UNIWERSYTET WSB MERITO POZNAŃ

Dokumentacja projektowa systemu informatycznego do zarzadzania dziekanatem

Autor:

Sebastian Mazurek, grupa: K32

Spis treści

1. Ogólne informacje o projekcie

System Zarządzania Dziekanatem to aplikacja internetowa wykonana w architekturze MVC oparta na platformie .NET 8, stworzona z myślą o wsparciu podstawowych czynności administracyjnych dziekanatu. System umożliwia rozliczanie wyników w nauce studentów w formie punkcji ECTS. Punkty ECTS (European Credit Transfer and Accumulation System) to system punktowy stosowany w europejskim szkolnictwie wyższym, który ułatwia porównywanie osiągnięć akademickich i przenoszenie ich między różnymi uczelniami oraz krajami. Został wprowadzony w ramach procesu bolońskiego, aby umożliwić większą mobilność studentów i lepsze uznawanie kwalifikacji w Europie. Aktorzy sytemu posiadają róne zestawy funkcjonalności w zależności od przypisanej roli. Najbardziej rozbudowane fukjonalności posiada dziekan, gdzie najciekawszą funkcjonalnością jest możliwość tworzenia rankingów najlepszych studentów, oraz sporządzanie rankingu obciążenia dydaktycznego wykładowców. Opracowanie stanowi dobrą bazę wyjściową do rozbudowy systemu i tworzenia następnych funkcjonalności zgodnych z oczekiwaniami zamawiającego.

2. **Użyte technologie**

a) **Backend:** .NET 8 (ASP.NET MVC)

b) **Baza danych:** SQL Server (skonfigurowany za pomocą Entity Framework Core w podejściu Code-First).

c) **Uwierzytelnianie:** Uwierzytelnianie oparte na ciasteczkach z ASP.NET Identity i haszowaniem haseł

d) Frontend: Strony Razor oraz Bootstrap.

3. Uruchomienie lokalne projektu

Aplikację można uruchomić w środowisku .NET 8 SDK. Mając prawidłowo zainstalowane w/w Środowisko, pobieramy pliki z publicznego repozytorium projektu z GitHub:

Repozytorium: https://github.com/mazureks/DeansOfficeManagement.git

Kopiujemy pliki repozytorium na lokalny dysk. Po skopiowaniu plików otwieramy rozwiązanie **DeansOfficeManagement.sln.**

W pliku **appsettings.json** znajduje się konfiguracja połączenia do bazy danych, którą w razie potrzeby można edytować.

Przed pierwszym uruchomieniem projektu należy wykonać aktualizację bazy danych:

Przejdź do

Narzędzia > **Menedżer pakietów NuGet** > **Konsola Menedżera pakietów**

Uruchom poniższą komendę, aby utworzyć bazę danych i zastosować migracje:

update-database

Uruchom projekt:

Naciśnij F5 lub wybierz Uruchom, aby rozpocząć projekt na lokalnym serwerze.

Uwaga: Możliwe, że przy uruchamianiu projektu zajdzie konieczność wykonania zmian w menu wyboru profilu uruchamiania w Visual Studio pomiędzy IIS a https

Rejestracja i logowanie:

Użytkownicy tworzą swoje konta przez rejestrację, **ale tylko pierwszy** wpisany użytkownik ma przupisaną rolę Administratora (dziekana). Następni uzytkownicy nie mają przypisanej domyślnej roli. Role użytkownikom, poprzez odpowiedni panel przypisuje dziekan (patrz: rozdział 8).

4. Struktura projektu

Projekt wykonany został w wrchitekturze **Model Views Controler** z tąd wynika struktura katalogów.

• **Controllers** - zawiera logikę aplikacji odpowiedzialną za obsługę żądań HTTP. Każdy kontroler odpowiada za konkretne zasoby (np. kursy, oceny, studenci, statystyki) i jest odpowiedzialny za przekazywanie danych do widoków oraz obsługę akcji (np. dodawanie, edytowanie).

• **Models** - definicje modeli danych, które reprezentują strukturę przechowywanych informacji w aplikacji. Modele są wykorzystywane do mapowania danych na bazę danych oraz do przekazywania danych między warstwami aplikacji.

• **Views** - Widoki Razor Pages używane do prezentacji danych użytkownikowi. Są to pliki HTML z kodem C#, które dynamicznie renderują dane na stronie.

• **wwwroot** - katalog, w którym znajdują się statyczne zasoby aplikacji. Jest to publiczny katalog, z którego przeglądarki mogą pobierać pliki. Wszystkie zasoby w tym folderze są udostępniane publicznie, czyli dostępne bezpośrednio poprzez URL.

Dodatkowo w projekcie znajduje się folder **DATA** zawierający dwa pliki:

a) **ApplicationDbContext.cs**

- Definiuje modele danych i relacje między nimi.
- Odpowiada za konfigurację bazy danych i jej inicjalizację.
- Integruje funkcjonalności Identity z modelami aplikacji, co pozwala na łatwe zarządzanie użytkownikami.

b) ModelBuilderExtensions.cs

Plik definiuje dane początkowe dla bazy danych projektu ASP.NET Core (roles, users, courses, grades, registrations, requests). Obecnie dane są zakomentowane więc nie są tworzone w trakcie działania programu. Funkcjonalność była przydatna na etapie prac deweloperskich.

5. Modele użyte w projekcie

W projekcie wykorzystano następujące modele domenowe:

a) ApplicationUser

Model ApplicationUser rozszerza IdentityUser i reprezentuje użytkowników systemu, takich jak studenci, wykładowcy czy pracownicy dziekanatu. Dodaje specyficzne właściwości, które pozwalają na zarządzanie rolami i danymi użytkowników.

- **Pola w modelu:**
- **FirstName**
  - Typ: string?
  - Walidacja: Maksymalna długość: 50 znaków.
- **LastName**
  - Typ: string?
  - Walidacja: Maksymalna długość: 50 znaków.
- **Role**
  - Typ: string?
- **StudentNumber**
  - Typ: string?
  - Walidacja: Maksymalna długość: 20 znaków.
- **CourseRegistrations**
  - Typ: ICollection<CourseRegistration>?
- **Requests**
  - Typ: ICollection<Request>?

b) AdminPanel

Model AdminPanel reprezentuje zadania administratora w systemie. Służy do zarządzania i przechowywania informacji o zadaniach oraz terminach ich realizacji.

- **AdminPanelId**
  - Typ: int
  - Walidacja: Klucz główny (automatycznie generowany).
- **TaskName**
  - Typ: string?
- **Deadline**
  - Typ: DateTime

c) Course

Model Course reprezentuje kurs w systemie. Przechowuje informacje o nazwie kursu, liczbie punktów ECTS oraz powiązaniach z wykładowcą, rejestracjami studentów i ocenami.

Pola w modelu:

- **CourseId**
  - Typ: int
  - Walidacja: Klucz główny (automatycznie generowany).
- **CourseName**
  - Typ: string?
  - Walidacja: Pole wymagane, maksymalna długość: 100 znaków.
- **Credits**
  - Typ: int
- **LecturerId**
  - Typ: string?
- **Lecturer**
  - Typ: ApplicationUser?
  - Walidacja: Klucz obcy wskazujący na użytkownika pełniącego rolę wykładowcy.

d) CourseRegistration

Model CourseRegistration reprezentuje rejestrację użytkownika (studenta) na kurs w systemie. Zarządza informacjami o statusie rejestracji oraz powiązaniami z odpowiednimi kursami i studentami.

Pola w modelu:

- **CourseRegistrationId**
  - Typ: int
  - Walidacja: Klucz główny (automatycznie generowany).
- **Status**
  - Typ: string?
- **StudentId**
  - Typ: string?
  - Walidacja: Klucz obcy wskazujący na użytkownika (studenta).
- **Student**
  - Typ: ApplicationUser?
  - Walidacja: Odniesienie do studenta zarejestrowanego na kurs.
- **CourseId**

- Typ: int
- Walidacja: Klucz obcy wskazujący na kurs.

- **Course**
  - Typ: Course?
  - Walidacja: Odniesienie do kursu, na który zarejestrowano studenta.

e) Grade

Pola w modelu:

- **GradeId**
  - Typ: int
  - Walidacja: Klucz główny (automatycznie generowany).
- **Score**
  - Typ: int
  - Walidacja: Zakres wartości od 0 do 100 ([Range(0, 100)]). Reprezentuje wynik procentowy.
- **GradeLetter**
  - Typ: string?
  - Walidacja: Brak. Opcjonalne pole przechowujące literowy zapis oceny, np. "A", "B".
- **StudentId**
  - Typ: string
  - Walidacja: Klucz obcy wskazujący na użytkownika (studenta).
- **Student**
  - Typ: ApplicationUser
  - Walidacja: Odniesienie do studenta, któremu przypisano ocenę.
- **CourseId**
  - Typ: int
  - Walidacja: Klucz obcy wskazujący na kurs, do którego przypisano ocenę.
- **Course**
  - Typ: Course
  - Walidacja: Odniesienie do kursu, w którym student otrzymał ocenę.

f) Request

Model reprezentuje wniosek złożony przez studenta w systemie. Może dotyczyć różnych spraw, takich jak urlop czy przedłużenie terminu egzaminu. Przechowuje szczegóły wniosku, datę złożenia oraz jego status.

Pola w modelu:

- **RequestId**
  - Typ: int
  - Walidacja: Klucz główny (automatycznie generowany).
- **RequestType**
  - Typ: string
- **Description**
  - Typ: string
- **DateSubmitted**
  - Typ: DateTime
- **Status**
  - Typ: string
- **StudentId**
  - Typ: string
  - Walidacja: Klucz obcy wskazujący na użytkownika (studenta), który złożył wniosek.
- **Student**
  - Typ: ApplicationUser
  - Walidacja: Odniesienie do studenta składającego wniosek.

Dodatkowo folder Models zawiera podfolder ViewsModels. Modele widoków (ViewModels) dostosowane do wymagań widoków. Służą do przekazywania danych między kontrolerami a widokami.Wyodrębnono, gdyż dane wymagane w widokach pochodzą z różnych modeli domenowych lub wymagana jest inna struktura danych.

a) AssignRolesViewModel

Model AssignRolesViewModel reprezentuje dane używane w widoku do przypisywania ról użytkownikom przez administratora lub dziekana. Umożliwia zarządzanie użytkownikami, listą dostępnych ról oraz wyborem ról do przypisania.

Pola w modelu:

- **UserId**
  - Typ: string?
  - Opis: Przechowuje identyfikator użytkownika, któremu przypisujemy role.
  - Walidacja: [Required].
- **SelectedRoles**
  - Typ: List<string>
  - Opis: Lista ról wybranych przez administratora do przypisania użytkownikowi.
  - Walidacja: [Required].
- **AvailableRoles**
  - Typ: List<string>
  - Opis: Lista wszystkich dostępnych ról do wyboru.
- **Users**
  - Typ: List<UserViewModel>
  - Opis: Lista użytkowników dostępnych do zarządzania w widoku.

c) ChangePasswordViewModel

Model ChangePasswordViewModel jest używany do obsługi zmiany hasła przez użytkownika. Przechowuje dane wprowadzone przez użytkownika, takie jak obecne hasło, nowe hasło i jego potwierdzenie, oraz zawiera odpowiednie walidacje.

Pola w modelu:

- **CurrentPassword**
  - Typ: string
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [DataType(DataType.Password)]: Wskazuje, że dane będą maskowane w widoku jako hasło.
    - ErrorMessage: "Obecne hasło jest wymagane."
  - Opis: Przechowuje obecne hasło użytkownika.

- **NewPassword**
  - Typ: string
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [StringLength(100, MinimumLength = 6)]: Hasło musi mieć długość od 6 do 100 znaków.
    - [DataType(DataType.Password)]: Wskazuje, że dane będą maskowane w widoku jako hasło.
    - ErrorMessage: "Nowe hasło jest wymagane." oraz "Hasło musi mieć co najmniej {2} znaków."
  - Opis: Przechowuje nowe hasło użytkownika.
- **ConfirmPassword**
  - Typ: string
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [DataType(DataType.Password)]: Wskazuje, że dane będą maskowane w widoku jako hasło.
    - [Compare("NewPassword")]: Walidacja sprawdza, czy potwierdzenie hasła pasuje do nowego hasła.
    - ErrorMessage: "Potwierdzenie nowego hasła jest wymagane." oraz "Hasła nie pasują do siebie."
  - Opis: Przechowuje potwierdzenie nowego hasła użytkownika.

c) ErrorViewModel

Model ErrorViewModel jest prostym modelem widoku używanym do przekazywania informacji o błędach w aplikacji. Umożliwia wyświetlanie identyfikatora żądania w przypadku wystąpienia błędu.

Pola w modelu:

- **RequestId**
  - Typ: string?
  - Walidacja: Brak.
  - Opis: Przechowuje identyfikator żądania, który może być używany do śledzenia błędów w systemie.
- **ShowRequestId**
  - Typ: bool
  - Walidacja: Brak.
  - Opis: Wartość logiczna wskazująca, czy identyfikator żądania powinien być wyświetlany.

d) GradeInputViewModel

Model **GradeInputViewModel** służy do przekazywania danych związanych z wprowadzaniem ocen dla kursu. Reprezentuje informacje o kursie i liście studentów z ich ocenami.

Pola w modelu:

- **CourseId**
  - Typ: int
  - Walidacja: Brak.
  - Opis: Identyfikator kursu, dla którego są wprowadzane oceny.
- **CourseName**
  - Typ: string?
  - Walidacja: Brak.
  - Opis: Nazwa kursu, dla którego są wprowadzane oceny.
- **Students**
  - Typ: List<StudentGradeInput>?
  - Walidacja: Brak.
  - Opis: Lista studentów zapisanych na kurs z informacjami potrzebnymi do wprowadzenia ocen.

e) LecturerDashboardViewModel

Model LecturerDashboardViewModel reprezentuje dane używane w widoku panelu wykładowcy. Przechowuje listę kursów prowadzonych przez wykładowcę i może być rozszerzony o inne dane potrzebne w dashboardzie.

Pola w modelu:

- **Courses**
  - Typ: IEnumerable<Course>?
  - Opis: Lista kursów przypisanych do wykładowcy, które mogą być wyświetlane w jego panelu.

f) LecturerLoadViewModel

Model LecturerLoadViewModel reprezentuje dane dotyczące obciążenia wykładowcy, takie jak liczba prowadzonych kursów i łączna liczba punktów ECTS. Służy do prezentowania informacji związanych z obciążeniem w widoku.

Pola w modelu:

- **LecturerName**
  - Typ: string
  - Opis: Imię i nazwisko wykładowcy.

- **CourseCount**
  - Typ: int
  - Opis: Liczba kursów prowadzonych przez wykładowcę.
- **TotalECTS**
  - Typ: int
  - Opis: Łączna liczba punktów ECTS przypisanych do prowadzonych kursów.

g) LoginViewModel

Model LoginViewModel jest używany do przekazywania danych logowania użytkownika między widokiem a kontrolerem. Służy do uwierzytelnienia użytkownika na podstawie jego danych logowania.

Pola w modelu:

- **Email**
  - Typ: string
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [EmailAddress]: Walidacja formatu e-mail.
  - Opis: Adres e-mail użytkownika używany do logowania.
- **Password**
  - Typ: string
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [DataType(DataType.Password)]: Wskazuje, że dane są hasłem i powinny być maskowane w widoku.
  - Opis: Hasło użytkownika do logowania.
- **RememberMe**
  - Typ: bool
  - Walidacja: Brak.
  - Opis: Wskazuje, czy użytkownik chce pozostać zalogowany na danym urządzeniu.

h) RegisterViewModel

Model RegisterViewModel służy do przekazywania danych rejestracyjnych użytkownika między widokiem a kontrolerem. Umożliwia tworzenie nowego konta w systemie, w tym wprowadzenie podstawowych danych użytkownika oraz hasła.

Pola w modelu:

- **Email**
  - Typ: string?
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [EmailAddress]: Walidacja formatu e-mail.
  - Opis: Adres e-mail użytkownika, który będzie używany jako login.
- **FirstName**
  - Typ: string?
  - Walidacja:
    - [Required]: Pole jest wymagane.
  - Opis: Imię użytkownika.
- **LastName**
  - Typ: string?
  - Walidacja:
    - [Required]: Pole jest wymagane.
  - Opis: Nazwisko użytkownika.
- **Password**
  - Typ: string?
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [DataType(DataType.Password)]: Wskazuje, że dane to hasło, które powinno być maskowane w widoku.
  - Opis: Hasło użytkownika.
- **ConfirmPassword**
  - Typ: string?
  - Walidacja:
    - [Required]: Pole jest wymagane.
    - [DataType(DataType.Password)]: Wskazuje, że dane to hasło.
    - [Compare("Password", ErrorMessage = "Hasło i potwierdzenie hasła nie są zgodne.")]: Sprawdza zgodność potwierdzenia hasła z hasłem.
  - Opis: Potwierdzenie hasła użytkownika.

i) StatisticsViewModel

Model StatisticsViewModel służy do prezentacji statystyk studentów i wykładowców w widokach, takich jak dashboardy czy raporty. Agreguje dane studentów i wykładowców w ustrukturyzowanej formie.

Pola w modelu:

- **TopStudents**
  - Typ: List<StudentData>
  - Walidacja: Brak.
  - Opis: Lista studentów z najlepszymi wynikami, zawierająca szczegółowe dane, takie jak imię, nazwisko, średnia ważona oraz liczba kursów.
- **LecturerStatistics**
  - Typ: List<LecturerData>
  - Walidacja: Brak.
  - Opis: Lista statystyk wykładowców, takich jak liczba prowadzonych kursów i suma punktów ECTS przypisanych do ich kursów.

j) StudentGradeViewModel

Model StudentGradeViewModel służy do prezentacji szczegółów dotyczących ocen studenta w widokach, takich jak historia ocen lub dashboard studenta.

Pola w modelu:

- **CourseName**
  - Typ: string
  - Opis: Nazwa kursu, w ramach którego została przypisana ocena.
- **GradeValue**
  - Typ: decimal
  - Opis: Wartość oceny, np. w formacie liczbowym (np. procentowym).
- **DateAssigned**
  - Typ: DateTime
  - Opis: Data przypisania oceny studentowi

k) SubmitRequestViewModel

Model SubmitRequestViewModel reprezentuje dane wymagane do złożenia nowego wniosku przez użytkownika. Obejmuje pola, które umożliwiają określenie rodzaju wniosku i jego opisu, wraz z odpowiednią walidacją.

Pola w modelu:

- **RequestType**
  - Typ: string?
  - Walidacja:
    - [Required]: Pole musi zostać wypełnione.
  - Opis: Typ wniosku, np. "Urlop", "Zmiana terminu egzaminu".
- **Description**
  - Typ: string?
  - Walidacja:
    - [Required]: Pole musi zostać wypełnione.
    - [StringLength(500)]: Maksymalna długość opisu to 500 znaków.
    - ErrorMessage: "Opis nie może przekraczać 500 znaków."
  - Opis: Szczegółowy opis wniosku, który wyjaśnia jego cel.

6.Kontrolery i metody użyte w projekcie

W projekcie wykorzystano następujące kontrolery oraz metody:

a) **AccountController**

Kontroler odpowiedzialny za zarządzanie operacjami związanymi z kontami użytkowników, takimi jak rejestracja, logowanie, wylogowanie oraz zmiana hasła. Korzysta z usług **UserManager**, **SignInManager**, i **RoleManager** dla obsługi użytkowników, uwierzytelnienia i ról.

Metody:

1. Rejestracja użytkownika

Register (GET): Wyświetla formularz rejestracji.

Register (POST): Obsługuje tworzenie nowego konta użytkownika.

2. Logowanie i wylogowanie

Login (GET): Wyświetla formularz logowania.

Login (POST): Obsługuje logowanie użytkownika i przekierowanie na podstawie roli.

Logout (POST): Wylogowuje użytkownika i przekierowuje na stronę główną.

3. Zmiana hasła

ChangePassword (GET): Wyświetla formularz zmiany hasła (tylko dla zalogowanych użytkowników).

ChangePassword (POST): Obsługuje proces zmiany hasła.

b) AdminController

Metody:

Index: Wyświetla główną stronę panelu administratora.

Students: Wyświetla listę wszystkich studentów.

AddStudent (GET): Wyświetla formularz dodania nowego studenta.

AddStudent (POST): Obsługuje tworzenie nowego konta studenta.

EditStudent (GET): Wyświetla formularz edycji danych istniejącego studenta.

EditStudent (POST): Obsługuje zapis zaktualizowanych danych studenta.

Courses: Wyświetla listę wszystkich kursów.

CreateCourse (GET): Wyświetla formularz tworzenia nowego kursu.

CreateCourse (POST): Obsługuje tworzenie nowego kursu i przypisywanie wykładowców.

EditCourse (GET): Wyświetla formularz edycji istniejącego kursu.

EditCourse (POST): Obsługuje zapis zaktualizowanych danych kursu.

CourseRegistrations: Wyświetla wszystkie rejestracje kursów przez studentów.

UpdateRegistrationStatus: Aktualizuje status rejestracji kursu (np. zatwierdzona,

odrzucona).

Requests: Wyświetla listę wszystkich wniosków złożonych przez studentów.

UpdateRequestStatus: Aktualizuje status wniosku (np. zatwierdzony, odrzucony).

AssignRoles (GET): Wyświetla formularz przypisywania ról użytkownikom.

AssignRoles (POST): Obsługuje przypisanie nowych ról użytkownikowi.

Statistics: Generuje dane statystyczne:

- Najlepsi studenci (średnia ważona ocen, liczba kursów).
- Wykładowcy (liczba prowadzonych kursów, suma punktów ECTS).

c) HomeController

HomeController jest kontrolerem odpowiadającym za obsługę głównych stron aplikacji:

Strona główna obsługuje zarówno użytkowników niezalogowanych (wyświetlenie widoku), jak i zalogowanych (przekierowanie na dashboard zgodny z rolą użytkownika).

Obsługuje dodatkowe funkcjonalności, takie jak polityka prywatności i obsługa błędów. Jest to podstawowy punkt wejścia do aplikacji.

Metody:

- **Index:**
  - Wyświetla stronę główną aplikacji.
  - Jeśli użytkownik jest zalogowany, sprawdza jego rolę i przekierowuje na odpowiedni dashboard (Student, Lecturer, Admin).
  - Przekazuje imię użytkownika do widoku za pomocą ViewBag.
- **Privacy:**
  - Opis: Wyświetla stronę z informacją o polityce prywatności.
- **Error:**
  - Opis: Wyświetla stronę błędu z informacjami diagnostycznymi.
  - Zwracane dane: Widok błędu, zawierający szczegóły błędu (np. RequestId).

d) LecturerController

LecturerController to kontroler odpowiedzialny za obsługę funkcji dostępnych dla wykładowców:

- Umożliwia zarządzanie przypisanymi kursami i ocenami studentów.
- Weryfikuje, czy dany wykładowca ma dostęp do edytowanych danych (np. kursu).
- Wszystkie metody wymagają zalogowania użytkownika z rolą "LECTURER". Kontroler działa jako punkt wejścia do kluczowych funkcjonalności związanych z prowadzeniem kursów i ocenianiem studentów.

Metody:

- **Index**:
  - **Opis**: Wyświetla panel główny wykładowcy, w tym listę kursów przypisanych do zalogowanego wykładowcy.
  - **Parametry**: Brak.
  - **Zwracane dane**: Widok z danymi kursów i innymi informacjami dashboardu.
- **GradeInput (GET)**:
  - **Opis**: Wyświetla formularz umożliwiający wprowadzenie lub edycję ocen dla studentów na wybranym kursie.
  - **Parametry**:
    - **int courseId** – identyfikator kursu.
  - **Zwracane dane**: Widok formularza wprowadzania ocen.
- **SaveGrades (POST)**:
  - **Opis**: Obsługuje zapis lub aktualizację ocen wprowadzonych przez wykładowcę.
  - **Parametry**:
    - **GradeInputViewModel model** – model zawierający dane o kursie i ocenach studentów.
  - **Zwracane dane**: Przekierowanie na dashboard wykładowcy w przypadku sukcesu lub widok z błędami walidacyjnymi.
- **Courses (GET)**:
  - **Opis**: Wyświetla listę kursów przypisanych do zalogowanego wykładowcy.
  - **Parametry**: Brak.
  - **Zwracane dane**: Widok listy kursów.

e) **StudentController**

StudentController obsługuje funkcjonalności dostępne dla użytkowników z rolą "STUDENT":

- Rejestracja na kursy oraz przegląd dostępnych kursów.
- Przegląd ocen uzyskanych na ukończonych kursach.
- Obsługa wniosków studenckich, takich jak zgłaszanie i przeglądanie.
- Wyświetlenie profilu użytkownika.
  Wszystkie metody są zabezpieczone przed nieautoryzowanym dostępem za pomocą atrybutu [Authorize(Roles = "STUDENT")].

Metody:

- **Index (GET)**:
  - **Opis**: Wyświetla dashboard studenta z przekazaniem imienia użytkownika do widoku.
  - **Zwracane dane**: Widok strony głównej studenta.

- **RegisterCourse (GET)**:
  - **Opis**: Wyświetla listę dostępnych kursów, na które student może się zarejestrować.
  - **Zwracane dane**: Widok z listą kursów.

- **RegisterCourse (POST)**:
  - **Opis**: Rejestruje studenta na wybrany kurs, sprawdzając, czy student nie jest już zarejestrowany.
  - **Parametry**:
    - **int courseId** – identyfikator kursu, na który student chce się zarejestrować.
  - **Zwracane dane**: Przekierowanie na widok dostępnych kursów z odpowiednimi komunikatami o sukcesie lub błędach.

- **GradeOverview (GET)**:
  - **Opis**: Wyświetla listę ocen studenta dla wszystkich ukończonych kursów.
  - **Parametry**: Brak.
  - **Zwracane dane**: Widok z listą ocen (StudentGradeViewModel).

- **Profile (GET)**:
  - **Opis**: Wyświetla szczegóły profilu zalogowanego studenta.
  - **Parametry**: Brak.
  - **Zwracane dane**: Widok z danymi profilu.

- **SubmitRequest (GET)**:
  - **Opis**: Wyświetla formularz do zgłaszania wniosków studenckich.

    - **Parametry**: Brak.
    - **Zwracane dane**: Widok formularza zgłoszenia (SubmitRequestViewModel).
- **SubmitRequest (POST)**:
    - **Opis**: Obsługuje przesyłanie wniosków przez studenta.
    - **Parametry**:
        - **SubmitRequestViewModel model** – model zawierający dane zgłoszenia (typ i opis wniosku).
    - **Zwracane dane**: Przekierowanie na listę wniosków z komunikatem o sukcesie lub ponowne załadowanie widoku w przypadku błędów.
- **ViewRequests (GET)**:
    - **Opis**: Wyświetla wszystkie zgłoszone wnioski przez zalogowanego studenta, posortowane według daty.
    - **Zwracane dane**: Widok listy wniosków.

7. Struktura użytkowników

W Systemie Zarządzania Dziekanatu wykorzystano mechanizm **ASP.NET Identity** do tworzenia kont użytkowników. Użytkownicy w systemie mogą mieć przypisaną jedną rolę spośród możliwych:

- student,
- wykładowca,
- dziekan (admin)

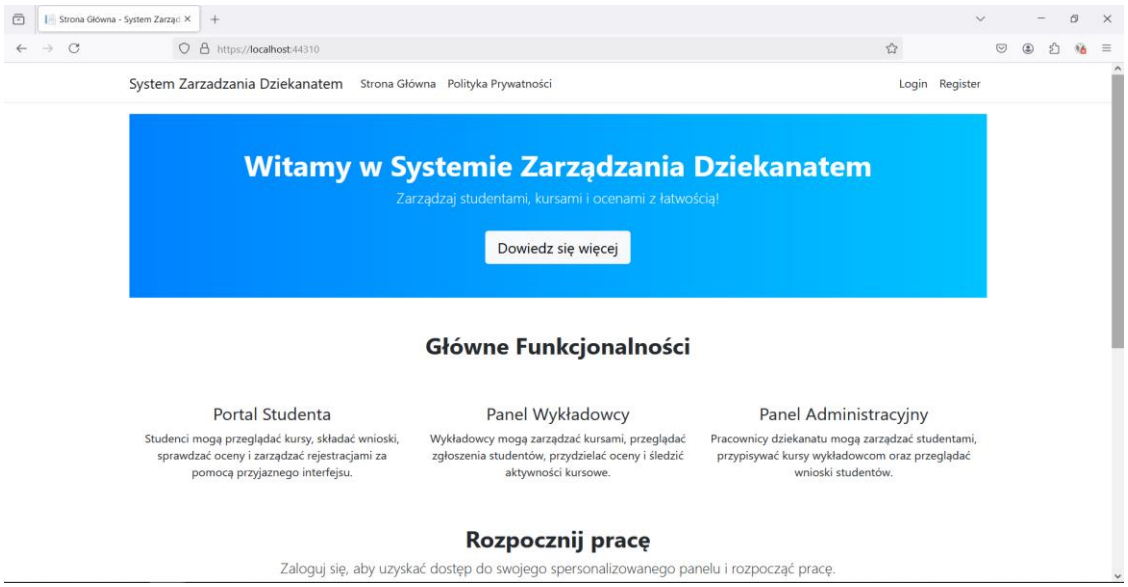
Pierwszy utworzony w ramach systemu użytkownik z automatu ma przypisaną rolę **dziekana**.

Następni użytkownicy tworzeni tą drogą są bez przypisanych ról. Rolę użytkownikom przypisuje lub zmienia dziekan w ramach modelu **AssignRolesViewModel**

Użytkownik niezalogowany widzi stronę informacyjną systemu. Zalogowany użytkownik uzyskuje dostęp do funkcjonalności zgodnych z jego rolą w systemie:

8. Działanie systemu

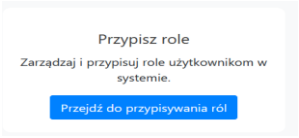
Widok niezalogowanego użytkownika:



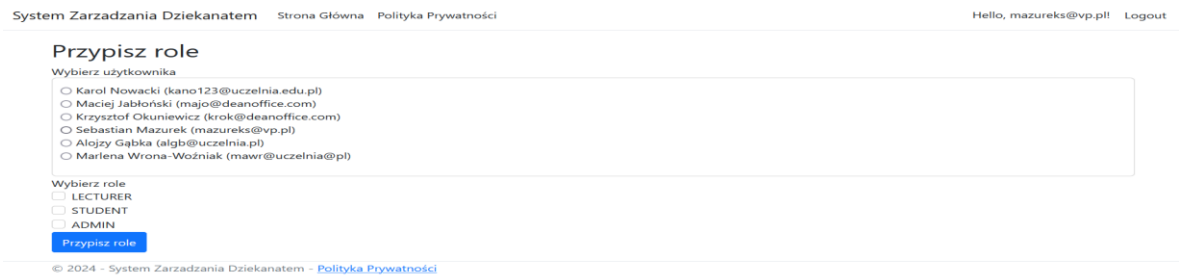
Aby w pełni można było zaprezentować funkcjonowanie systemu , oprócz dziekana, w systemie powinno być wpisanych co najmniej 3 studentów i co najmniej dwóch wykładowców (potrzebni do statystyk i rankingów).

Pierwsze utworzone przez rejestrację konto ma prawa dziekana. Następne tworzone tą drogą konta nie mają przypisanej roli. Rolę przypisuje i zmienia dziekan (w kodzie nazywany ***Admin***) stosownym interfejsem:

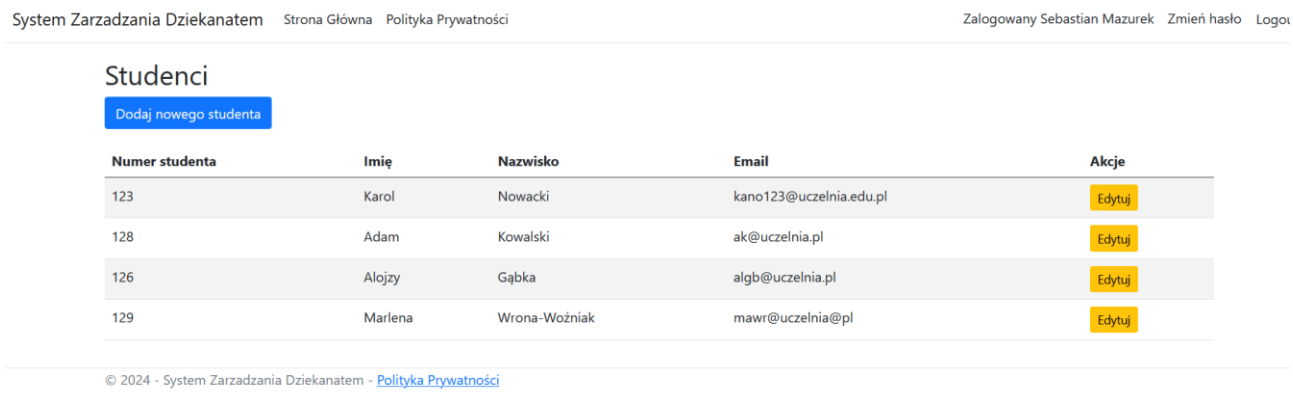
Interfejs ustalania i zmiany ról w systemie: uruchamiany jest przyciskiem :



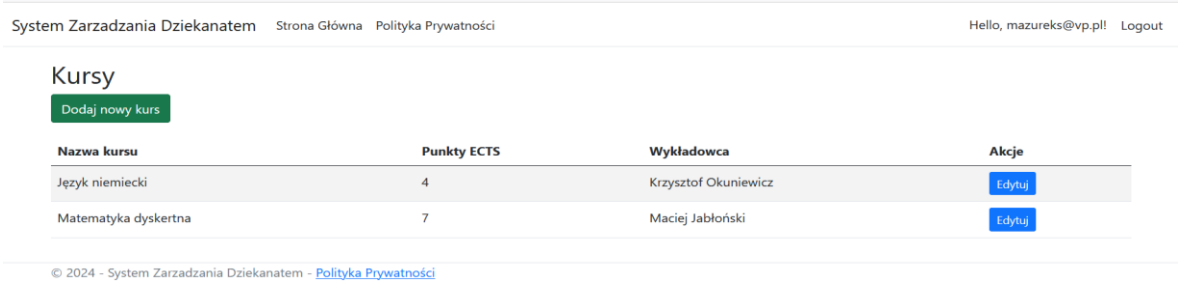
Administrator / dziekan ma do dyspozycji funkcjonalność zmiany lub przypisania roli użytkownikowi systemu. Funkcjonalność uruchamia przyciskiem:



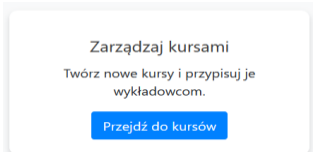
Załóżmy że mamy wpisanych studentów



Oraz dwu wykładowców.

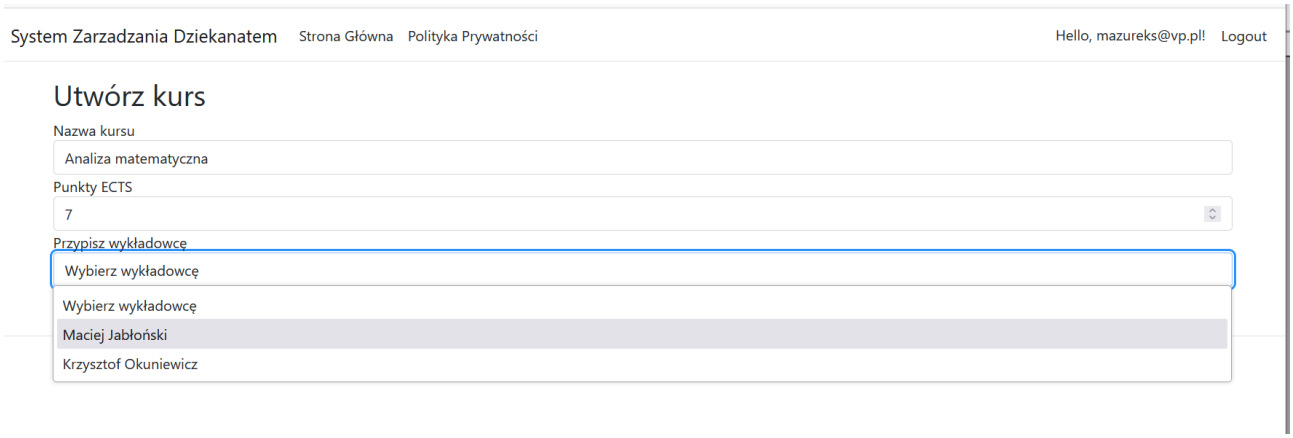


Dziekan przez przycisk:



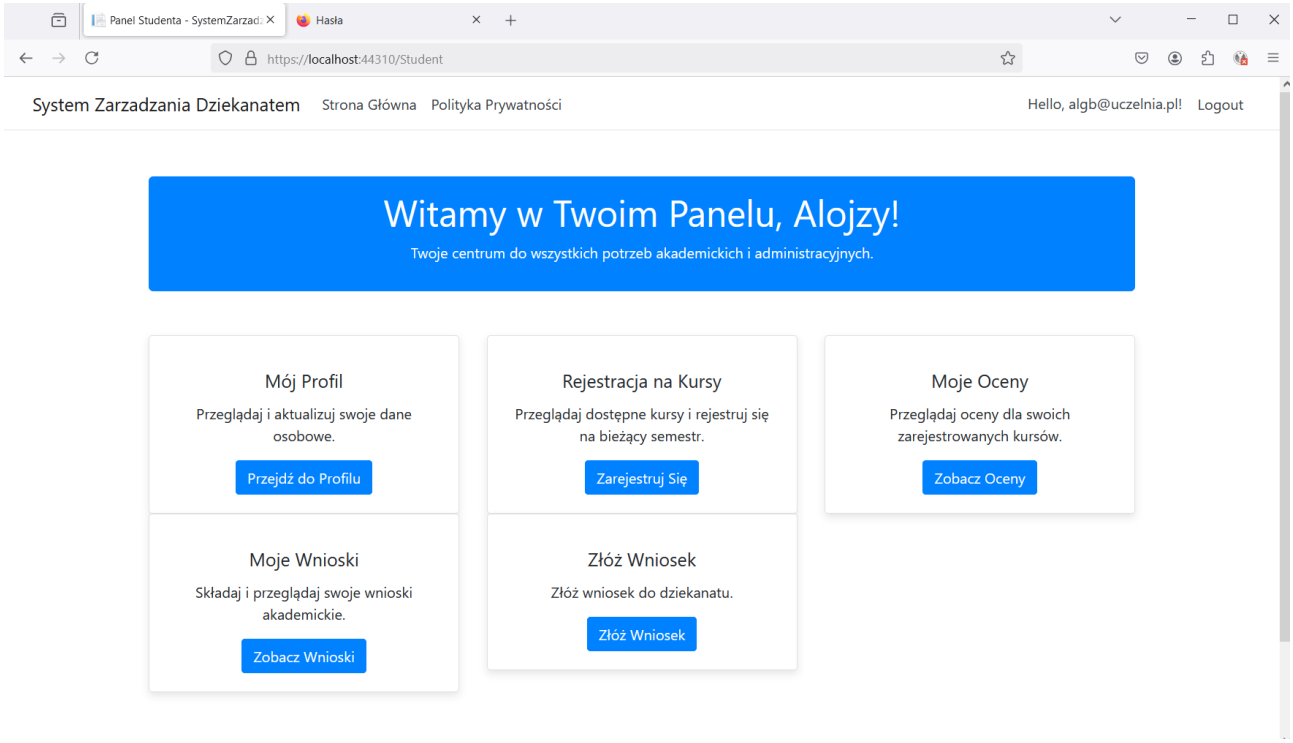
Uruchamia panel zarządzania kursami, na którym widoczny jest wykaz kursów:

Panel daje możliwość dopisania nowego kursu lub edycji już istniejących. Po kliknięciu przycisku „Dodaj kurs” pojawia się panel wprowadzania kursu. Dziekan musi wprowadzić nazwę kursu , jego punktację ECTS oraz przypisać wykładowcę wybierając go z listy dostępnych wykładowców.

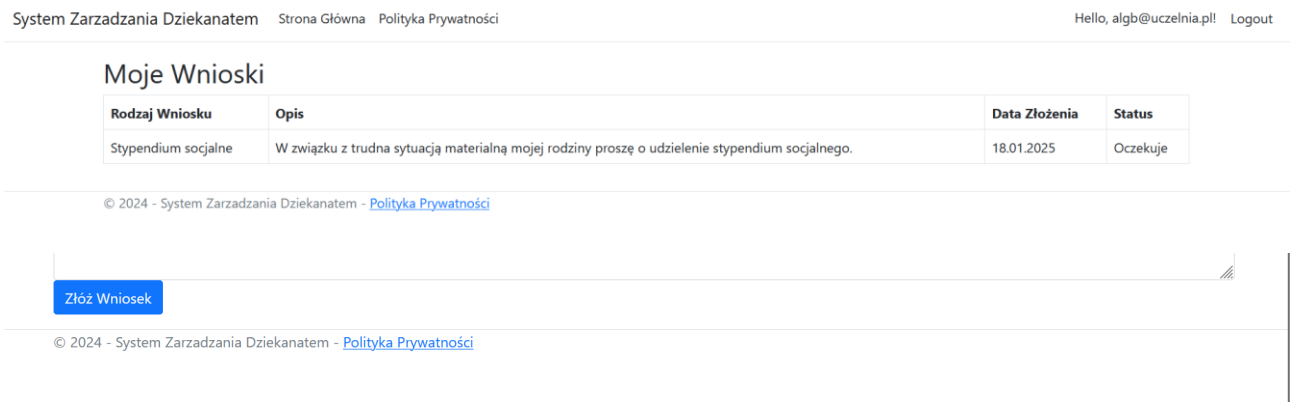


Na koniec przyciskiem „Utwórz kurs” kurs zostaje wprowadzony.

Panel studenta.
Zalogowany student ma dostępny panel przycisków udostępniających funkcjonalności:



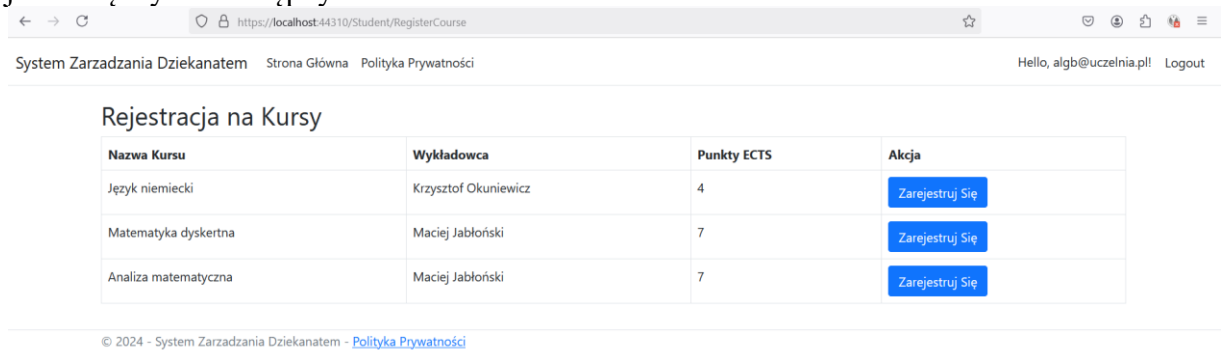
Przykład złożenie wniosku:
Po kliknięciu złóż wniosek, studentowi ukazuje się widok, w którym należy wybrać kategorię z listy rozwijanej. Uzupełnić pole szczegóły i kliknąć przycisk „Złóż Wniosek”. Po kliknięciu tego przycisku wniosek zostaje złożony, oraz widok przechodzi do wykazu wniosków danego studenta:



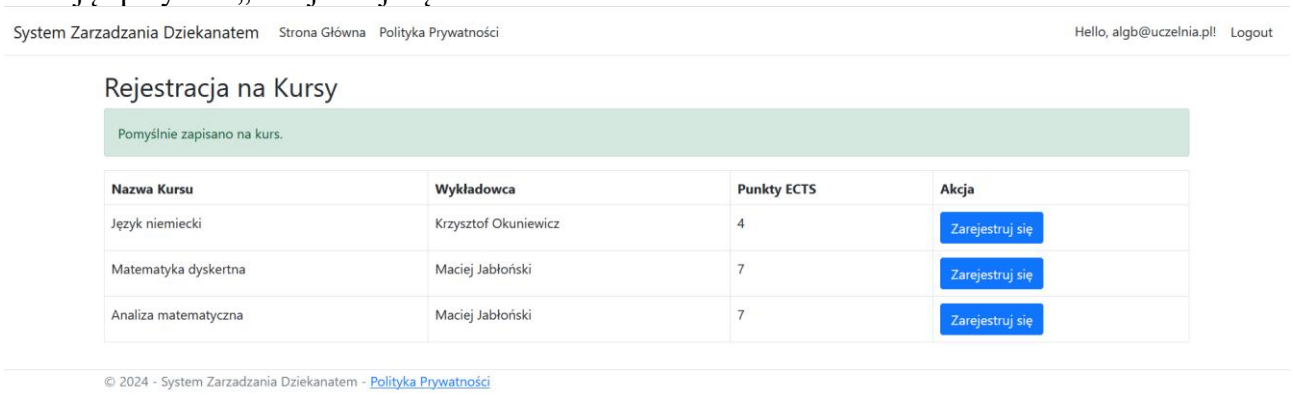
Funkcjonalność studenta – rejestracja na kurs.

Jeżeli są już wpisane przez dziekana kursy, studenci mogą się na nie zapisywać. Funkcjonalność dostępna jest po kliknięciu na przycisk „Rejestracja na Kursy”:

Pojawia się wykaz dostępnych kursów:

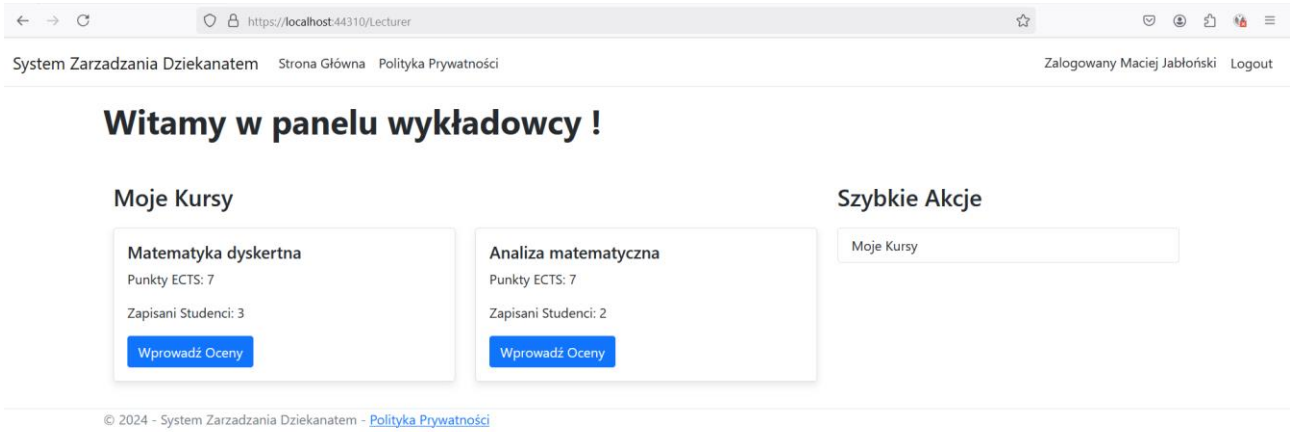


Klikając przycisk „Zarejestruj się”



Funkcjonalność wykładowcy

Po zalogowaniu osoby z rolą Wykładowcy udostępniony zostaje panel wykładowcy:



Widoczny jest wykaz kursów. Po kliknięciu w przycisk prowadzonego kursu, pojawia się panel wprowadzania i modyfikacji ocen studentów. Oceny wyrażane są w %:

Funkcjonalność Dziekana

Dziekan ma możliwość sporządzania rankingu wyników studentów.

1. Wyjaśnienie logiki rankingu

Ocena ważona dla studenta:

Dla każdego studenta obliczamy średnią ważoną według wzoru:

$$\text{Średnia ważona} = \frac{\sum(\text{ECTS kursu} \times \text{Ocena z kursu})}{\sum(\text{ECTS kursu, dla których jest ocena})}$$

1. **Suma liczników**:
   Sumujemy iloczyn punktów ECTS kursu i oceny z kursu dla wszystkich zarejestrowanych kursów, gdzie ocena została wystawiona.
2. **Suma mianowników**:
   Sumujemy tylko punkty ECTS kursów, dla których student ma ocenę.
3. **Ranking**:
   Sortujemy studentów malejąco według ich średniej ważonej.
4. Filtrujemy wyniki – wyświetlamy tylko trzech najlepszych studentów.

Dodatkowo w panelu statystyki obliczane jest obciążenie dydaktyczne wykładowców:

System Zarzadzania Dziekanatem Strona Główna Polityka Prywatności Zalogowany Sebastian Mazurek Logout

Ranking najlepszych studentów

Pozycja	Imię i nazwisko	Średnia ważona	Liczba kursów
1	Alojzy Gąbka	73,45	4
2	Adam Kowalski	50,50	2
3	Karol Nowacki	38,00	2

Raport obciążenia dydaktycznego wykładowców

Imię i nazwisko	Liczba kursów	Suma punktów ECTS
Maciej Jabłoński	2	14
Krzysztof Okuniewicz	2	6