POLITECHNIKA KRAKOWSKA

IM. TADEUSZA KOŚCIUSZKI

WYDZIAŁ FIZYKI MATEMATYKI I INFORMATYKI

KIERUNEK INFORMATYKA

DANIEL ŁASAK

**WIELOWARSTWOWA APLIKACJA WSPOMAGAJĄCA PROWADZENIE ZAJEĆ DYDAKTYCZNYCH NA UCZELNI WYŻSZEJ**

**MULTI-TIER APPLICATION SUPPROTING TEACHING ACTIVITIES AT THE UNIVERSITY**

PRACA INŻYNIERSKA

STUDIA STACJONARNE

Promotor: Dr Agnieszka Krok

Kraków 2014

Spis treści

[1. Wstęp 2](#_Toc397332180)

[1.1. Pojęcie i rozwój architektury wielowarstwowej 3](#_Toc397332181)

[1.2. Zastosowanie systemów informatycznych w procesie prowadzenia zajęć dydaktycznych 5](#_Toc397332182)

[1.3. Cel i zakres pracy 5](#_Toc397332183)

[2. Specyfikacja wymagań 6](#_Toc397332184)

[2.1. Identyfikacja aktorów 6](#_Toc397332185)

[2.2. Wymagania funkcjonalne 6](#_Toc397332186)

[2.3. Wymagania niefunkcjonalne 9](#_Toc397332187)

[3. Model systemu 10](#_Toc397332188)

[3.1. Diagram klas modelu domeny 11](#_Toc397332189)

[3.2. Diagram klas odpowiadający komponentom 13](#_Toc397332190)

[3.3. Diagram komponentów 14](#_Toc397332191)

[3.4. Diagram wdrożenia 15](#_Toc397332192)

[4. Wybór i opis technologii zastosowanych w projekcie 16](#_Toc397332193)

[4.1. Platforma Java Enterprise Edition 17](#_Toc397332194)

[4.2. Framework Spring 17](#_Toc397332195)

[4.3. Biblioteki i narzędzia wspomagające 18](#_Toc397332196)

[5. Opis implementacji 20](#_Toc397332197)

[5.1. Struktura katalogów i plików projektu 20](#_Toc397332198)

[5.2. Rozwarstwienie i zarządzanie komponentami 21](#_Toc397332199)

[5.3. Warstwa dostępu do danych 22](#_Toc397332200)

[5.4. Warstwa logiki biznesowej 25](#_Toc397332201)

[5.5. Warstwa prezentacji 27](#_Toc397332202)

[5.6. Uwierzytelnienie i autoryzacja dostępu 28](#_Toc397332203)

[5.6.1. Konfiguracja modelu uprawnień 29](#_Toc397332204)

[5.6.2. Zabezpieczenie aplikacji 30](#_Toc397332205)

[6. Testowanie 31](#_Toc397332206)

[6.1. Testy jednostkowe 31](#_Toc397332207)

[6.2. Testy integracyjne 33](#_Toc397332208)

[6.3. Testy graficznego interfejsu użytkownika 34](#_Toc397332209)

[7. Prezentacja i opis działania systemu 34](#_Toc397332210)

[8. Wnioski 39](#_Toc397332211)

[9. Bibliografia 41](#_Toc397332212)

# Wstęp

Zrealizowany projekt został wykonany na potrzeby pracy inżynierskiej na kierunku Informatyka na Politechnice Krakowskiej. Poniższy rozdział prezentuje kontekst podjętego przeze mnie tematu - przedstawia pojęcie i rozwój architektury wielowarstwowej, zastosowanie informatyki w procesie prowadzenia zajęć dydaktycznych oraz określa dokładne cele i zakres pracy.

## Pojęcie i rozwój architektury wielowarstwowej

Sposób organizowania elementów systemu określany jest mianem architektury. Jej zadaniem jest określenie jak poskładać wszystkie elementy w całość, która będzie dostarczać rozwiązanie spełniające wymagania biznesowe stawiane przez klienta.

Początkowo ~~w~~ wraz z powstaniem obszaru oprogramowania stosowano podejście monolityczne, co do kształtu i budowy systemów informatycznych. Wszystkie czynności i zadania bez względu na ich charakter były przeplatane pomiędzy sobą by tak dostarczyć gotowe rozwiązanie. Wadą takiego modelu jest ścisłe przyleganie elementów powodując utratę skalowalności. Taki system z biegiem czasu staje się trudny, a nawet niemożliwy w dalszym utrzymaniu.

W procesie analizy budowy oprogramowania pojawił się krok, który pozwolił zmienić podejście to sprawy. To właśnie powstanie architektury typu klient-serwer (architektura dwuwarstwowa) otworzyło nowe możliwości w procesie kształtowania budowy systemów.

Z jej pomocą możliwe jest rozdzielenie zadań pomiędzy różne jednostki wykonawcze z użyciem sieci komputerowych. Istnieją dwie odmiany tej architektury, pierwsza „fat client/thin server”, która określa, że logika aplikacji i prezentacji znajduje się po stronie klienta, a dostęp do serwera bazy danych uzyskuje się przy użyciu różnego rodzaju API dla języka SQL. Druga forma to to „thin client/fat server” i wskazuje ona, że serwer bazy danych również może realizować pewne funkcje logiki aplikacji. Ma to miejsce w przypadku niektórych baz danych np. Oracle przy użyciu procedur składowanych. Jednak „fat client/thin server” znacząco obciąża sieć danymi, zaś „thin client/fat server” w przypadku awarii powoduje zmniejszenie wydajności pracy systemu z przyczyn wpływu operacji logiki aplikacji po stronie serwera na procesy bazy danych.

W końcu zaprojektowano architekturę trójwarstwową, jako nowe rozwiązanie usprawniające poprzedni model. Teraz, klient oraz serwer bazy danych został odciążony od operacji wykonujących logikę aplikacji[1].

W tej architekturze wyróżnić można następujące warstwy i ich funkcje:

1. Warstwa klienta

W najmniejszym zakresie do jej zadań należy prezentacja wyników oraz inicjacja żądań logiki aplikacji znajdującej się w warstwie drugiej. Najczęściej reprezentowana jest w postaci przeglądarki internetowej.

1. Warstwa aplikacji

Jest odpowiedzialna za właściwe przetwarzanie żądań od klientów, realizuje operacje logiki aplikacji związane z wymaganiami ustalonymi przez klienta.

1. Warstwa danych

Odpowiedzialna jest za składowanie oraz wyszukiwanie danych. Zazwyczaj odpowiada procesowi systemu zarządzania bazą danych.

Wykorzystanie podejścia komponentowego oraz wprowadzenia serwerów aplikacyjnych, jako środowiska pozwalającego na wdrażanie i uruchamianie dowolnych aplikacji w warstwie drugiej dla wyżej omówionego modelu przyczyniło się do powstania architektury wielowarstwowej [2].

Architekturą wielowarstwową nazywamy powiększoną architekturę trójwarstwową poprzez kolejny podział warstwy drugiej. Jednak ma ona charakter czysto logiczny, a proces rozwarstwienia polega na wertykalnym grupowaniu komponentów oprogramowania ze względu na ich funkcjonalność, które działają pod kontrolą serwera aplikacyjnego. Wprowadzenie dodatkowych warstw komponentów jest przydatne głównie dla programistów i architektów oprogramowania. W literaturze podział na grupy komponentów określany jest też mianem rozwarstwienia logicznego (multilayer), ponieważ zazwyczaj ma zastosowanie w ramach jednej lokalizacji fizycznej, np. serwer aplikacyjny. [3].

Architektura wielowarstwowa jest najbardziej powszechną, jeśli chodzi o systemy klasy enterprise. Głównym czynnikiem jej popularności jest zwiększona skalowalność oraz niezależność fragmentów systemu np. powiększenie przestrzeni dyskowej bez konieczności ingerencji w stację, na której działa serwer aplikacyjny, albo zmiana sposobu pobierania informacji z bazy danych bez konieczności zmiany reszty programowych komponentów odpowiedzialnych za prezentacje wyników. Również klasyfikacja programowych elementów systemu umożliwia rozdzielenie prac na różne zespoły posiadające odpowiednią wiedzę i umiejętności. Architektura wielowarstwowa jest obecnie sprawdzonym rozwiązaniem w projektowaniu i budowie skalowalnych, niezawodnych oraz bezpiecznych systemów informatycznych wykorzystujących sieć komputerową. Do takich systemów można właśnie zaliczyć pewną grupę, których zadaniem jest wspomaganie procesu edukacji. W następnym rozdziale zostanie przedstawiony kontekst problemu, którego rozwiązanie posłuży, jako przykład realizacji omówionej architektury.

## Zastosowanie systemów informatycznych w procesie prowadzenia zajęć dydaktycznych

W dzisiejszej dobie nieustającej informatyzacji naszego życia można poszukiwać coraz to nowszych obszarów jej zastosowania. Takim przykładem może być proces prowadzenia zajęć dydaktycznych na uczelni wyższej. Wykładowcy w wraz z asystentami odpowiedzialni są za prowadzenie i przechowywanie listy ocen oraz obecności zazwyczaj w tradycyjnej formie lub w postaci arkusza kalkulacyjnego. Skutkuje to pewnym utrudnieniem w procesie komunikacji zarówno między prowadzącymi jak i samymi uczestnikami kursów, studentami. Osoba odpowiedzialna za prowadzenie przedmiotu chciałaby mieć wgląd w przebieg zajęć w trakcie semestru. By to osiągnąć asystenci zobowiązani są do regularnego przesyłania wyników najczęściej drogą mailową i w różnych postaciach. Studenci posiadają możliwość podglądu swoich wyników oraz obecności wyłącznie w trakcie trwania zajęć, co powoduje skrócenie czasu i spadek wydajności w procesie prowadzenia zajęć dydaktycznych. W dodatku sposób określania końcowych wyników jest procesem wymagającym dodatkowego, często żmudnego wysiłku.

Wychodząc naprzeciw tej sytuacji zostanie stworzony system ułatwiający przeprowadzenie tych czynności przez prowadzących zajęcia na uczelni wyższej. Usprawni on dostęp do danych związanych z procesem nauczania i tym samym ułatwi komunikację pomiędzy wszystkimi osobami zaangażowanymi w przebieg zajęć. Oszczędzi czas i ujednolici sposób przeprowadzania zajęć dydaktycznych w zakresie niektórych czynności. Taki system powinien być przede wszystkim bezpieczny i łatwy w użytkowaniu. Zakres jego usług powinien dostępny spoza jednostek uczelni, co powoduje wymóg wykorzystania istniejącej infrastruktury sieciowej. Wszystkie te cechy wskazują na zastosowanie architektury wielowarstwowej przy budowie opisywanego rozwiązania informatycznego.

## Cel i zakres pracy

Celem pracy jest projekt oraz implementacja systemu w architekturze wielowarstwowej, który wspomagać będzie proces prowadzenie zajęć dydaktycznych na uczelni wyższej.

W pracy zostanie pokazana i omówiona przykładowa realizacja wyżej wymienionej architektury z użyciem wybranych narzędzi informatycznych.

# Specyfikacja wymagań

Specyfikacja wymagań w bardziej szczegółowy sposób określa kształt oraz wymagania tworzonego systemu. W tym celu w pierwszym kroku zidentyfikowano aktorów wchodzących w interakcję z aplikacją. Następnie przedstawiono wszystkie czynności realizowane przez system z poziomu określonych użytkowników.

## Identyfikacja aktorów

W systemie zidentyfikowano czterech aktorów posiadających określone znaczenie. Ich nazwy oraz opis roli zostały przedstawione w formie tabelarycznej.

|  |  |
| --- | --- |
| Nazwa | Opis |
| Administrator | Użytkownik odpowiedzialny za zarządzanie wszystkimi użytkownikami w systemie |
| Wykładowca | Użytkownik reprezentujący rolę wykładowcy na uczelni wyższej. Odpowiedzialny za definiowanie nadzorowanych przez siebie przedmiotów oraz prowadzenie zajęć audytoryjnych. |
| Asystent | Użytkownik reprezentujący rolę asystenta na uczelni wyższej.  Odpowiedzialny za prowadzenie zajęć laboratoryjnych, ćwiczeniowych lub projektowych. |
| Student | Użytkownik reprezentujący studenta, który może uczestniczyć w zajęciach w ramach prowadzonych przedmiotów. |

## Wymagania funkcjonalne

Wymagania funkcjonalne określają czynności, które system powinien realizować by osiągnąć wyznaczony cel. Poniżej została przedstawiona lista pogrupowanych po roli funkcji, oferowanych przez tworzony system. Wymagania zostały zebrane na podstawie własnych obserwacji, gdyż autor pracy częściowo uczestniczył w omawianym procesie. Pomocną rolę odegrały także rozmowy z pracownikami dydaktycznymi, prowadzącymi zajęcia.

**Administrator:**

* Tworzenie, edycja różnych użytkowników systemu z jednoczesnym określeniem ich roli (administrator, wykładowca, asystent, student).

**Wykładowca:**

* Definiowanie, edytowanie, usuwanie wyłącznie prowadzonych przez siebie przedmiotów.
* Określanie liczby wykładów oraz zajęć w semestrze dla wszystkich grup
* Określanie rodzaju drugiej części zajęć w ramach przedmiotu (projekt, laboratoria lub ćwiczenia).
* Dodawanie, edytowanie, usuwanie grup wykładowych
* Dodawanie, usuwanie studentów dla grup wykładowych
* Dodawanie, edytowanie, usuwanie grup zajęciowych
* Przypisywanie asystentów do poszczególnych grup
* Definiowanie wag poszczególnych ocen w celu automatycznego obliczenia oceny końcowej z przedmiotu.
* Definiowanie procentowej skali ocen dla całego przedmiotu
* Edytowanie obecności dla grupach audytoryjnych
* Edytowanie ocen z egzaminu
* Możliwość podglądu wszystkich ocen studentów dla przedmiotu – cząstkowe z zajęć, egzamin, finalne automatycznie wyliczone z zajęć i z przedmiotu
* Podgląd wszystkich obecności studentów dla przedmiotu
* Edytowanie adresu email
* Możliwość poglądu adresu email studentów, wykładowców oraz asystentów

**Asystent:**

* + Dodawanie, usuwanie studentów dla prowadzonych grup zajęciowych
  + Definiowanie liczby ocen w semestrze dla grup laboratoryjnych, projektowych lub ćwiczeniowych
  + Definiowanie liczby niedozwolonych nieobecności po przekroczeniu, których student nie uzyskuje się oceny z zajęć
  + Edytowanie ocen studentów w prowadzonych grupach
  + Edytowanie obecności studentów w prowadzonych grupach
  + Możliwość podglądu ocen z zajęć dla studentów przypisanych do prowadzonych grup
  + Możliwość podglądu obecności z zajęć dla studentów przypisanych do prowadzonych grup
  + Edytowanie adresu email
* Możliwość poglądu adresu email studentów, wykładowców oraz asystentów

**Student:**

* + Edytowanie numeru albumu
  + Edytowanie adresu email
  + Możliwość podglądu swoich ocen (cząstkowe i końcowe)
  + Możliwość podglądu swoich obecności
  + Możliwość podglądu informacji kontaktowych na temat wykładowców oraz asystentów

Wymaganiem funkcjonalnym systemu, które nie jest związane bezpośrednio z żadnym użytkownikiem jest automatyczne obliczanie ocen.

Obliczanie oceny końcowej z przedmiotu występuje w przypadku edycji:

* Wagi dla oceny
* Skali ocen
* Obecności studentów w dowolnych grupach
* Ocen z egzaminu

Końcowy wynik przeliczany jest na podstawie zdefiniowanej przez prowadzącego skali ocen oraz sumy następujących składników:

* waga\_egzamin \* ocena\_egzamin
* waga\_zajęcia \* ocena\_zajęcia
* waga\_obecności \* (liczba\_obecności/liczba\_wszystkich\_wykładów) \* 5.0, gdzie suma poszczególnych wag jest równa 1.

Obliczanie oceny z zajęć laboratoryjnych, projektowych lub ćwiczeniowych przeprowadzane jest na podstawie średniej arytmetycznej ocen uzyskanych w ramach kursu i występuje w przypadku edycji:

* Obecności w grupach innych zajęciowych
* Ocen cząstkowych z zajęć
* Liczby ocen cząstkowych w semestrze z zajęć
* Liczby dozwolonych nieobecności

Dodatkowym kryterium uzyskania pozytywnej oceny z tej części zajęć jest nieprzekroczenie liczby niedozwolonych nieobecności zdefiniowanej przez asystenta prowadzącego grupę.

Każda zmiana oceny z zajęć skutkuje również ponownym przeliczeniem oceny końcowej z przedmiotu.

## Wymagania niefunkcjonalne

Kategoria wymagań niefunkcjonalnych przedstawia ograniczenia nałożone na funkcje oraz wydajność systemu. Pozwala na określenie innych oczekiwań, niż było to w przypadku wymagań funkcjonalnych. Poniżej przedstawiono w postaci tabel wszystkie wymagania, które będą miały wpływ na jakość działania tworzonego systemu.

|  |  |
| --- | --- |
| ID | NFR\_0 |
| Nazwa | Bezpieczeństwo danych |
| Opis | Dane związane z konkretnym użytkownikiem powinny być zabezpieczone przed niepowołanym dostępem. Nie można dopuścić do wycieku ocen lub obecności przechowywanych w systemie. |
| Uzasadnienie | Zapewnianie bezpieczeństwa wrażliwych danych systemu jest obligatoryjnym działaniem oferowanym przez prawie każdy system informatyczny. Wymaganie spowoduje wzrost zaufania aktualnych i przyszłych użytkowników. |

|  |  |
| --- | --- |
| ID | NFR\_1 |
| Nazwa | Łatwość użytkowania |
| Opis | System nie powinien sprawiać trudności w jego użytkowaniu. Interfejs użytkownika powinien być prosty i przejrzysty. |
| Uzasadnienie | Przejrzysty interfejs oszczędzi wysiłek na tworzenie dokładnej dokumentacji użytkowania systemu. |

|  |  |
| --- | --- |
| ID | NFR\_2 |
| Nazwa | Łatwość instalacji |
| Opis | By korzystać z systemu końcowy użytkownik będzie potrzebował wyłącznie przeglądarki internetowej obsługującej javascript.  Wspierane wersje przeglądarek: IE10+, Firefox 20+, Google Chrome 20+ |
| Uzasadnienie | Brak potrzeby instalacji dodatkowych elementów po stronie klienta spowoduje ułatwienie w dostępie do systemu i chęć korzystania z niego. |

|  |  |
| --- | --- |
| ID | NFR\_3 |
| Nazwa | Skalowalność |
| Opis | System powinien być przystosowany do przezroczystej rozbudowy urządzeń bez przerwy w działaniu systemu. Powinien również pozwolić na przyszłą rozbudowę jego elementów programowych bez wpływu, na jakość pracy systemu. |
| Uzasadnienie | Wraz ze wzrostem użytkowników lub funkcjonalności systemu, jakość jego działania nie zmieni się. |

Wszystkie zaprezentowane wymagania niefunkcjonalne podkreślają wybór zastosowania architektury wielowarstwowej.

# Model systemu

Przed procesem implementacji niezbędną czynnością jest spojrzenie na system z innej perspektywy. Właśnie do tego celu służy proces modelowania, który ma za zadanie przedstawić strukturę oraz działanie systemu w bardziej ujednolicony oraz abstrakcyjny sposób. Najczęściej w tym celu stosuje się określone notacje, które zwięźle i precyzyjnie artykułują złożone pomysły. W pracy wykorzystano metodykę obiektową do budowy struktur oraz elementów systemu, dlatego główną notacją będzie język UML, który jest standardem w przemyśle wytwórczym oprogramowania.

W projekcie architektura oparta została na trzech logicznych warstwach:

* Warstwa prezentacji
* Warstwa logiki biznesowej
* Warstwa dostępu do danych

Zastosowany podział klasyfikuje operacje na serwerze aplikacyjnym ze względu na dostarczane funkcje dla poszczególnych warstw fizycznych, opisanych w rozdziale 1.1.

## Diagram klas modelu domeny

W pierwszej kolejności należy stworzyć model koncepcyjny wszystkich elementów specyficznych dla naszego zagadnienia, tak zwany model domeny. Do tego celu użyto notacji diagramu klas, który używany jest do opisu struktury systemu używając takich określeń jak klasy, atrybuty i relacje pomiędzy nimi. Model domeny w architekturze wielowarstwowej jest anemiczny, ponieważ nie zawiera rozbudowanej logiki biznesowej i jest przeznaczony głównie dla komponentów rezydujących w warstwach logiki biznesowej i dostępu do danych, które przetwarzają ich składowe. Poniższy schemat przedstawia model związany z prowadzeniem zajęć dydaktycznych na uczelni wyższej.



Diagram klas modelu domeny

Zaprezentowane elementy mają następujące znaczenie w stosunku do modelowanego zagadnienia:

**Klasa *User***

Reprezentuje użytkownika systemu, który posiada określoną rolę. Zawiera atrybuty odzwierciedlające: nazwę użytkownika i hasło przetwarzane w procesie logowania, status aktywacji w systemie, imię i nazwisko. W zależności od roli niektóre atrybuty zostaną wykorzystane, są nimi: tytuł naukowy dla wykładowców i asystentów oraz numer albumu w przypadku studenta.

**Klasa *Subject***

Klasa reprezentująca przedmiot. Zawiera takie atrybuty jak: tytuł przedmiotu, rok akademicki oraz numer semestru, typ drugiej części zajęć (laboratoria, ćwiczenia, projekt), współczynniki ocen cząstkowych, progi procentowe dla skali ocen oraz użytkownika reprezentującego wykładowcę odpowiedzialnego za konkretny przedmiot.

**Klasa *Group***

Reprezentacja grupy wykładowej, laboratoryjnej, ćwiczeniowej lub projektowej. Posiada związek z przedmiotem. Zawiera numer grupy, jej opis, rodzaj (podział na audytoryjną i zajęciową), liczbę ocen w semestrze oraz liczbę niedozwolonych nieobecności w przypadku grup innych niż audytoryjne. W tej klasie są również przechowywane daty poszczególnych zajęć.

**Klasa *GroupAssociation***

Reprezentacja przynależności studenta do grupy. Jest przypisaniem studenta do określonej grupy. Przechowuje historie przebiegu zajęć w grupy dla studenta. Zawiera obecności i oceny zdobyte podczas uczestnictwa w zajęciach.

**Klasa *SubjectAssociation***

Reprezentacja uczestnictwa studenta w dowolnych zajęciach z przedmiotu. Identyfikuje studenta z przedmiotem w przypadku, gdy przynależy do co najmniej jednej grupy w ramach tego przedmiotu. Zawiera wyliczone wyniki końcowe studenta uzyskane z przedmiotu to jest ocenę finalną, ocenę z egzaminu, ocenę z zajęć projektowych, laboratoryjnych lub ćwiczeniowych oraz odpowiadające im średnie, które przeliczane są z uwzględnieniem progów procentowych.

**Klasa *Role***

Przechowuje stałe identyfikujące dostępne role użytkowników w systemie.

**Klasy *GroupAssociationId*** **oraz** ***SubjectAssociationId***

Klasy pomocnicze w modelu reprezentujące klucz kompozytowy, to jest klucz złożony z więcej niż jednego elementu, który identyfikuje obiekt.

## Diagram klas odpowiadający komponentom

Cechą charakterystyczną komponentów jest ścisłość określania dostarczanych interfejsów oraz wymaganych zależności, co powoduje, że są one dobrymi, zamkniętymi i reużywalnymi jednostkami budowy systemów informatycznych. W procesie implementacji komponenty zostały zrealizowane poprzez uściślenie metodyki obiektowej. Pokazany poniżej diagram klas przedstawia elementy będące wyrazem pewnej grupy komponentów, które jako całość realizują operacje związane z przedmiotami. Lista oraz postać operacji reprezentowana jest przez interfejs komponentu logiki biznesowej o nazwie *ISubjectService*.



Wszystkie elementy intensywnie wykorzystują model domeny omówiony w poprzednim rozdziale i pokazanym na schemacie w postaci pakietu *duniv.model*. Na komponent realizujący operacje związane z przedmiotami składa się klasa *SubjectService*, która wystawia i realizuje interfejs *ISubjectService*. Klasa do swojego działania wymaga komponentów wystawiającyh interfejsy *ISubjectDAO* oraz *ISubjectAssociationDAO* oraz ich realizacji w postaci klas SubjectDAO oraz SubjectAssociationDAO.

## Diagram komponentów

Poprzedni model w postaci diagramu klas prezentuje elementy systemy w sposób bardzo dokładny. W celu pokazania ogólnej struktury elementów oraz zależności pomiędzy nimi skorzystano z diagramu komponentów również nalężącego do rodziny języka UML. Z punktu widzenia aplikacji najważniejsze do zaprojektowania są elementy warstwy logiki biznesowej oraz dostępu do danych. Warstwa prezentacji może zostać zbudowana z wykorzystaniem dowolnej technologii i z tego powodu proces modelowania tej części zostanie pominięty w pracy. Załączony poniżej diagram prezentuje podział odpowiedzialności pomiędzy komponentami logiki biznesowej oraz ich zależności od komponentów warstwy dostępu do danych.

## Diagram wdrożenia

Diagram wdrożenia służy do wizualizacji fizycznego rozmieszczenia elementów systemu. System podzielono zgodnie z architekturą trójwarstwową na trzy węzły, co prezentuje poniżej załączony schemat. Paczka wdrożeniowa „duniverity.war” zawiera logicznie pogrupowane komponenty oprogramowania będące pod kontrolą węzła serwera aplikacyjnego, co w połączeniu z takim fizycznym rozmieszczeniem odzwierciedla architekturę wielowarstwową.



# Wybór i opis technologii zastosowanych w projekcie

W celu pozytywnej realizacji projektu istotnym czynnikiem jest dobór odpowiednich technologii i bibliotek programistycznych. Błędna decyzja może skutkować niepowodzeniem lub brakiem możliwości dalszego rozwoju systemu.

Przy wyborze należy mieć na uwadze następujące kryteria:

1. Sposób licencjonowania

Należy uwzględnić czy zasady licencjonowania nie zostaną naruszone oraz czy wymagają dodatkowych nakładów finansowych.

1. Wielkość społeczności rozwijającej produkt

Ma duży wpływ na jakość produktu oraz długość okresu przez jaki będzie wspierany.

1. Popularność

Popularność również przyczynia się do wzrostu stabilności technologii.

W wraz z jej wzrostem rośnie dostępność materiałów, która przyspiesza i ułatwia rozwiązywanie często napotykanych problemów.

1. Zastosowanie

Trzeba pamiętać, że narzędzia mają określone przeznaczenie. Dobór powinien być dokonany z uwzględnieniem swoich celów i możliwości oferowanych przez wybierane biblioteki.

Po analizie powyższych czynników językiem, który zostanie wybrany do realizacji procesu implementacji będzie Java. Jest ona jedną z najpopularniejszych technologii, jeśli chodzi o rozwiązania typu enterprise. Wielkość społeczności zaangażowanej w proces jej udoskonalenia daje gwarancję przyszłego rozwoju projektu. Dodatkowo zastosowanie oferowanych dla niej bibliotek i narzędzi wspiera budowanie systemów wielowarstwowych.

## Platforma Java Enterprise Edition

Cały projekt został oparty na wycinku platformy Enterprise Edition dla języka javy. Dostarcza ona środowiska serwerowego do budowy systemów w architekturze wielowarstwowej. Platforma ma formę specyfikacji utrzymywanej przez grupy społeczności uczestniczących w programie Java Comunity Process. Na konkretną jej realizację wybrany został serwer Apache Tomcat w wersji siódmej. Jest on lekkim serwerem aplikacyjnym, który zapewnia środowisko wykonawcze dla różnego typu komponentów programowych realizujących logikę aplikacji.

Oprogramowania Apache Tomcat posłużyło w projekcie w roli „niepełnej” realizacji warstwy drugiej architektury trójwarstwowej. Poprzez niepełność należy rozumieć, że nie jest ona kompletnym rozwiązaniem, a środowiskiem dostarczającym podstawowych usług infrastruktury oprogramowania (np. obsługa protokołu http). Dzięki temu w trakcie tworzenia systemu skupiono się na modelowania zagadnienia, a środki do komunikacji pomiędzy „fizyczną” warstwą pierwszą i trzecią zostały dostarczone. Finalnie aplikacja tworzona na potrzeby tej pracy stanowi zbiór komponentów działających pod kontrolą tego serwera.

## Framework Spring

Framework Spring jest alternatywą jak i uzupełnieniem platformy Enterprise Edition. Ma budowę modułową i dostarcza szereg dodatkowych funkcjonalności. Wyznacza standard oraz sposób zarządzania pracą komponentów działających na serwerze aplikacyjnym. Framework został wykorzystany w procesie rozwarstwienia warstwy drugiej. W projekcie rozważano użycie innej technologii do tego celu, a mianowicie Enterprise Java Beans. Jednak obecny wybór uzasadniony był tym, ze Spring jest jednym z pierwszych i sprawdzonych rozwiązań tego typu, oferując ponad to bogate środowisko w zakresie uwierzytelnienia oraz zarządzania transakcjami. W sposób łatwy integruje się z innymi rozwiązaniami. W projekcie użyto jego głównej funkcjonalności, kontenera IoC, który pozwala na definiowanie, tworzenie i wiązanie komponentów. Dokładny sposób podziału został omówiony w rozdziale poświęconym implementacji.

## Biblioteki i narzędzia wspomagające

Decydującym kryterium wyboru narzędzi i bibliotek wspierających budowę systemu był sposób licencjonowania, brak dodatkowych kosztów z nimi związanych oraz ich zastosowanie. W projekcie wykorzystano wyłącznie rozwiązania typu open source. Rozdział przedstawia i pokrótce opisuje główne narzędzia i biblioteki mające zastosowanie w procesie implementacji.

**Apache Maven 3.2**

Maven jest narzędziem zarządzającym procesem budowy projektu. Do jego zadań należy ujednolicenie struktury projektu przez wprowadzenie archetypów, czyli wzorców określających kształt i proces budowy oprogramowania. Najbardziej użyteczną cechą jest oferowany system zarządzania zależnościami. Dzięki temu w łatwy sposób można określić lub z aktualizować zależności biblioteczne używane w projekcie, które zostaną automatycznie pobrane z publicznego repozytorium mavena.

**MySql 5.6**

Jest najbardziej powszechnym rozwiązaniem, jeśli chodzi o systemy zarządzania relacyjną bazą danych. Otwartość, brak dodatkowych kosztów i dobre narzędzia ułatwiające dostęp do bazy danych, takie jak „MySQL Workbench” przyczyniły się do wyboru tego rozwiązania.

**JSF 2.2**

Java Server Faces to technologia, która również wchodzi w skład platformy Enterprise Edition. Jest standardem budowy graficznego interfejsu użytkownika dla aplikacji biznesowych. Stanowi abstrakcję na klasyczny model komunikacji http i jest do swojego działania potrzebuje wyłącznie środowiska serwerowego wspierającego specyfikację serwletów, np. Apache Tomcat.

JSF ściśle przylega do architektury serwerowej przez zastosowanie komponentowego modelu interfejsu użytkownika, który przechowywany jest po stronie serwera. Technologia wprowadza swój autorski mechanizm szablonów dla budowy formularzy, które komunikują się w procesie żądania z programowymi komponentami na serwerze.

**Primefaces 5.0**

Primefaces to biblioteka dostarczająca rozszerzonych komponentów graficznego interfejsu użytkownika dla biblioteki JSF. Wzbogaca zbór standardowych elementów graficznych Java Server Faces.

**Hibernate 4.x**

Jest dostawcą specyfikacji JPA(Java Persistence Api), która stanowi standard mapowania obietkowo-relacyjnego dla javy. Ze względu ma wymieszanie koncepcji między relacyjnymi bazami danych i podejściem obiektowym potrzebny jest mechanizm odwzorowania jednego modelu na drugi. Z wykorzystaniem tej biblioteki nie ma potrzeby dokonywania tego w sposób manualny. Hibernate zapewnia możliwości oznaczenia klas określając tym samym sposób ich reprezentacji oraz składowania w relacyjnej bazie danych.

**Spring Security**

Moduł Spring Security to sprawdzone rozwiązanie w dziedzinie uwierzytelnienia i autoryzacji dostępu do aplikacji. Chroni przed nieautoryzowanym dostępem do danych na poszczególnych poziomach. W warstwie prezentacji zabezpiecza dostęp do linku, zaś w warstwie logiki biznesowej tylko użytkownicy z określonymi prawami mogą mieć dostęp do określonej funkcji. Spring Security dostarcza konfigurowalny model zabezpieczeń przez możliwość konfiguracji dostawcy uwierzytelnienia. Z zastosowaniem modułu spełnione zostało wyspecyfikowane wymaganie niefunkcjonalne NFR\_0: „Bezpieczeństwo danych”.

**Eclipse**

Jest z integrowanym środowiskiem programistycznym bazującym na modelu wtyczek, które rozszerzają jego możliwości. Dostawcy narzędzi i technologii informatycznych dbają o dostarczenie wtyczek umożliwiającą graficzną pracę z projektem w jednym miejscu. Wybór tego środowiska był również uzasadniony tym, że Eclipse wspiera platformę Enterprise Edition, framework Spring, oraz narzędzia do budowy projektu takie jak Maven.

**Git**

Powszechny, rozproszony system wersjonowania. Został wykorzystany do wersjonowania kodu źródłowego oraz dokumentów razem z prywatnym repozytorium GitLab.

# Opis implementacji

Implementacja odnosi się do procesu technicznej realizacji przedsięwzięcia programistycznego, popartego wcześniejszym planem i projektem. Ważnymi aspektami tego zagadnienia jest organizacja plików składających się na wykonywany projekt oraz sposób realizacji poszczególnych elementów.

## Struktura katalogów i plików projektu

Fizyczna zawartość projektu znajduje się w katalogu o nazwie „duniversity” i jest zbliżona do struktury klasycznego projektu na platformę Enterprise Edition z wykorzystaniem narzędzi Eclipse oraz Maven. W folderze można odnaleźć plik „pom.xml”, który zawiera konfigurację procesu budowy oraz zależności bibliotecznych.

Interfejsy i klasy odpowiadające komponentom lub modelowi domeny zostały pogrupowane z użyciem pakietów w celu uzyskania większego uporządkowania. Wszystkie pakiety znajdują się w katalogu „src” i z poziomu systemu mają postać zwykłych katalogów.

Ich znaczenie zostało przedstawione w postaci poniższej tabeli.

|  |  |
| --- | --- |
| Pełna nazwa pakietu | Opis zawartości |
| duniv.model | Zawiera klasy reprezentujące dziedzinę modelowanego problemu, przedstawione w rozdziale 3.1. |
| duniv.dao | Zawiera pliki interfejsów wystawianych przez komponenty warstwy dostępu do danych. |
| duniv.dao.impl | Zawiera pliki klas, które realizują wystawiane interfejsy z duniv.dao poprzez ich implementację, tworząc właściwe komponenty warstwy dostępu do danych. |
| duniv.service | Zawiera pliki interfejsów wystawianych przez komponenty warstwy logiki biznesowej. |
| duniv.service.impl | Zawiera pliki klas, które realizują wystawiane interfejsy z duniv.service poprzez ich implementację tworząc właściwe komponenty warstwy logiki biznesowej |
| duniv.view | Zawiera klasy reprezentujące zarejestrowane obiekty warstwy prezentacji technologii JSF |

Katalog „WebContent/WEB-INF” zawiera trzy pliki konfiguracyjne:

1. “faces-config.xml”

Plik konfiguracyjny biblioteki Java Server Faces.

1. “applicationContext.xml”

Plik konfiguracyjny dla frameworka Spring(moduły Core, Security). Zawiera deklarację podstawowych komponentów realizujących nisko poziomowy dostęp do bazy danych(warstwa danych), kontrolujących transakcyjność oraz uwierzytelnienie w dostępie do funkcji i zasobów.

1. “web.xml”

Plik konfiguracyjny całej aplikacji dla środowiska Java Enterprise Edition. Z jego pomocą zostały zarejestrowane biblioteki Spring i JSF.

Ostatnim istotnym katalogiem jest „WebContent/pages”, który zawiera strony widoków formularzy w technologii facelet JSF.

## Rozwarstwienie i zarządzanie komponentami

Definiowanie, wiązanie oraz grupowanie komponentów oprogramowania, które znajdują się na serwerze Apache Tomcat uzyskano z wykorzystaniem frameworka Spring.

W systemie większość komponentów funkcjonują pod kontrolą kontenera IoC(Inversion of Control). Jego nazwa odzwierciedla fakt, że zadanie tworzenia i zarządzania komponentami zostało przerzucone na bibliotekę. Kontrola obiektów jest jakby odwrócona w stosunku do klasycznego podejścia. W celu definiowania komponentów wykorzystano dwie oferowane przez framework możliwości.

Pierwsza to plik konfiguracyjny „applicationContext.xml”, który zawiera podstawowe deklaracje komponentów w postaci znaczników języka xml. Druga możliwość to użycie odpowiednich adnotacji języka Java w kodzie źródłowym na polach klas lub klasach odpowiadających komponentom.

Rozwarstwienie uzyskano poprzez użycie odpowiednich andotacji *@Repository* oraz *@Service* określanych mianem stereotypu w bibliotece Spring. Oznaczenie obiektu taką adnotacją jest jednoznaczne z deklaracją komponentu, jego utworzeniem i przypisaniem do określonej warstwy. Każdy komponent w projekcie składa się z co najmniej dwóch części, pierwsza to interfejs stanowiący kontrakt w procesie komunikacji pomiędzy innymi komponentami w różnych warstwach oraz druga, która jest realizacją wystawianego interfejsu. Komunikacja pomiędzy poszczególnymi warstwami oparta jest więc na interfejsach. Oczywiście liczba i funkcja warstw może być różna dzięki wsparciu podejścia aspektowego, jednak w projekcie skorzystano z domyślnych stereotypów.

W celu powiązania komponentów znajdujących się w różnych warstwach wykorzystano wzorzec wstrzykiwania zależności (dependency injection) oferowany przez framework. W kodzie źródłowym zadeklarowanych komponentów umieszczono referencję do interfejsów komponentów warstwy niższej, a dzięki specjalnej adnotacji *@Autowired* biblioteka automatycznie przeskanuje kontener komponentów i „wstrzyknie” odnaleziony pasujący do interfejsu komponent.

W dalszej części zostanie przedstawiona charakterystyka poszczególnych warstw zastosowanych w projekcie.

## Warstwa dostępu do danych

Warstwa ta składa się z komponentów, których zadaniem jest komunikacja z bazą danych.

Elementy w tej warstwie mają budowę opartą na wzorcu projektowym Data Access Object.

Z poziomu oferowanych funkcji dostęp realizowany jest na wyższym poziomie abstrakcji, poprzez udostępnianie metod, których argumentami są obiekty klas z modelu domeny przedstawionego w rozdziale 3.1. Liczba i nazwy komponentów również częściowo odpowiadają klasom z modelu domeny, ponieważ realizują operacje z nimi związane i to właśnie stanowiło główne kryterium podziału odpowiedzialności. W projekcie wyróżniono następujące interfejsy określające funkcje komponentów w warstwie dostępu do danych:

***ISubjectDAO***

Operacje bazodanowe związane z przedmiotami.

***IGroupDAO***

Operacje bazodanowe związane z grupami wykładowymi, projektowymi, ćwiczeniowymi lub laboratoryjnymi.

***IGroupAssociationDAO***

Operacje bazodanowe związane z obiektami reprezentującymi przynależność studenta do grupy.

***ISubjectAssociationDAO***

Operacje bazodanowe związane z obiektami reprezentujących uczestnictwo studenta w przedmiocie.

***IUserDAO***

Operacje bazodanowe związane z użytkownikami.

Poniższa tabela przedstawia interfejs, jego implementację oraz rejestrację dla komponentu *groupDAO*.

|  |  |
| --- | --- |
| **Interfejs komponentu** | **Implementacja i rejestracja komponentu** |
| **public** **interface** IGroupDAO {  **void** save(Group group);  **void** update(Group Group);  **void** saveOrUpdate(Group group);  **void** delete(Group group);  Group findById(Long id);  List<Group> findAll();  } | @Repository("groupDAO")  **public** **class** GroupDAO **implements** IGroupDAO {    @Autowired  SessionFactory sf;  @Override  **public** **void** save(Group entity) {  // some validation  sf.getCurrentSession().save(entity);  }  ...  } |

Adnotacja *@Repository* określa, że klasa *GroupDAO* implementująca interfejs *IGroupDAO* będzie zarejestrowana, jako komponent o nazwie *groupDAO* w warstwie dostępu do danych w kontenerze IoC.

Na konkretną komunikację z bazą danych wykorzystano mapowanie obiektowo relacyjne w postaci biblioteki hibernate. Każdy komponent wykorzystuje obiekt *sessionFactory* który służy do uzyskania sesji komunikacyjnej z właściwym systemem zarzadzania bazą danych i jest globalnie zadeklarowany w pliku „applicationContext.xml”. Mapowanie obiektowo relacyjne używane podczas każdej sesji zostało skonfigurowane przy pomocy adnotacji standardu Java Persistence Api na klasach modelu domeny.

Poniżej znajduje się fragment klasy *Subject* z mapowanej na strukturę bazodanową.

|  |
| --- |
| @Entity  @Table(name = "subjects")  public class Subject implements Serializable{  @ManyToOne  @JoinColumn(name = "lecturer\_id", nullable = false)  private User lecturer;  @Column(nullable = false)  private String title;  ...  } |

Adnotacja *@Entity* i *@Table* określa tutaj, że klasa *Subject* ma odpowiednik w bazie danych w postaci tabeli o nazwie *subjects*.

Warstwa dostępu do danych pełni też funkcję tłumaczenia wyjątków wyrzucanych przez różne technologie komunikacji z bazą danych. Niezależnie czy w projekcie zastosowano by klasyczny sposób, czy mapowanie obiektowo relacyjne z użyciem dowolnej biblioteki to wyjątki zgłaszane podczas komunikacji są przekształcane do jednakowej hierarchii wyjątków oferowanych przez framework Spring.

## Warstwa logiki biznesowej

Komponenty w tej grupie realizują operacje związane z wymaganiami funkcjonalnymi określonymi w rozdziale 2.2. Jest warstwą wyższą w stosunku do warstwy dostępu do danych. Przynależność do warstwy została zdefiniowana dzięki adnotacji *@Service*, która podobnie jak wyżej pozwala określić nazwę komponentu w kontenerze Spring. Poniżej został przedstawiony jeden z wielu występujących w projekcie komponentów w postaci interfejsu oraz jego implementacji, który realizuje operacje związane z grupami zajęciowymi.

|  |
| --- |
| **Interfejs komponentu realizującego operacje związane z grupami** |
| **public** **interface** IGroupService {    /\*  \* Dodanie grupy wykładowej dla przedmiotu z określeniem wykładowcy, ktróry jest  \* odpowiedzialny za przedmiot przedmiot  \*/  @PreAuthorize(“hasRole(‘LECTURER’)”)  **void** addLectureGroup(Group group, Subject subject, User lecturer);  /\* Edycja grupy wykładowej \*/  @PreAuthorize(“hasRole(‘LECTURER’)”)  **void** editLectureGroup(Group group);  /\* Dodanie grupy drugiej części zajęć w ramach przedmiotu z przypisaniem  \* asystenta do grupy \*/  @PreAuthorize(“hasRole(‘LECTURER’)”)  **void** addClassGroup(Group group, Subject subject, User assistant);  /\* Edycja wskazanej grupy z drugiej częśći zajęć.\*/  **void** editClassGroup(Group group);  /\* Usuwanie wskazanej grupy. \*/  @PreAuthorize(“hasRole(‘LECTURER’)”)  **void** removeGroup(Group group);    /\* Dodanie studenta do wskazanej grupy. \*/  **void** addStudentToGroup(User student, Group group);  } |

|  |
| --- |
| **Implementacja i rejestracja komponentu realizującego operacje związane z grupami** |
| @Service("groupService")  **public** **class** GroupService **implements** IGroupService {  @Autowired  IGroupDAO groupDAO;    @Autowired  IUserDAO userDAO;  @Autowired  IGroupAssociationDAO groupAssociationDAO;  @Override  @Transactional  **public** **void** addStudentToGroup(User student, Group group) {  **try** {  subjectService.addStudentToSubject(student, group.getSubject());  } **catch** (StudentAlreadyAddedToSubjectException e) {}  GroupAssociation ga = **new** GroupAssociation();  ga.setStudent(student);  ga.setGroup(group);  ga.setExcusesNumber(0);  groupAssociationDAO.save(ga);  }  . . .  } |

Komponent *groupService* do swojego działania wykorzystuje komponenty niższej warstwy wystawiające następujące interfejsy: *IUserDAO*, *IGroupAssociationDAO* i *IUserDAO*,co odpowiada zależnościom przedstawionym w rozdziale 3.3 w postaci diagramu komponentów.

Warstwa logiki biznesowej w systemie pełni też funkcję zarządzania transakcjami.

Framework Spring dostarcza abstrakcyjny model infrastruktury transakcji i pozwala na deklaratywne zarządzanie przy pomocy adnotacji. W projekcie skorzystano z jednego z menadżerów transakcji przygotowanego pod obsługę biblioteki hibernete.

Deklaracja tego menadżera razem z powiązaniem go z adnotacjami została również umieszczona w pliku „applicationContext.xml”, za co odpowiada jest poniższy fragment.

|  |
| --- |
| <tx:annotation-driven transaction-manager="txm" />  <bean id="txm"  class="org.springframework.orm.hibernate4.HibernateTransactionManager">  <property name="sessionFactory" ref="sf"></property>  </bean> |

Wszystkie metody i klasy w warstwie logiki biznesowej zostały oznaczone adnotacją *@Transactional,* co w połączeniu z powyższą konfiguracją pozwoliło na zamknięcie operacji różnych obiektów warstwy dostępu do danych w ramach pojedyńczej transakcji.

## Warstwa prezentacji

Warstwę prezentacji utworzono z wykorzystaniem technologii Java Server Faces opartej na dobrze znanym wzorcu MVC. Wprowadza ona swój kontener komponentów i metodę wstrzykiwania zależności, która jest podobna do tej realizowanej w bibliotece Spring.

Elementami tej warstwy w projekcie są:

* Strony formularzy w technologii facelet dla JSF
* Obiekty (managed beans), które są zarejestrowane i zarządzane przez bibliotekę JSF

W projekcie oznaczenie dowolnej klasy adnotacją *@ManagedBean* jest jednoznaczne z rejestracją komponentu przypisaniem go do warstwy prezentacji.

Z użyciem specjalnego języka „expression language” strony formularzy w trakcie przetwarzania przez serwer komunikują się z już zarejestrowanymi obiektami widoku poprzez odwoływanie się do ich metod oraz wartości. W celu osiągnięcia wzajemnej komunikacji pomiędzy komponentami widoku oraz logiki biznesowej w pliku konfiguracyjnym „faces-config.xml” wskazano klasę *SpringBeanFacesELResolver,* jako realizację przeszukiwania kontenera JSF. Teraz biblioteka JSF oprócz zbiornika z komponentami widoku również będzie przeszukiwać kontener z biblioteki Spring, zawierający komponenty logiki biznesowej i dostępu do danych. Czynność ta pozwoliła na integrację warstwy prezentacji z warstwą logiki biznesowej pomimo różnic technologicznych, w dalszym ciągu opierając się na koncepcji wstrzykiwania zależności.

Wszystkie komponenty warstwy prezentacji posiadają sufiks Controller i zostały skategoryzowane według funkcji związanej z treścią, prezentowaną przez fragmenty formularzy. Do głównych zadań tej warstwy należą:

* gromadzenie i walidacja danych pochodzących od użytkownika
* przygotowanie danych do wyświetlenia w formularzach prezentowanych na żądanie użytkownika
* inicjacja właściwych operacji dla aplikacji zawartych w komponentach logiki biznesowej z użyciem zebranych i z walidowanych danych

Poniższa lista przedstawia najważniejsze komponenty oraz odpowiadająca im treść formularzy.

|  |  |
| --- | --- |
| Nazwa komponentu | Powiązana treść formularzy |
| UserController | użytkownicy systemu(administratorzy, wykładowcy, asystenci, studenci) |
| SubjectController | przedmioty użytkownika |
| MarksController | oceny z zajęć, egzaminu i przedmiotu |
| LectureGroupsController | grupy wykładowe |
| LectureAttendancesController | obecności grup wykładowych |
| ClassGroupsController | grupy inne niż wykładowe |
| ClassAttendancesController | obecności grup innych niż wykładowe |

## Uwierzytelnienie i autoryzacja dostępu

Proces uwierzytelnienia oraz autoryzacji utworzono z użyciem elastycznego modułu Spring Security. Konfiguracja zabezpieczeń znajduje się w pliku „applicationContext.xml”.

W dalszej części zostanie omówiony sposób realizacji modelu uprawnień i zabezpieczenia aplikacji.

### Konfiguracja modelu uprawnień

|  |
| --- |
| <authentication-manager>  <authentication-provider>  <jdbc-user-service data-source-ref="dataSource"  users-by-username-query="SELECT username, password, enabled  FROM users  WHERE username = ?"  authorities-by-username-query="SELECT username , rolename  FROM users  WHERE username = ?" />  </authentication-provider>  </authentication-manager> |

W pierwszym kroku skonfigurowano model uprawnień używanych w projekcie i aplikacji. Model uprawnień określa miejsce oraz sposób składowania nazw, haseł i uprawnień użytkowników. Poniższy fragment odpowiada konfiguracji tego modelu w projekcie.

By określić model uprawnień wykorzystano jedną z oferowanych przez moduł postaci, a mianowicie parametryzowany SQL. Znacznik *jdbc-user-service* mówi, że informację na temat użytkowników oraz ich uprawnień będą pozyskiwane z użyciem języka SQL oraz standardowego interfejsu dostępu do bazy danych JDBC dla javy. Znacznik posiada trzy atrybuty: *data-source-ref*, *users-by-username-query* oraz *authorities-by-username-query*.

Pierwszy wskazuje na obiekt źródła danych, również skonfigurowany w pliku. Jest odpowiedzialny za właściwe nawiązanie połączenia z warstwą danych(MySQL).

Drugi zawiera wyrażenie SQL, które pozwala wyszukać hasło i status aktywności użytkownika w systemie po parametrze określającym nazwę użytkownika.

Trzeci atrybut zawiera wyrażenie SQL, które pozwala wyszukać wszystkie role użytkownika po parametrze reprezentującym jego nazwie.

W zależności od postaci tych zapytań model uprawnień może być różny. W systemie stworzono model uprawnień, w którym rola użytkownika jest jednoznaczna z jego uprawnieniem. W dodatku każdemu użytkownikowi przyporządkowana jest tylko i wyłącznie jedna rola, która przechowywana jest w tym samym miejscu, co jego nazwa, hasło oraz status aktywności. Taka ziarnistość w zupełności wystarcza by spełnić wymagania stawiane przed systemem.

### Zabezpieczenie aplikacji

Posiadając skonfigurowany model uprawnień, zabezpieczenia aplikacji dokonano na różnych poziomach jej funkcjonowania:

1. **Dostęp do zasobów przez protokół http (warstwa prezentacji)**

Klient uzyskuje dostęp do aplikacji poprzez żądania pobrania formularzy przy użyciu protokołu http. Poniższy wpis konfiguracyjny posłużył do zabezpieczenia przed niepowołanym dostępem do określonych zasobów.

|  |
| --- |
| <http use-expressions="true">  <form-login login-page="/login.jsp" />  <intercept-url pattern="/login.jsp" access="permitAll" />  <intercept-url pattern="/\*\*" access="isAuthenticated()" />  </http> |

Konfiguracja przy pomocy znacznika „form-login” określa domyślny formularz logowania, który będzie wyświetlony podczas próby pierwszego dostępu do zasobu w celu weryfikacji danych użytkownika względem określonego wcześniej modelu uprawnień. Znacznik *intercept-url* przy użyciu wyrażenia regularnego określa wymagane uprawnienia do grup zasobów. Dla formularza logowania „login.jsp” ustawiono dostęp publiczny. Dla całej reszty zasobów wymagane jest poprawne przejście uwierzytelnienia z domyślnym formularzem logowania. Sprawdzaniem czy żądania w ramach sesji zostały już uwierzytelnione zajmuje się biblioteka Spring.

1. **Autoryzacja dostępu do funkcji komponentów logiki biznesowej**

**(warstwa logiki biznesowej)**

Przy pomocy adnotacji *@PreAuthorize* oferowanej przez Spring Security, zabezpieczono dostęp do funkcji komponentów logiki biznesowej. Poniżej znajduje się fragment kodu przedstawiający część interfejsu komponentu odpowiedzialnego za wykonywanie operacji związanych z grupami studenckimi.

|  |
| --- |
| public interface IGroupService {  @PreAuthorize(“hasRole(‘LECTURER’)”)  void editLectureGroup(Group group);    ...  } |

Uprawnienia dostępu do funkcji *editLectureGroup* realizującej edycję grupy wykładowej uzyskać może tylko użytkownik posiadający rolę wykładowcy.

# Testowanie

W dzisiejszych czasach, jakość oferowanych rozwiązań informatycznych jest jednym z głównych czynników sukcesu. Testowanie odgrywa istotną rolę w osiągnięciu wysokiej jakości tworzonego oprogramowania. Praktycznie żadne przedsięwzięcie projektowe nie może obyć się bez tej czynności. Rozdział ten opisuje proces testowania zastosowany do tworzonego na potrzeby pracy systemu.

## Testy jednostkowe

Testowanie jednostkowe dotyczy testowania poszczególnych elementów systemu w ścisłej izolacji. Powodem takiego podejścia jest fakt, że w przypadku wykrycia błędu jest on w szybki sposób lokalizowany i naprawiany. Dodatkowo ciągła zautomatyzowana weryfikacja zgodności oczekiwanego rezultatu podczas pracy z rozwijanym kodem źródłowym daje większą gwarancję jakości. Testy jednostkowe muszą być przede wszystkim szybkie w swoim działaniu ze względu na powtarzający się cykl ich przeprowadzania. W metodyce obiektowej jednostką oprogramowania może być poszczególna metoda lub klasa, która implementuje pewne operacje na różnym poziomie abstrakcji,[5].

W projekcie wykorzystano bibliotekę „junit”, będącą de facto standardem, jeśli chodzi o przeprowadzanie testów automatycznych i jednostkowych. Testom poddano pojedyncze elementy w warstwie logiki biznesowej.

By osiągnąć niezależność pomiędzy komponentami skorzystano również z możliwości oferowanej przez bibliotekę mockito, która pozwala na szybkie tworzenie sztucznych obiektów, tak zwanych „mocków”.

Sposób testowania warstwy logiki biznesowej polegał na spreparowaniu obiektów reprezentujących zależne komponenty z warstwy dostępu do danych, w taki sposób by odzwierciedlały możliwe przypadki ich działania oraz na wielokrotnym wywoływaniu testowanej funkcji z przypuszczalnymi argumentami mając na uwadze ich warunki brzegowe i oczekiwany rezultat.

Poniżej zamieszczony został przypadek testowy w postaci klasy testującej metodę dodawania nowej grupy wykładowej. Adnotacja *@Mock* tworzy sztuczny obiekt komponentu na podstawie interfejsu *IGroupDAO*. Przy pomocy specjalnych wyrażeń z biblioteki mockito określono możliwe scenariusze działania testowanej funkcji.

* Pierwszy uwzględnia poprawne utworzenie i zapisanie grupy z użyciem sztucznego obiektu *groupDAO*
* Drugi oczekuje wystąpienia zdarzenia wyjątkowego podczas dodania grupy w przypadku, gdy już istnieje lub zostaną naruszone więzy integralności w dostępie do zasobów danych
* Trzeci oczekuje zdarzenia wyjątkowego w przypadku pominięcia dowolnego argumentu w wywołaniu funkcji

|  |
| --- |
| @RunWith(MockitoJUnitRunner.**class**)  **public** **class** GroupServiceTest {  @Mock  **private** IGroupDAO groupDAO;    @InjectMocks  **private** GroupService groupService;    **private** Group newGroup;  **private** Subject subject;  **private** User lecturer;  @Before  **public** **void** init() { /\* create mock newGroup, subject, lecturer \*/ }    @Test  **public** **void** testTryingAddLectureGroup() {  **try** {  groupService.addLectureGroup(newGroup, subject, lecturer);  } **catch** (Exception e) {  *fail*("Failed to add lecture Group!");  }  }    @Test(expected = LectureGroupAlreadyExistException.**class**)  **public** **void** testTryingAddLectureGroupWhenAlreadyExist() {  *doThrow*(**new** DataIntegrityViolationException("")).when(groupDAO).save(newGroup);  groupService.addLectureGroup(newGroup, subject, lecturer);  }    @Test(expected = IllegalArgumentException.**class**)  **public** **void** testTryingAddLectureGroupFailsWithInvalidArguments() {  groupService.addLectureGroup(**null**, subject, lecturer);  groupService.addLectureGroup(newGroup, **null**, lecturer);  groupService.addLectureGroup(newGroup, subject, **null**);  }  } |

## Testy integracyjne

Ten typ testowania ma na celu sprawdzenie jakości wzajemnie współdziałających komponentów oprogramowania, które składają się na większy fragment systemu. W przeciwieństwie do testów jednostkowych ich obecność jest opcjonalna, a proces przeprowadzania znacznie rzadszy, co powoduje, że nie są nałożone na nie żadne wymogi związane z typem oraz czasem wykonywanych operacji.

Komponenty warstwy dostępu do danych zostały przetestowane z użyciem testów integracyjnych ze względu na ich charakter oraz realizowane funkcje. Fragmentem systemu, który będzie uczestniczył również w tym procesie jest konkretny system zarządzania bazą danych.

Do realizacji czynności w dalszym ciągu wykorzystano bibliotekę junit, jako sposób automatyzacji. W celu zachowania nienaruszalności pierwotnej struktury bazodanowej skorzystano z dodatkowego silnika bazy danych HSQLDB. Jego cechą charakterystyczną jest możliwość uruchomienia wszystkich elementów w pamięci operacyjnej na żądanie użytkownika.

Poniżej znajduje się fragment kodu, przypadku testującego metodę zapisu nowego użytkownika do testowej bazy HSQLDB. Andotacja *@ContextConfiguration* wskazuje położenie pliku konfiguracyjnego dla testowego kontenera komponentów Spring. Znajdują się w nim ustawienia testowej bazy danych oraz deklaracje testowanych komponentów.

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations = {"classpath:testAppCtx.xml"})  @Transactional  **public** **class** UserDAOTest {  @Autowired  IUserDAO userDAO;    @Test  **public** **void** testSaveUser() {  User user = **new** User();  user.setUsername("mnowic");  user.setPassword("mnowic");  user.setFirstName("Marek");  user.setLastName("Nowicki");  user.setRolename(Role.*LECTURER*);    *assertNull*(user.getId());  userDAO.save(user);  *assertNotNull*(user.getId());  *assertTrue*(user.getId() > 0);  }  } |

## Testy graficznego interfejsu użytkownika

Ostatnią grupą testów przeprowadzonych na potrzeby tworzonej aplikacji są testy graficznego interfejsu użytkownika. Na podstawie opinii potencjalnych odbiorców zbadano jakość w procesie graficznej komunikacji pomiędzy użytkownikami oraz systemem. Wymaganie niefunkcjonalne „NFR\_0: Łatwość użytkowania” określa, że system ma być na tyle przejrzysty, by nie sprawiał większego problemu z jego obsługą. Dobrze zaprojektowany interfejs użytkownika pozwoli na automatyczną i intuicyjną naukę korzystania z systemu.

Weryfikacja polegała na zaproszeniu grupy osób w różnym wieku i zrealizowaniu kilku czynności przedstawionych w rozdziale związanym z wymaganiami funkcjonalnymi systemu.

W testach brały udział: dwie osoby w roli wykładowców, cztery osoby w roli asystentów oraz dziesięć osób w przedziale wiekowym 20 - 25 w roli studentów. Całość została rozbita na dwa przedmioty prowadzone przez dwóch innych wykładowców uczestniczących w badaniu. Reszta asystentów i studentów uczestniczyła w obydwu przedmiotach poprzez ich odpowiednie przypisanie, zgodne z działaniem aplikacji.

Użytkowanie systemu oraz wykonywanie określonych operacji nie stanowiło żadnej trudności. Osobom testującym bardzo odpowiadał zarówno kolor jak kształt poszczególnych elementów interfejsu użytkownika. Wskazywali oni zalety zastosowania intuicyjnych grafik, dzięki którym odnalezienie poszczególnych funkcji systemu było naprawdę proste i szybkie.

Na podstawie ich opinii stwierdzono, ze wynik testu okazał się więc pozytywny.

# Prezentacja i opis działania systemu

W tym rozdziale zostało zaprezentowane działanie systemu. Poniżej pokazano najważniejsze oferowane funkcje w formie graficznej i opisowej. Niezależnie od posiadanej przez użytkownika roli uzyskuje on dostęp do zasobów systemu tylko w przypadku pozytywnego zakończenia procesu uwierzytelnienia.

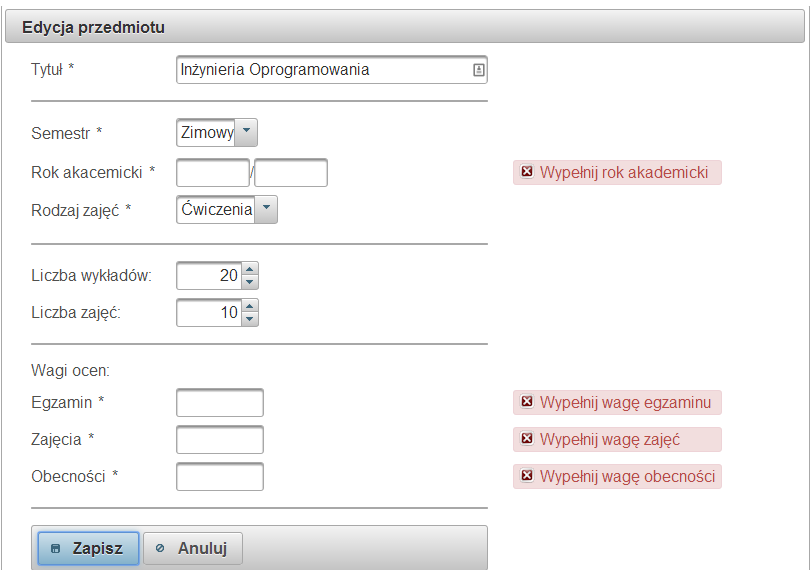
Po zalogowaniu do systemu każdy użytkownik otrzymuje odpowiednią listę przygotowanych dla niego przedmiotów. Wykładowca zobaczy przedmioty, które zdefiniował w systemie, asystent tylko te, w których jest przypisany jako prowadzący jakiejkolwiek grupy, student zaś tylko te w których uczestniczy, poprzez obecność w dowolnej grupie dla przedmiotu. Poniżej zaprezentowany jest widok dla wykładowcy, który posiada największe uprawnienia w stosunku do swoich przedmiotów.

W ostatniej kolumnie każdego wiersza znajdują się kolejno przyciski umożliwiające: podgląd szczegółów przedmiotu, uzyskanie informacji na temat sposobu wyliczenia oceny końcowej, edycję lub podgląd skali ocen, edycję przedmiotu oraz jego usunięcie.

Rys. 1

Lista przedmiotów dla użytkownika

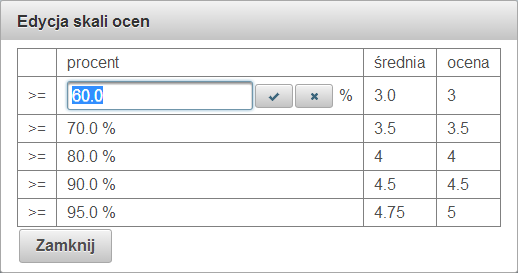
Tylko dla wykładowcy jest dostępny przycisk „Dodaj”, znajdują się w dole ekranu na liście przedmiotów. Z jego pomocą może zdefiniować nowy, prowadzony przez siebie przedmiot. Podczas tego działania określa okres trwania przedmiotu, typ drugiej części zajęć(laboratoria, projekt, ćwiczenia), liczbę wykładów, liczbę zajęć dla reszty grup oraz wagi poszczególnych ocen cząstkowych. Całość przedstawia następujący formularz.



Rys. 2

Definiowanie nowego przedmiotu

Kolejną czynnością możliwą dla wykładowcy jest określenie skali ocen dla przedmiotu oraz grup zajęciowych. Na poniższym formularzu zaprezentowana jest edycja dolnej granicy procentowego progu dla oceny 3.0. Jej górną granicą jest dolny próg procentowy kolejnej oceny.

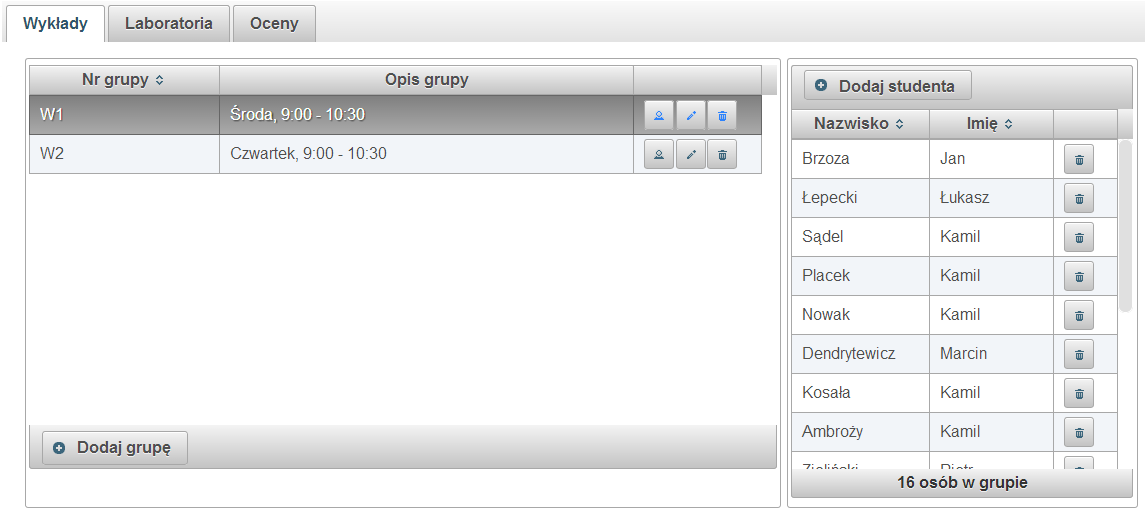


Rys. 3

Edycja skali ocen

Dla każdego przedmiotu w systemie można uzyskać szczegółowe dane na jego temat przy pomocy pokazanego przycisku z symbolem strzałki na rys. 1. Formularz podzielony jest na trzy części w postaci zakładek.

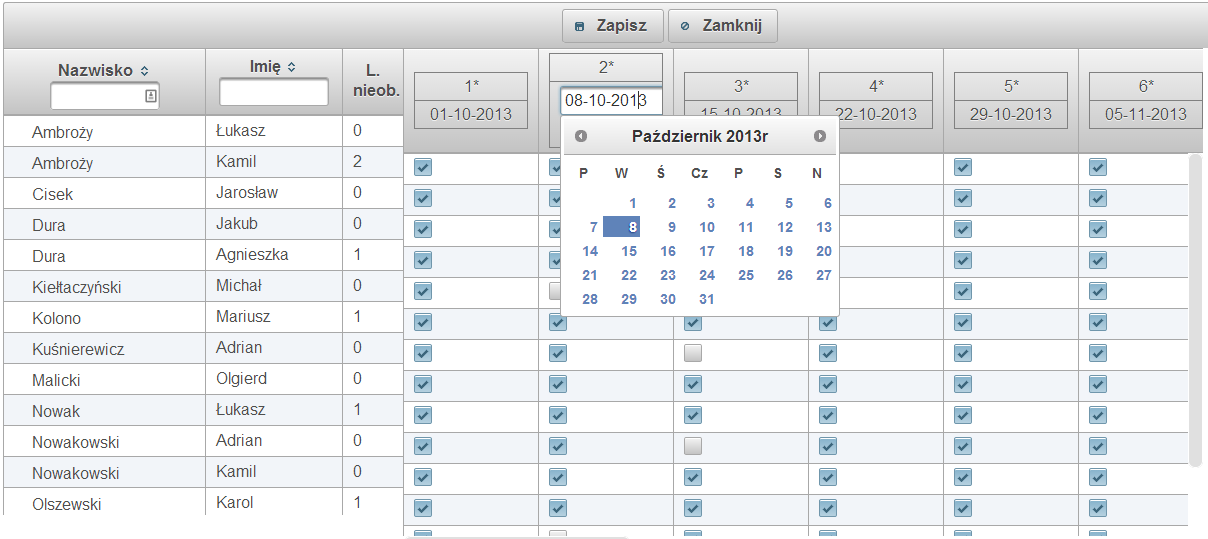
Pierwsza dotyczy informacji na temat grup wykładowych. Zakładka jest dostępna tylko dla wykładowcy oraz studenta. Ten drugi widzi wyłącznie swoją grupę audytoryjną bez żadnych dodatkowych informacji na temat innych studentów w niej uczestniczących.

Na formularzu widać przyciski „Dodaj grupę” oraz „Dodaj studenta”, które pozwalają odpowiednio na dodanie nowej grupy wykładowej oraz na przypisanie studenta do zaznaczonej grupy. W wierszu każdej grupy znajdują się przyciski umożliwiające kolejno: podgląd lub edycje obecności, edycję oraz usunięcie grupy.

Rys. 4

Podgląd szczegółów przedmiotu

Poniższy formularz przedstawia edycję obecności dla grupy wykładowej i jest on analogiczny dla innych rodzajów grup. Podgląd dla studenta jest w takiej samej formie za wyjątkiem braku możliwości edycji i widoczności tylko swojego rekordu.



Rys. 5

Edycja obecności dla grup

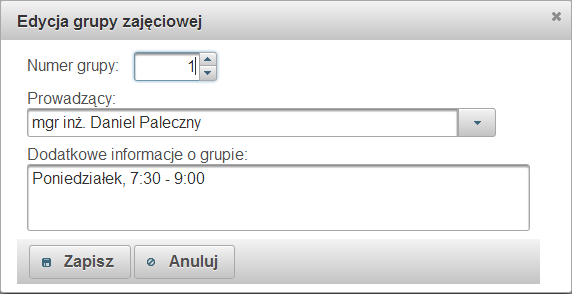
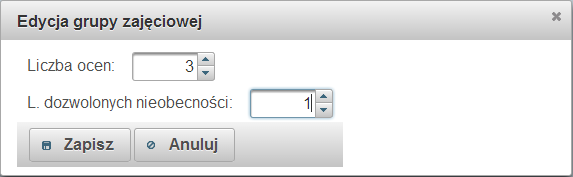
Kolejna zakładka dotyczy drugiej części zajęć określonych przy dodawaniu przedmiotu – projekt, laboratoria lub ćwiczenia. Jest ona zbliżona funkcjonalnością i sposobem prezentacji do poprzedniej. Jednak w tym przypadku oferowane czynności przeznaczone są głównie dla asystenta, który prowadzi poszczególne grupy. Wykładowca dodając grupę określa asystenta odpowiedzialnego za przebieg zajęć. Asystent po zalogowaniu widzi przedmiot z przypisaną grupą, a w widoku szczegółów może również edytować swoją grupę zajęciową, jednak w tym przypadku opcje zmiany będą inne.

Na poniższych rysunkach widać edycję grupy drugiej części zajęć dla wykładowcy oraz asystenta. Ten drugi może określić liczbę ocen oraz liczbę dozwolonych nieobecności w semestrze, po których przekroczeniu student nie otrzymuje zaliczenia z tej części zajęć.

Po lewej stronie widać dialog edycji grupy zajęciowej dla wykładowcy, po prawej dla asystenta.

Rys. 6

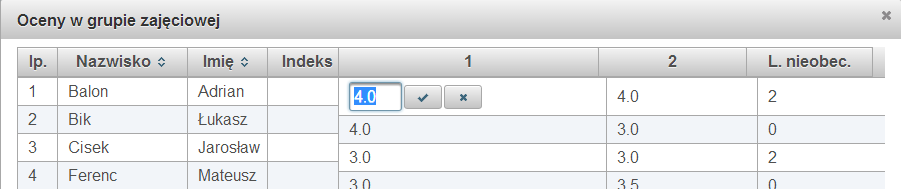
Edycja grupy - wykładowca



Rys. 7

Edycja grupy - asystent

W stosunku do poprzedniej zakładki w wierszu grupy dochodzi przycisk, który umożliwia podgląd ocen dla studenta i wykładowcy oraz ich edycję w przypadku asystenta, co prezentuje poniższy formularz.



Rys. 8

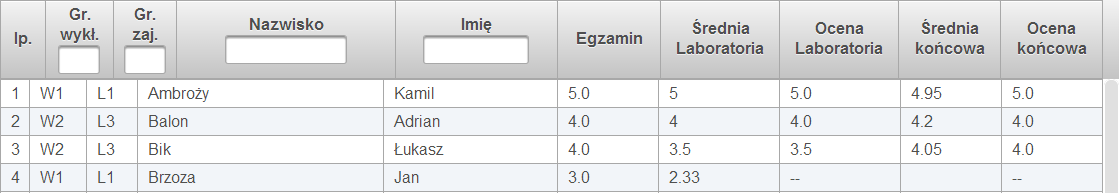
Podgląd i edycja ocen dla grup projektowych, laboratoryjnych i ćwiczeniowych

Ostatnia zakładka prezentuje wszystkie oceny cząstkowe i końcowe dla przedmiotu.

Wykładowca, asystent i student widzą odpowiednią dla siebie liczbę wpisów i kolumn.

Prowadzący przedmiot ma możliwość edycji oceny z egzaminu, zaś reszta użytkowników

wyłącznie podglądu wyliczonych przez system ocen.



Rysunek 2 Fragment widoku ocen dla wykładowcy

# Wnioski

Celem pracy było stworzenie systemu w architekturze wielowarstwowej, który będzie wspomagać prowadzenie zajęć na uczelni wyższej. Wszystkie założenia stawiane w celach pracy oraz w wymaganiach projektowych zostały zrealizowane. Udało się zbudować system wykorzystujący wszystkie właściwości tej architektury.

W procesie implementacji systemu zauważono, że taka architektura wprowadza dodatkowy nakład pracy w postaci realizacji każdej warstwy. Sposób budowy jej elementów z użyciem narzędzi informatycznych wykorzystuje niestandardowe podejście i może stanowić trudność w procesie poznawczym. Potrzeba zmiany komponentów staje się początkowo procesem żmudnym przez jednoczesne utrzymanie interfejsów oraz klas ich realizujących. Rozbicie i określenie kształtu komponentów nie jest też procesem ustandaryzowanym. Jednak w wraz z rozrostem projektu, cały ten wysiłek się opłaca. Do największych zauważalnych zalet należy sam fakt dekompozycji pewnych zagadnień. Już w przypadku takiego projektu ciężko było by go spójnie rozwijać i panować nad całością. Wielokrotne użycie tych komponentów w warstwie dostępu do danych znacząco ułatwiła i skróciła czas wykonania pewnych operacji. Sposób grupowania i wiązania komponentów z użyciem wstrzykiwania zależności staje się wręcz wymogiem stosowania w stosunku do klasycznego podejścia. Odnalezienie definicji komponentów oraz ich zależności po opanowaniu tego sposobu staje się przejrzyste i proste. Dojść można również do wniosku, że model takiej architektury zbliża się do programowania proceduralnego z powodu wykorzystania modelu domeny, zawierającego bardzo niewielką ilość logiki biznesowej, a służącego głównie jako zbiornik dla danych przetwarzanych przez właściwe komponenty logiki biznesowej oraz dostępu do danych. Architektura przede wszystkim zmniejsza podatność na błędy i zwiększa poziom zabezpieczeń poprzez testowanie komponentów i wprowadzenie mechanizmów zabezpieczeń i walidacji danych na poziomie kilku warstw.

Sam system może zostać wdrożony w wielu jednostkach dydaktycznych i z pewnością ułatwi proces prowadzenia spójnej historii przebiegu zajęć w postaci listy ocen oraz obecności. Uwolni asystentów od okresowego wysyłania wyników swoich podopiecznych prowadzącym. System ma możliwość dalszego rozwoju i jako jedną z nich jest integracja z innymi modułami oprogramowania przeznaczonych dla uczelni wyższych np. moduły EHMS stosowane między innymi na Politechnice Krakowskiej. Integracja z prawdziwą bazą studentów i pracowników dydaktycznych oraz dostęp do edycji informacji związanych z przebiegiem studiów przechowywanych i zarządzanych przez taki moduł znacząco usprawnia również pracę dziekanatu.

# Bibliografia

1. Heiko Schuldt,

„Multi-Tier Architecture”

1. Andrzej Sikorski,

„Rozszerzone architektury wielowarstwowe w środowisku komunikacyjnym aplikacji rozproszonych”

1. Buschmann Frank; Meunier Regine; Rohnert Hans; Sommerlad Peter; Stal Michael,

“Pattern-Oriented Software Architecture”, Volume 1

1. Bernd Bruegge, Allen H. Dutoit,

“Object-Oriented Software Engineering Using UML, Patterns, Java”

1. Kshirasagar Naik, Priyadarshi Tripathy

“Software Testing and Quality Assurance”

1. Dokumentacja Tomcat,

<http://tomcat.apache.org/>

1. Dokumentacja Spring,

<http://projects.spring.io/spring-framework/>

1. Dokumentacja Primefaces, http://www.primefaces.org/docs/guide/primefaces\_user\_guide\_5\_0.pdf
2. Dokumentacja Hibernate,

http://hibernate.org/orm/documentation/

1. Zbiór artykułów,

http://www.mkyong.com/