);
    uczelnia->getKurs(1)->setOcena(s, 3.0f);

    // Glowna petla programu z interfejsem
    show_interface(uczelnia);

    return 0;
}

```

W pliku main.cpp zauważyć możemy całą implementację interfejsu użytkownika (zarówno studenta, jak i wykładowcy).



Między innymi wyszczególnić możemy takie funkcjonalności, jak:

- `clear_screen()`, która czyści ekran terminala/konsoli w zależności od systemu operacyjnego użytkownika końcowego,
- strukturę `Komenda`, która przechowuje nazwę komendy oraz odpowiadającą jej funkcję
- mapy: `komendy_student` oraz `komendy_wykladowca` która odpowiadają odpowiednio za wyświetlenie dostępnych akcji dla odpowiedniego użytkownika (po uprzednim zalogowaniu)
- metody pomocnicze, które mają za zadanie uniemożliwienie użytkownikowi zarządzanie elementami systemu bezpośrednio (np `wystawOcene`, `zarządzajKursem`, `wyslijWiadomosc`, `utwórzKurs` itp.)
- `show_interface`, która wyświetla interfejs użytkownika służący do logowania/rejestracji użytkownika.
- `show_menu`, która pokazuje menu użytkownika z komendami wcześniej zdefiniowanymi w mapach `komendy_student` oraz `komendy_wykladowca`.

Ponadto w samej funkcji `int main()` zauważyć można stworzenie uczelni, rejestrację przykładowego wykładowcy oraz studenta, tworzenie przykładowych kursów i przypisywanie my wykładowcy, wysyłanie wiadomości do zalogowanego studenta w systemie, dopisywanie mu kursu i wystawianie oceny. Następnie kontrolę nad programem przejmuje użytkownik poprzez wywołanie programu zawierającego interfejs użytkowy użytkownika (funkcję logowania/rejestracji).

## Testy

### Testy jednostkowe

W ramach testów jednostkowych stworzonych zostało 49 przypadków testowych które (o ile to możliwe) testują każdą klasę niezależnie od reszty systemu.

Pliki `"Kurs.test.h"`, `"Wiadomosc.test.h"`, `"Uzytkownik.test.h"` oraz `"Uczelnia.test.h"`

```
#include "common.test.h"
void kurs_run_tests();
```

```
#include "common.test.h"
void wiadomosc_run_tests();
```

```
#include "common.test.h"
void uzytkownik_run_tests();
```

```
#include "common.test.h"
void uczelnia_run_tests();
```

Pliki nagłówkowe testów jednostkowych zawierają jedynie deklarację metod służących do przetestowania klas.

Kurs.test.cpp

Kod:

```
#include "Kurs.test.h"

#include <iostream>

#include "../Student.h"
#include "../Wykladowca.h"
#include "../Kurs.h"

void kurs_run_tests() {
    std::cout << "Tests: Kurs" << std::endl;

    Student* s1 = new Student("student1", "haslo1", "Imie Nazwisko");
    Wykladowca* w1 = new Wykladowca("wykladowca1", "haslo1", "Imie Nazwisko");

    Kurs* kurs = new Kurs(0, "Kurs Testowy", w1);

    assertm("kurs->getId() zwraca id", kurs->getId() == 0);
    assertm("kurs->getNazwa() zwraca nazwe", kurs->getNazwa() == "Kurs Testowy");
    assertm("kurs->getProwadzacy() zwraca prowadzacego",
kurs->getProwadzacy().getLogin() == w1->getLogin());
    kurs->setNazwa("Zmieniona Nazwa");
    assertm("kurs->getNazwa() zwraca zmieniona nazwe", kurs->getNazwa()
== "Zmieniona Nazwa");

    assertm("kurs->getOcena() dla nie dolaczonego studenta zwraca -1.f",
kurs->getOcena(s1) == -1.f);
    assertm("kurs->setOcena() dla nie dolaczonego studenta zwraca false", !kurs->setOcena(s1, 5.f));

    assertm("kurs->getUczestnicy() ma zero uczestnikow",
kurs->getUczestnicy().size() == 0);

    assertm("kurs->czyNalezy() dla nie dolaczonego studenta zwraca false", !kurs->czyNalezy(s1));
```

```

    assertm("kurs->dolacz() dla nie dolaczonego studenta zwraca true",
kurs->dolacz(s1));
    assertm("kurs->czyNalezy() dla dolaczonego studenta zwraca true",
kurs->czyNalezy(s1));
    assertm("kurs->dolacz() dla dolaczonego studenta zwraca false",
!kurs->dolacz(s1));

    assertm("kurs->getUczestnicy() ma jednego uczestnika",
kurs->getUczestnicy().size() == 1);

    assertm("kurs->getOcena() dla dolaczonego studenta bez oceny zwraca
0.f", kurs->getOcena(s1) == 0.f);
    assertm("kurs->setOcena() dla dolaczonego studenta zwraca true",
kurs->setOcena(s1, 5.f));
    assertm("kurs->getOcena() dla dolaczonego studenta zwraca ocene",
kurs->getOcena(s1) == 5.f);

    std::cout << "Kurs ALL OK!" << std::endl << std::endl;
}

```

Test jednostkowy wykonany dla klasy Kurs sprawdza po kolei funkcjonalność wszystkich metod zawartych w klasie.

Wynik testów:

```

Tests: Kurs
kurs->getId() zwraca id :: OK
kurs->getNazwa() zwraca nazwe :: OK
kurs->getProwadzacy() zwraca prowadzacego :: OK
kurs->getNazwa() zwraca zmieniona nazwe :: OK
kurs->getOcena() dla nie dolaczonego studenta zwraca -1.f :: OK
kurs->setOcena() dla nie dolaczonego studenta zwraca false :: OK
kurs->getUczestnicy() ma zero uczestnikow :: OK
kurs->czyNalezy() dla nie dolaczonego studenta zwraca false :: OK
kurs->dolacz() dla nie dolaczonego studenta zwraca true :: OK
kurs->czyNalezy() dla dolaczonego studenta zwraca true :: OK
kurs->dolacz() dla dolaczonego studenta zwraca false :: OK
kurs->getUczestnicy() ma jednego uczestnika :: OK
kurs->getOcena() dla dolaczonego studenta bez oceny zwraca 0.f :: OK
kurs->setOcena() dla dolaczonego studenta zwraca true :: OK
kurs->getOcena() dla dolaczonego studenta zwraca ocene :: OK
Kurs ALL OK!

```

Jak możemy zauważyć, poszczególne testowane przez nas funkcjonalności podały odpowiedni wynik zarówno w przypadku poprawnych, jak i niepoprawnych danych.

## Wiadomosc.test.cpp

Kod:

```
#include "Wiadomosc.test.h"

#include <iostream>

#include "../Student.h"
#include "../Wiadomosc.h"

void wiadomosc_run_tests() {
    std::cout << "Tests: Wiadomosc" << std::endl;
    Student* s1 = new Student("student1", "haslo1", "Imie Nazwisko");
    Student* s2 = new Student("student2", "haslo2", "Imie Nazwisko");

    Wiadomosc* wiadomosc = new Wiadomosc(s1, s2, "Temat", "Lorem
Ipsum");

    assertm("wiadomosc->getNadawca() zwraca nadawce",
wiadomosc->getNadawca().getLogin() == s1->getLogin());
    assertm("wiadomosc->getOdbiorca() zwraca odbiorce",
wiadomosc->getOdbiorca().getLogin() == s2->getLogin());
    assertm("wiadomosc->getTemat() zwraca temat", wiadomosc->getTemat()
== "Temat");
    assertm("wiadomosc->getTresc() zwraca tresc", wiadomosc->getTresc()
== "Lorem Ipsum");

    std::cout << "Wiadomosc ALL OK!" << std::endl << std::endl;
}
```

Proste testy sprawdzające funkcjonalność funkcji wiadomości w systemie

Wyniki testu:

```
Tests: Wiadomosc
wiadomosc->getNadawca() zwraca nadawce :: OK
wiadomosc->getOdbiorca() zwraca odbiorce :: OK
wiadomosc->getTemat() zwraca temat :: OK
wiadomosc->getTresc() zwraca tresc :: OK
Wiadomosc ALL OK!
```

Przeprowadzone testy nie wykazały błędów w funkcjonalności tego modułu.

## Uzytkownik.test.cpp

Kod:

```
#include "Uzytkownik.test.h"

#include <iostream>

#include "../Uzytkownik.h"
#include "../Student.h"
#include "../Wykladowca.h"

void uzytkownik_run_tests() {
    std::cout << "Tests: Uzytkownik/Student/Wykladowca" << std::endl;
    Student* s1 = new Student("student1", "haslo", "Imie Nazwisko");
    Wykladowca* w1 = new Wykladowca("wykladowca1", "haslo", "Imie
Nazwisko");
    Uzytkownik* u_student = s1;
    Uzytkownik* u_wykladowca = w1;

    assertm("uzytkownik->getLogin() zwraca login", s1->getLogin() ==
"student1");
    assertm("uzytkownik->getHaslo() zwraca haslo", s1->getHaslo() ==
"haslo");
    assertm("uzytkownik->getNazwisko() zwraca nazwisko",
s1->getNazwisko() == "Imie Nazwisko");

    s1->setHaslo("nowehaslo");
    s1->setNazwisko("Nowe Nazwisko");
    assertm("uzytkownik->getHaslo() po zmianie hasla", s1->getHaslo() ==
"nowehaslo");
    assertm("uzytkownik->getNazwisko() po zmianie nazwiska",
s1->getNazwisko() == "Nowe Nazwisko");

    assertm("dynamic_cast<Student> dla Uzytkownika studenta",
dynamic_cast<Student*>(u_student) != nullptr);
    assertm("dynamic_cast<Wykladowca> dla Uzytkownika wyklowcy",
dynamic_cast<Wykladowca*>(u_wykladowca) != nullptr);

    assertm("dynamic_cast<Student> dla Uzytkownika wyklowcy",
dynamic_cast<Student*>(u_wykladowca) == nullptr);
    assertm("dynamic_cast<Wykladowca> dla Uzytkownika studenta",
dynamic_cast<Wykladowca*>(u_student) == nullptr);

    std::cout << "Uzytkownik/Student/Wykladowca ALL OK!" << std::endl <<
std::endl;
}
```

Wyniki testu:

```
Tests: Uzytkownik/Student/Wykladowca
uzytkownik->getLogin() zwraca login :: OK
uzytkownik->getHaslo() zwraca haslo :: OK
uzytkownik->getNazwisko() zwraca nazwisko :: OK
uzytkownik->getHaslo() po zmianie hasla :: OK
uzytkownik->getNazwisko() po zmianie nazwiska :: OK
dynamic_cast<Student> dla Uzytkownika studenta :: OK
dynamic_cast<Wykladowca> dla Uzytkownika wyklawowcy :: OK
dynamic_cast<Student> dla Uzytkownika wyklawowcy :: OK
dynamic_cast<Wykladowca> dla Uzytkownika studenta :: OK
Uzytkownik/Student/Wykladowca ALL OK!
```

Plik testowy sprawdza poprawność funkcjonowania metod zarówno dla klasy Użytkownik, Student jak i Wykładowca, przy czym dla ostatniej dwójki sprawdza poprawność rzutowania dynamicznego w stosunku do klas podrzędnych.

Uczelnia.test.cpp

Kod:

```
#include "Uczelnia.test.h"

#include <iostream>

#include "../Student.h"
#include "../Wykladowca.h"
#include "../Kurs.h"
#include "../Uczelnia.h"

void uczelnia_run_tests() {
    std::cout << "Tests: Uczelnia" << std::endl;

    Uczelnia* uczelnia = new Uczelnia("Uczelnia");

    assertm("uczelnia->getNazwa() zwraca nazwe", uczelnia->getNazwa() ==
"Uczelnia");

    assertm("uczelnia->zarejestruj() zwraca true dla poprawnych danych",
uczelnia->zarejestruj("student1", "haslo", "Imie Nazwisko"));
    assertm("uczelnia->zarejestruj() zwraca false dla niepoprawnych
danych", !uczelnia->zarejestruj("student1", "haslo", "Imie Nazwisko"));
    assertm("uczelnia->zarejestrujWykladowce() zwraca true dla
poprawnych danych", uczelnia->zarejestrujWykladowce("wykladowca1",
```

```
"haslo", "Imie Nazwisko"));
    assertm("uczelnia->zarejestrujWykladowce() zwraca false dla
niepoprawnych danych", !uczelnia->zarejestrujWykladowce("wykladowca1",
"haslo", "Imie Nazwisko"));

    assertm("uczelnia->getUzytkownik() zwraca uzytkownika",
uczelnia->getUzytkownik("wykladowca1")->getLogin() == "wykladowca1");
    assertm("uczelnia->getUzytkownik() zwraca nullptr",
uczelnia->getUzytkownik("nieistniejacy") == nullptr);

    assertm("uczelnia->zaloguj() zwraca nullptr dla zlych danych",
uczelnia->zaloguj("wykladowca1", "zlehaslo") == nullptr);
    auto wyklowca =
dynamic_cast<Wykladowca*>(uczelnia->zaloguj("wykladowca1", "haslo"));
    assertm("uczelnia->zaloguj() zwraca uzytkownika dla poprawnych
danych", wyklowca->getLogin() == "wykladowca1");

    auto student = dynamic_cast<Student*>(uczelnia->zaloguj("student1",
"haslo"));

    assertm("uczelnia->getWiadomosci(uzytkownik) ma 0 wiadomosci",
uczelnia->getWiadomosci(student).size() == 0);
    assertm("uczelnia->wyslijWiadomosc() zwraca false dla
nieistniejacego uzytkownika", !uczelnia->wyslijWiadomosc(wyklowca,
"nieistniejacy", "Temat", "Tresc"));
    assertm("uczelnia->wyslijWiadomosc() zwraca true dla istniejacego
uzytkownika", uczelnia->wyslijWiadomosc(wyklowca, "student1", "Temat",
"Tresc"));
    assertm("uczelnia->getWiadomosci(uzytkownik) ma 1 wiadomosc",
uczelnia->getWiadomosci(student).size() == 1);

    assertm("uczelnia->getKursy() ma 0 kursow",
uczelnia->getKursy().size() == 0);

    auto kurs0 = uczelnia->utworzKurs("Testowy Kurs", wyklowca);
    auto kurs1 = uczelnia->utworzKurs("Testowy Kurs 2", wyklowca);

    assertm("uczelnia->getKursy() ma 2 kursy",
uczelnia->getKursy().size() == 2);

    assertm("uczelnia->getKursy(student) ma 0 kursow",
uczelnia->getKursy(student).size() == 0);
    uczelnia->getKurs(0)->dolacz(student);
    assertm("uczelnia->getKursy(student) ma 1 kurs",
uczelnia->getKursy(student).size() == 1);
```

```

    assertm("uczelnia->utworzKurs() zwraca utworzony kurs",
kurs0->getNazwa() == "Testowy Kurs" && kurs0->getProwadzacy().getLogin()
== wyklawowca->getLogin());
    assertm("Pierwszy utworzony kurs ma id 0", kurs0->getId() == 0);
    assertm("Drugi utworzony kurs ma id 1", kurs1->getId() == 1);

    assertm("uczelnia->getKurs() zwraca kurs",
uczelnia->getKurs(1)->getNazwa() == kurs1->getNazwa());

    std::cout << "Uczelnia ALL OK!" << std::endl << std::endl;
}

```

*Znaleziony błąd podczas testowania Uczelnia::getUzytkownik()*

W trakcie testów jednostkowych dla modułu uczelni metoda getUzytkownik() dla nieistniejącego użytkownika wyrzucała wyjątek zamiast zwracać nullptr, co nie spełniało założeń testu.

Po dodaniu klauzuli try-catch do metody getUzytkownik funkcja zaczęła spełniać założenia testów:

Wyniki testu (przed naprawą):

```

Tests: Uczelnia
uczelnia->getNazwa() zwraca nazwe :: OK
uczelnia->zarejestruj() zwraca true dla poprawnych danych :: OK
uczelnia->zarejestruj() zwraca false dla niepoprawnych danych :: OK
uczelnia->zarejestrujWykladowce() zwraca true dla poprawnych danych :: OK
uczelnia->zarejestrujWykladowce() zwraca false dla niepoprawnych danych :: OK
uczelnia->getUzytkownik() zwraca uzytkownika :: OK
terminate called after throwing an instance of 'std::out_of_range'
  what():  unordered_map::at
./test.sh: linia 2: 11444 Przerwane          (zrzut pamięci) tests/uczelnia_test

```



Wyniki testu (po dodaniu klauzuli try-catch):

```
Tests: Uczelnia
uczelnia->getNazwa() zwraca nazwe :: OK
uczelnia->zarejestruj() zwraca true dla poprawnych danych :: OK
uczelnia->zarejestruj() zwraca false dla niepoprawnych danych :: OK
uczelnia->zarejestrujWykladowce() zwraca true dla poprawnych danych :: OK
uczelnia->zarejestrujWykladowce() zwraca false dla niepoprawnych danych :: OK
uczelnia->getUzytkownik() zwraca uzytkownika :: OK
uczelnia->getUzytkownik() zwraca nullptr :: OK
uczelnia->zaloguj() zwraca nullptr dla zlych danych :: OK
uczelnia->zaloguj() zwraca uzytkownika dla poprawnych danych :: OK
uczelnia->getWiadomosci(uzytkownik) ma 0 wiadomosci :: OK
uczelnia->wyslijWiadomosc() zwraca false dla nieistniejacego uzytkownika :: OK
uczelnia->wyslijWiadomosc() zwraca true dla istniejacego uzytkownika :: OK
uczelnia->getWiadomosci(uzytkownik) ma 1 wiadomosc :: OK
uczelnia->getKursy() ma 0 kursow :: OK
uczelnia->getKursy() ma 2 kursy :: OK
uczelnia->getKursy(student) ma 0 kursow :: OK
uczelnia->getKursy(student) ma 1 kurs :: OK
uczelnia->utworzKurs() zwraca utworzony kurs :: OK
Pierwszy utworzony kurs ma id 0 :: OK
Drugi utworzony kurs ma id 1 :: OK
uczelnia->getKurs() zwraca kurs :: OK
Uczelnia ALL OK!
```

Jak można zauważyć, po dodaniu rzeczony klauzuli wszystkie pozostałe testy nie zwracały błędów, przez co możemy zauważyć komunikat informujący nas o pomyślnym przejściu wszystkich testów klasy Uczelnia

main.test.cpp

Kod:

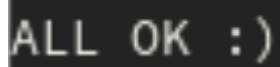
```
#include <iostream>

#include "Wiadomosc.test.h"
#include "Uzytkownik.test.h"
#include "Kurs.test.h"
#include "Uczelnia.test.h"

int main() {
    uzytkownik_run_tests();
    wiadomosc_run_tests();
    kurs_run_tests();
    uczelnia_run_tests();

    std::cout << "ALL OK :)" << std::endl;
}
```

Wyniki wszystkich testów:



Plik `main.test.cpp` służył jako główny plik do wywoływania funkcji testujących poszczególne klasy. W przypadku przejścia testów, program zwraca komunikat zamieszczony wyżej. Jednak w przypadku braku pomyślnego przejścia testu, program wyrzuca wyjątek (przykład: błąd podczas testowania klasy `Uczelnia`). W ten sposób nasz projekt przeszedł testy jednostkowe.

## Testy akceptacyjne

### Scenariusz testu akceptacyjnego

1. Zaloguj się jako wykładowca
2. Utwórz co najmniej 10 kursów
3. Zmień nazwę co najmniej jednego kursu
4. Wyświetl swoje kursy
5. Wyloguj się
6. Zarejestruj się jako nowy student
7. Zaloguj się jako nowy student
8. Wyświetl kursy
9. Dołącz do kursu
10. Wyświetl swoje kursy
11. Wyświetl wykładowców
12. Wyślij wiadomość do wykładowcy
13. Wyloguj się
14. Zaloguj się jako wykładowca
15. Wystaw ocenę dla nowego studenta w kursie
16. Wyświetl swoje wiadomości
17. Wyloguj się
18. Zaloguj się jako nowy student
19. Wyświetl swoje oceny
20. Wyloguj się
21. Wyłącz aplikację

## Znaleziony błąd w testach systemowych/akceptacyjnych

Początkowo w menu zarządzania testami wybrane id testu wczytywaliśmy do char. W trakcie testów systemowych/akceptacyjnych zauważyliśmy brak możliwości wyboru kursu w przypadku więcej niż 10 kursów.

```
Zarządzanie kursami
```

```
-----
```

```
0. Filozofia  
1. Ekonomia  
2. Kurs Testowy  
3. Kurs Testowy  
4. Kurs Testowy  
5. Kurs Testowy  
6. Kurs Testowy  
7. Kurs Testowy  
8. Kurs Testowy  
9. Kurs Testowy  
10. Kurs Testowy  
11. Kurs Testowy  
x. Wroc
```

```
Wybierz kurs: 10
```

```
Zarządzanie kursem 'Ekonomia'
```

```
-----
```

```
1. Zmien nazwe  
2. Wystaw ocene  
x. Wyjdz
```

W celu naprawienia błędu zmieniliśmy wczytywanie wybranej opcji do string.

# System Obsługi Studenta - interfejs użytkownika (poszczególne elementy)

Aby zadbać o wygodne korzystanie z aplikacji zaimplementowany został prosty interfejs użytkownika wyświetlany w terminalu/konsoli.

## Ekran główny (logowanie/rejestracja)

```
System Obsługi Uczelni
-----
1. Zaloguj sie
2. Zarejestruj sie
x. Wyjdz

Wybierz opcje: 1
Przykładowi uzytkownicy:
    Wykladowca: login 'jkowalski', haslo 'haslo'
    Student: login '1234', haslo 'haslo'

Podaj login: jkowalski
Podaj haslo: haslo
```

**(Ważna uwaga:** przykładowy wykładowca/student został pokazany na potrzeby prezentacji systemu i zaprezentowania systemu logowania!)

Ekran główny Systemu Obsługi Uczelni posiada trzy opcje: logowanie, rejestrację oraz wyjście z aplikacji. W przykładzie wyżej możemy zaobserwować operację logowania, która uruchamiana jest po wpisaniu przez użytkownika odpowiedniej opcji.

## Panel zarządzania (Wykładowca)

```
System Obsługi Uczelni (zalogowano jako wykładowca prof. dr hab. n. med. Jan Kowalski)
-----
1. Wyświetl Twoje kursy
2. Wyświetl Twoje wiadomosci
3. Wyświetl wykładowcow
4. Zarządzaj kursami
5. Utworz kurs
6. Wysluj wiadomosc
x. Wyloguj sie

Wybierz opcje: 5
```

Panel zarządzania wykładowcy składa się z 7 opcji:

- Wyświetlanie kursów, którymi się zarządza
- Wyświetlanie wiadomości, które użytkownik otrzymał
- Wyświetlenie innych wykładowców
- Zarządzanie istniejącymi kursami
- Utworzenie kursu

- Wysłanie wiadomości
- Wylogowanie

Interfejs użytkownika pozwala na sprawne zarządzanie systemem przez wykładowcę.

Aby pokazać dalsze działanie systemu, wybraliśmy opcję stworzenia kursu

## Stwórz kurs (Wykładowca)

```
System Obsługi Uczelni (zalogowano jako wykładowca prof. dr hab. n. med. Jan Kowalski)
-----
1. Wyświetl Twoje kursy
2. Wyświetl Twoje wiadomości
3. Wyświetl wykładowców
4. Zarządzaj kursami
5. Utwórz kurs
6. Wyślij wiadomość
x. Wyloguj się

Wybierz opcję: 5
Podaj nazwę kursu: Kurs Testowy
Utworzono kurs 'Kurs Testowy' o id 2
[ENTER]
```

W powyższym przypadku wykładowca tworzy kurs o nazwie “Kurs Testowy”. Po wpisaniu nazwy automatycznie przypisywane jest mu id, a informacje o nim zostaną przechowane w systemie.

## Rejestracja studenta

```
System Obsługi Uczelni
-----
1. Zaloguj się
2. Zarejestruj się
x. Wyjdź

Wybierz opcję: 2
Podaj imię i nazwisko: Student Testowy
Podaj hasło: hasło

Konto utworzone pomyślnie, twój numer indeksu (który jest loginem) to '151420'!
[ENTER]
```

W przypadku rejestracji studenta, użytkownik zostanie poproszony o wpisanie imienia, nazwiska oraz hasła. Następnie zapisywany jest on w systemie, przy czym jednocześnie zostaje przypisany mu losowy numer albumu (który jednocześnie służy do logowania).

## Logowanie wcześniej zarejestrowanego studenta

```
System Obsługi Uczelni
-----
1. Zaloguj sie
2. Zarejestruj sie
x. Wyjdz

Wybierz opcje: 1
Przykładowi uzytkownicy:
    Wykladowca: login 'jkowalski', haslo 'haslo'
    Student: login '1234', haslo 'haslo'

Podaj login: 151420
Podaj haslo: haslo
```

## Panel Zarządzania (Student)

```
System Obsługi Uczelni (zalogowano jako student Student Testowy)
-----
1. Wyświetl kursy
2. Wyświetl Twoje kursy
3. Wyświetl Twoje oceny
4. Wyświetl Twoje wiadomosci
5. Wyświetl wykładowcow
6. Wysluj wiadomosc
x. Wyloguj sie

Wybierz opcje: 6
Podaj login adresata: jkowalski
Podaj temat: hello world
Podaj wiadomosc: Hello world!
Wiadomosc zostala wyslana!
[ENTER]
```

Jak można zauważyć, logowanie się wcześniej zarejestrowanego studenta powiodło się i ma on dostęp do panelu zarządzania. Jednakże student ma ograniczone możliwości jeśli chodzi o możliwości zarządzania. Należą do nich:

- Wyświetlanie wszystkich kursów
- Wyświetlanie kursów, na które uczęszcza student
- Wyświetlanie ocen z kursów
- Wyświetlanie wiadomości
- Wyświetlanie wykładowców
- Wysłanie wiadomości
- Wylogowanie

W przykładzie podanym na obrazku widzimy działającą funkcjonalność wysyłania wiadomości (w tym przypadku do wykładowcy).

## Wyświetlanie kursów (z możliwością dołączenia)

```
System Obsługi Uczelni (zalogowano jako student Student Testowy)
-----
1. Wyświetl kursy
2. Wyświetl Twoje kursy
3. Wyświetl Twoje oceny
4. Wyświetl Twoje wiadomości
5. Wyświetl wykładowców
6. Wyślij wiadomość
x. Wyloguj się

Wybierz opcję: 1
Dostępne kursy:
    [0] Filozofia
    [1] Ekonomia
    [2] Kurs Testowy
Czy chcesz dołączyć do kursu? [t/N]: t
Podaj id kursu do którego chcesz dołączyć: 2
Dołączono do kursu 'Kurs Testowy'!
[ENTER]
```

System wyświetlania kursów pozwala na szybkie wylistowanie dostępnych kursów, a także daje studentowi możliwość dołączenia do danego kursu.

## Panel zarządzania kursami (wykładowca)

```
Zarządzanie kursami
-----
0. Filozofia
1. Ekonomia
2. Kurs Testowy
x. Wroc

Wybierz kurs: 2

Zarządzanie kursem 'Kurs Testowy'
-----
1. Zmien nazwe
2. Wyświetl studentów
3. Wystaw ocene
x. Wyjdz

Wybierz opcję: 3
```

```
Wystawianie ocen w kursie 'Kurs Testowy'
```

```
-----
```

```
[151420] Student Testowy
```

```
x. Wyjdz
```

```
Wybierz studenta: 151420
```

```
Wystawianie oceny dla 'Student Testowy' w 'Kurs Testowy'
```

```
Podaj ocene <0.0 - 5.0>: 5.0
```

```
Wystawiono ocene 5 studentowi 'Student Testowy' w 'Kurs Testowy'
```

```
[ENTER]
```

Dzięki tej funkcjonalności wykładowca może w pełni zarządzać należącym do niego kursem. W przypadku powyżej widzimy przykładowe wystawienie oceny dla “Studenta Testowego”, który zapisany jest na “Kurs Testowy”.

## Wyświetlanie wiadomości

```
System Obsługi Uczelni (zalogowano jako wykładowca prof. dr hab. n. med. Jan Kowalski)
```

```
-----
```

```
1. Wyświetl Twoje kursy
```

```
2. Wyświetl Twoje wiadomości
```

```
3. Wyświetl wykładowców
```

```
4. Zarządzaj kursami
```

```
5. Utwórz kurs
```

```
6. Wysłij wiadomość
```

```
x. Wyloguj się
```

```
Wybierz opcję: 2
```

```
Wiadomości użytkownika 'prof. dr hab. n. med. Jan Kowalski':
```

```
    Od Student Testowy: hello world
```

```
        Hello world!
```

```
[ENTER]
```

Opcja ta wyświetla wszystkie napisane wiadomości. W tym przypadku mamy wiadomość wcześniej napisaną do wykładowcy o tytule “Hello world!”.



## Wyświetlanie kursów studenta, na które jest zapisany

```
System Obsługi Uczelni (zalogowano jako student Student Testowy)
-----
1. Wyświetl kursy
2. Wyświetl Twoje kursy
3. Wyświetl Twoje oceny
4. Wyświetl Twoje wiadomości
5. Wyświetl wykładowców
6. Wyślij wiadomość
x. Wyloguj się

Wybierz opcję: 2
Kursy użytkownika 'Student Testowy':
      [2] Kurs Testowy
[ENTER]
```

Przykład funkcjonalności wyświetlania kursów, na które zapisany został student.