**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

# *Serwis internetowy dla studentów zaimplementowany w architekturze SOA*

# *Internet service for students implememented in SOA architecture*

Autor: *Marek Ryznar*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Mirosław Gajer*

Kraków, 2015

*Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*

*Marek Ryznar*

*Serdecznie dziękuję mojemu promotorowi za wsparcie merytoryczne.*
*Oraz narzeczonej za wsparcie mentalne.*

# Spis treści

# 1. Wprowadzenie

W dobie współczesnej technologii i internetu, wymiana notatek z zajęć dydaktycznych oraz przekazywanie sobie wzajemnie przez studentów informacji dotyczących odwołanych zajęć bądź też informacji o kolokwiach i egzaminach, nie stanowi problemu. Najbardziej znanym sposobem na wymianę plików jest wysyłanie za pomocą wiadomości email. Często niestety bywa tak, że załączone materiały przekraczają dopuszczalny rozmiar załącznika, wtedy z pomocą przychodzą dyski internetowe, które pozwalają na przechowywanie większych plików. Najbardziej znane wśród studentów to google drive, dropbox i One drive.

Wszystkie wymienione wyżej rozwiązania są często stosowane przez studentów, aczkolwiek trudno znaleść odpowiedni serwis dedykowany bezpośrednio dla nich. Celem tej pracy jest implementacja serwisu internetowego pomagającego w zarządzaniu materiałami dydaktycznymi z zajęć. Głównym przeznaczeniem tworzonego portalu jest dodawanie plików oraz udostępnianie ich innym użytkownikom. Dodatkową planowaną funkcjonalnością, która może pomóc studentowi w organizacji roku akademickiego jest przeznaczony dla niego kalendarz, w którym użytkownik mógłby zapisywać wszystkie ważne wydarzenia takie jak kolokwia lub egzaminy. Kolejną funkcją systemu, która ma na celu ułatwienie studentowi zarządzania kontem serwisu jest strona powiadomień na której znajdować się będą przypomnienia o zbliżających się wydarzeniach, informacje o udostępnionych dla niego plikach oraz informacje dotyczące uczelni.

Niniejsza praca opisuje proces powstawania przedstawionego serwisu. W pierwszym rozdziale czytelnik zostanie zaznajomiony z najważniejszymi pojęciami dotyczącymi opisywanego projektu. W kolejnym rozdziale znajduje się koncepcja systemu przedstawiona za pomocą dokładnie opisanych (słownie oraz za pomocą diagramów czynności oraz przypadków użycia) zaplanowanych funkcji systemu. W czwartym rozdziale za pomocą modelu bazy danych oraz modelu klas przedstawiony jest projekt systemu. Na samym końcu opisane są najważniejsze szczegóły implementacyjne projektu.

# 2. Podstawy teoretyczne

Rozdział ten przedstawia podstawy teoretyczne projektu. W pierwszym podrozdziale znajduje się krótkie wprowadzenie do architektóry SOA, na której oparty jest serwis. W następnym opisany jest język implementacji. Trzeci podrozdział prezentuje formę przechowywania danych w serwisie. Ostatni punkt związany jest z przeznaczeniem serwerów aplikacyjnych.

## 2.1. SOA

Rozwinięciem skrótu SOA jest *Service Oriented Architecture* czyli architektura zorientowana na usługi. Według książki *SOA Design Principles for Dummies* napisanej przez Claus'a T. Jensen'a SOA składa się z trzech podstawowych aspektów:

- **Usługa** - czyli powtarzalne zadanie biznesowe np. Dodawanie plików na serwer lub tworzenie nowego konta,
- **Orientacja na usługi** - sposób postrzegania produktu jako zbiór usług połączonych w całość,
- **SOA** - podejście architektoniczne bazujące na zasadach orientacji na usługi.

Architektura zorientowana na usługi posiada następujące właściwości:

- **Luźne powiązanie usług,**
- **Abstrakcyjne usługi,**
- **Autonomiczność usług,**
- **Możliwość wielokrotnego użycia usług,**
- **Bezstanowość usług,**
- **Dzielenie usług na komponenty** [1].

SOA jest architekturą wspierającą modularność więc produkty tworzone w tej architekturze łatwo jest rozwijać poprzez dodawanie nowych usług.

## 2.2. Java

Język programowania Java został stworzony na początku lat dziewięćdziesiątych przez grupę inżynierów z firmy Sun zwaną "Green Team". Ideą tworzonego języka była praca w rozwijającym się środowisku internetu [2]. Java została uformowana na podstawie języka C++, przy czym wymusza ona programowanie obiektowe. Dzięki swojej funkcjonalności oraz wszechstronnemu zastosowaniu Java szybko zdobyła popularność wśród programistów.

Java dzięki swojej platformie programistycznej Java EE (Java Enterprise Edition) pozwala pisać multi platformowe, wielowarstwowe, niezawodne i bezpieczne aplikacje sieciowe. Dzięki wielu darmowo udostępnianym bibliotekom i frameworkom wielu programistów decyduje się na wybranie tej właśnie platformy [3].

Główne frameworki platformy Java EE pomagające w tworzeniu opisywanej aplikacji to:

– **JSF** (ang. Java Server Faces) - umożliwia łatwiejsze tworzenie internetowego interfejsu użytkownika, dzięki udostępnianym tagom używanych przy kreacji stron internetowych. Posiada także ManagedBeany czyli klasy odpowiedzialne za połączenie logiki aplikacji z widokiem (stronami internetowymi),

– **Hibernate** - biblioteka dostarczającą framework do realizacji połączenia z bazą danych. Umożliwia generowanie klas Java na podstawie tabel z bazy danych i na odwrót. Dzięki Hibernate dostęp do bazy danych jest realizowany za pomocą wysokopoziomowych metod, które operują na zmapowanych wcześniej obiektach [4].

## 2.3. Przechowywanie danych

Wirtualne dane można przechowywać na wiele sposobów, najprostszym z nich jest przechowywanie danych lokalnie czyli na dysku twardym stacji roboczej, aczkolwiek w tym wypadku istnieje zagrożenie utraty tych danych w razie awarii komputera. W obawie przed utratą danych użytkownicy często szukają sposobu na umieszczenie ich zapasowej kopii w internecie. Istnieje wiele sposobów na przechowywanie naszych plików w sieci:

– **Dyski sieciowe** - jest to typ urządzenia przechowującego dane, które zapewnia dostęp do nich w lokalnej sieci LAN,

– **Dyski internetowe** - umożliwiają przechowywanie plików w w sieci internetowej m.in. za pomocą chmur,

– **Bazy danych** - z tego rozwiązania korzysta się raczej w przypadku ustrukturyzowanych danych np. Informacje o użytkownikach (login użytkownika serwisu w postaci krótkiego pola tekstowego lub rok jego urodzenia w postaci liczby całkowitej). Pomimo tego niektóre bazy danych udostępniają

możliwość przechowywania danych w postaci binarniej czyli bez podania ich typu co pozwala na zapisywanie danych bez wymuszonego ich typu.

Do implementacji niniejszej aplikacji użyto bazy danych. Wybór był podyktowany tym, że platforma Java EE udostępnia framework Hibernate, który pozwala na prosty dostęp do bazy danych. Wybrany system do zarządzania relacyjną bazą danych to PostgreSQL, system ten jest wolnodostępny.

Do przechowywania danych o użytkownikach i ich kalendarzach wystarczą podstawowe typy danych dostępne we większości sytemów zarządzających bazą danych. Natomiast w przypadku plików o większych rozmiarach, np. całych książek zapisanych w dowolnym formacje (pdf, doc. i inne) zastosowany zostanie typ danych BYTEA, który zajmuje się przechowywaniem plików binarnych.

## 2.4. Serwer Aplikacyjny

Serwer aplikacyjny jest kontenerem webowym, który odpowiada za przechowywanie, zdalne uruchamianie i użytkowanie aplikacji Java EE. Ponadto pomaga programiście tworzyć aplikacje oraz umożliwia oddzielenie logiki biznesowej od usług dostarczanych przez dany serwer aplikacyjny np. zabezpieczenia aplikacji [5]. Najbardziej znane serwery aplikacyjne to JBoss, IBM WebSphere, Apache Tomcat, GlassFish. Serwer używany do implementacji opisywanej aplikacji to JBoss AS 7.1.1. Serwer ten został wybrany ponieważ jest darmowy (udostępniany na licencji *GNU Lesser General Public License*), ponadto implementuje pełen zestaw usług JEE. Dodatkowym argumentem przemawiającym za wyborem właśnie tego serwera była jego intergracja z Eclipse (środowiskiem programistycznym w którym wykonany został projekt) za pomocą wtyczki JBossTools co znacznie ułatwiło pracę.

# 3. Opracowanie koncepcji systemu

Niniejszy rozdział przedstawia koncepcję implementowanej aplikacji. Zapisany został w dwóch podrozdziałach, przy czym pierwszy z nich obejmuje opis systemu, drugi natomiast specyfikację wymagań. Dokumentacja projektu zawarta w tym rozdziale oraz w następnym została częściowo oprata na szablonie dokumentacji projektu stworzonym przez dr Piotra Szweda [6].

## 3.1. Opis systemu

W tym podrozdziale opisany jest cel systemu, jego granice i lista możliwości, a także typy jego użytkowników. Ponadto za pomocą diagramów aktywności przedstawione jest działanie aplikacji.

### 3.1.1. Cel projektu

Celem projektu jest implementacja serwisu internetowego dedykowanego dla studentów w architekturze SOA. Głównym zadaniem serwisu będzie możliwość przechowywania plików oraz udostępnianie ich innym użytkownikom. Ponadto praktyczną funkcjonalnością będzie koordynacja kalendarza, dedykowanego indywidualnie dla każdego użytkownika.

### 3.1.2. Użytkownicy

Grupę użytkowników stanowią wszyscy korzystającyz serwisu, z wyszczególnieniem:

– **Użytkownik** - Docelowym użytkownikiem serwisu jest student potrzebujący dodać lub uzyskać dostęp do pliku poprzez udostępnienie od innych użytkowników serwisu. Nie jest warunkiem koniecznym, aby użytkownik był studentem,

– **Administrator** - Użytkownik posiadający wszystkie uprawnienia, takie jak usuwanie użytkowników oraz wgląd do wszystkich plików w systemie.

### 3.1.3. Granice systemu

Granicą systemu zarówno dla użytkownika jak i administratora, będą strony internetowe aplikacji, gdzie zawarty będzie dostęp do wszystkich funkconalności systemu. Dodatkowym uprawnieniem ad-

ministratora będzie dostęp do specjalnego panelu za pomocą którego będzie mógł zarządzać kontami wszystkich użytkowników.

Wejścia w systemie:

- formularz nowego konta – użytkownik wypełnia dane: login, hasło, imie, nazwisko, adres e-mail,
- formularz logowania – zalogowanie się do systemu za pomocą loginu i hasła użytkownika,
- formularz edycji konta – zmiana danych użytkownika,
- formularz nowego pliku – załącznik oraz opcjonalnie tytuł i opis pliku,
- formularz edycji pliku - zmiana załącznika, tytułu lub opisu,
- formularz udostępnienia pliku - login użytkownika, któremu ma być udostępniony wybrany plik,
- formularz nowego wydarzenia - wybrana data, nazwa wydarzenia oraz opcjonalny opis,
- formularz edycji wydarzenia - nowe dane do wybranego wydarzenia.

Wyjścia w systemie:

- Strona powiadomień - na tej stronie znajdować się będą informacje o udostępnionych plikach oraz o przypomnienia o nadchodzących wydarzeniach,
- Strona przeglądania własnych plików - użytkownik na tej stronie może przeglądać dodane przez siebie pliki,
- Strona przeglądania udostępnionych plików - tutaj użytkownik będzie mógł przeglądać pliki mu udostępnione,
- Strona kalendarza - Strona odpowiedzialna za wyświetlanie kalendarza.

### 3.1.4. Lista możliwości

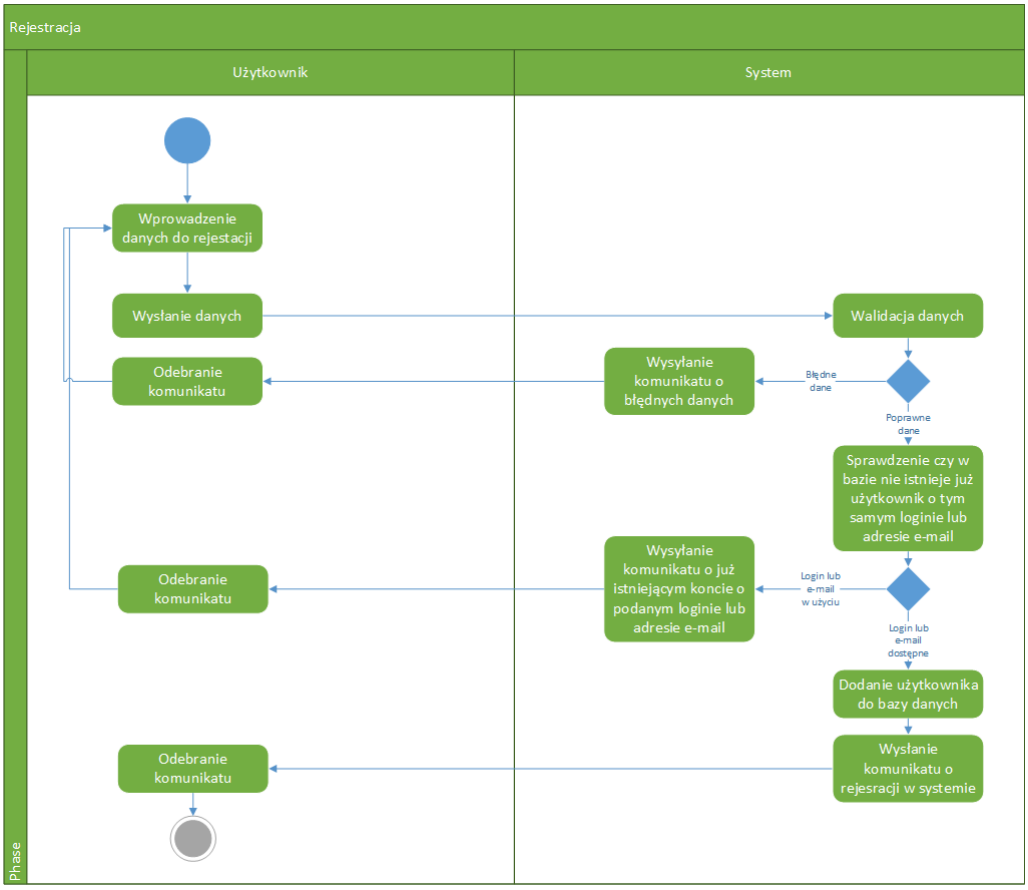
**Lista wymagań funkcjonalnych:**

- Logowanie,
- Wylogowanie,
- Rejestracja,
- Zarządzanie kontem,
- Zarządzanie materiałami,
- Pobieranie pliku,
- Udostępnianie pliku,
- Zarządzanie kalendarzem,
- Otrzymywanie powiadomień.

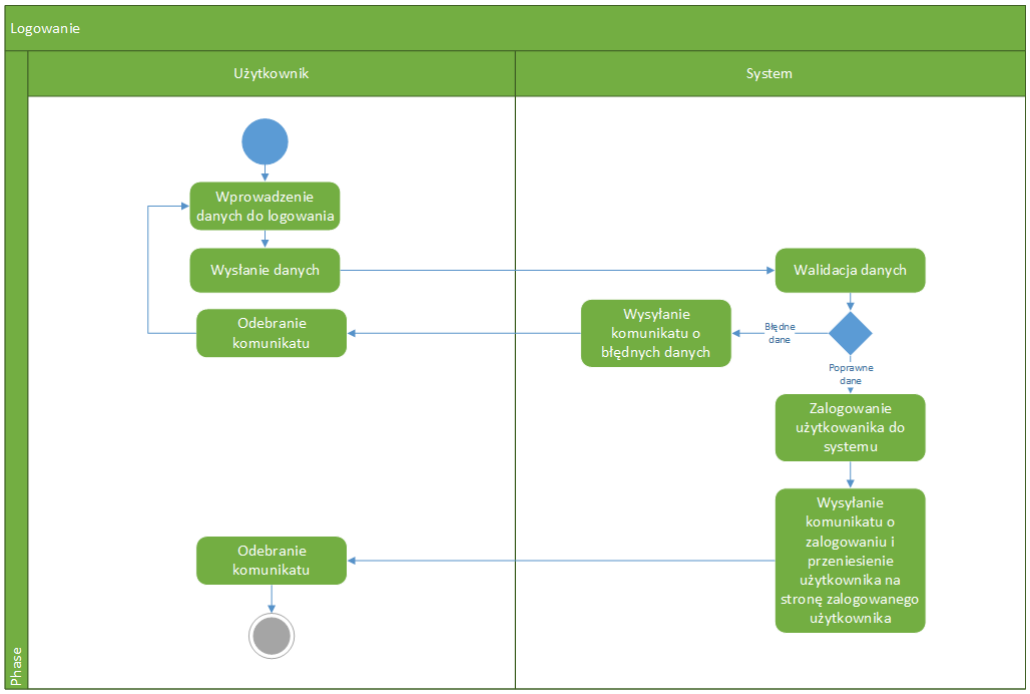
**Dodatkowe funkcje administratora:**

- Zarządzanie kontami użytkowników.

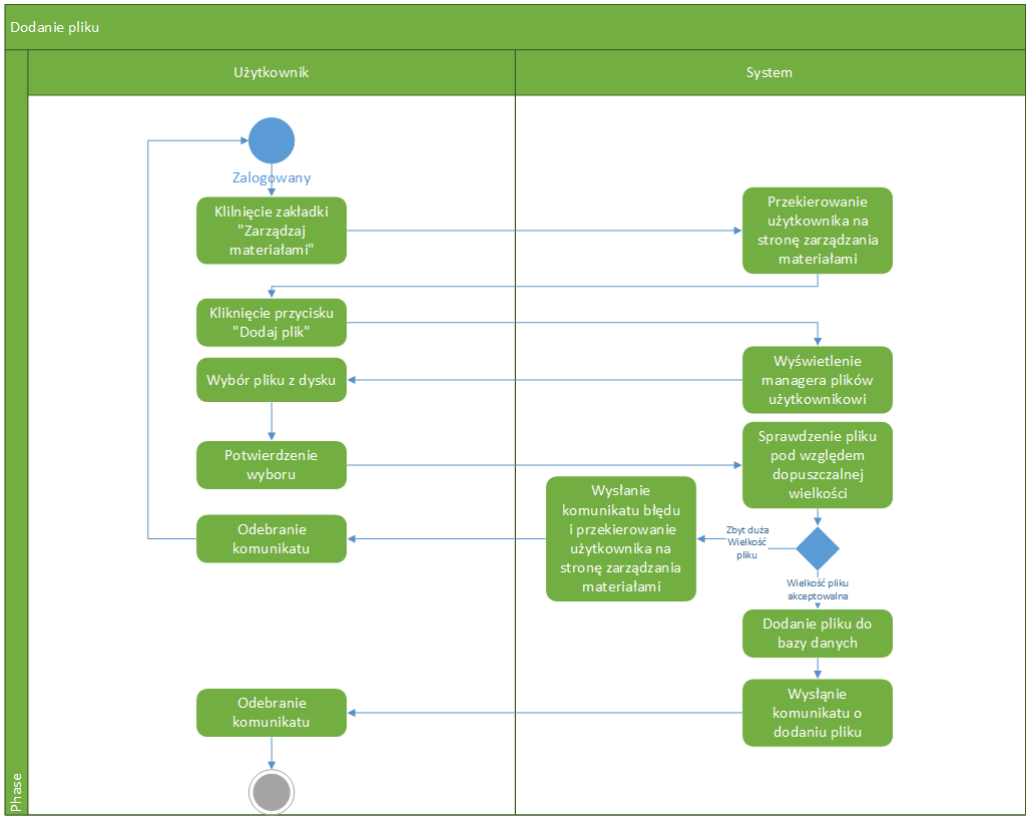
**Poniżej zostały przedstawione diagramy czynności typowych akcji.**



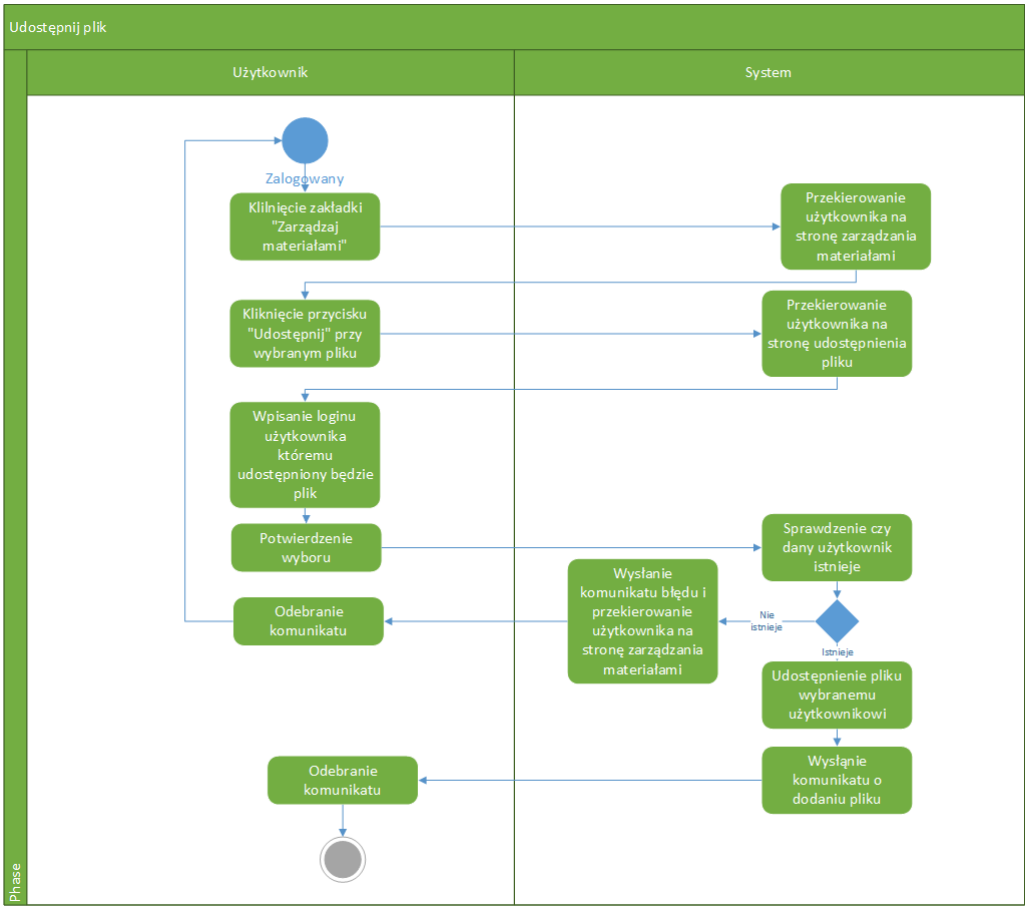

**Rys. 3.1.** Rejestracja




**Rys. 3.2.** Logowanie




**Rys. 3.3.** Dodanie Pliku




**Rys. 3.4.** Udostępnienie Pliku

## 3.2. Specyfikacja i analiza wymagań

Dzięki analizie wymagań funkcjonalnych możemy zidentyfikować oczekiwane zachowania budowanego systemu. Wymaganie funkcjonalne jest to „stwierdzenie, jakie usługi ma oferować system, jak ma reagować na określone dane wejściowe oraz jak ma się zachowywać w określonych sytuacjach. W niektórych wypadkach wymagania funkcjonalne określają, czego system nie powinien robić” [7]. W tym podrozdziale opisane zostaną wymagania niefunkcjonalne oraz przedstawiony zostanie diagram przypadków użycia z dołączonymi scenariuszami.

### 3.2.1. Wymagania oprogramowania

Wymagania funkcjonalne zostały wypisane w podpunkcie 3.1.4.

Wymagania niefunkcjonalne dzielimy na:

1. Wymaganie organizacyjne:

- Implementacji - językiem programowania jest Java,
- Użyteczności - interfejs graficzny użytkownika jest przejrzysty i łatwy do przyswojenia,

2. Wymagania zewnętrzne:

- Poufności - projekt przestrzega wymagań prawnych ustanowionych w Ustawie o Ochronie Danych Osobowych,
- Bezpieczeństwa - system nie pozwala użytkownikom na nieautoryzowany dostęp do bazy danych.

### 3.2.2. Ryzyka operacyjne

Przy implementacji systemu instnieje wiele ryzyk operacyjnych. W opisywanym projekcie są to:

– Niedostateczne zabiezpieczenie bazy danych przed nieautoryzowanym dostępem,

– Przeciążenie bazy danych przez zbyt wielu użytkowników i zapisywane przez nich pliki,

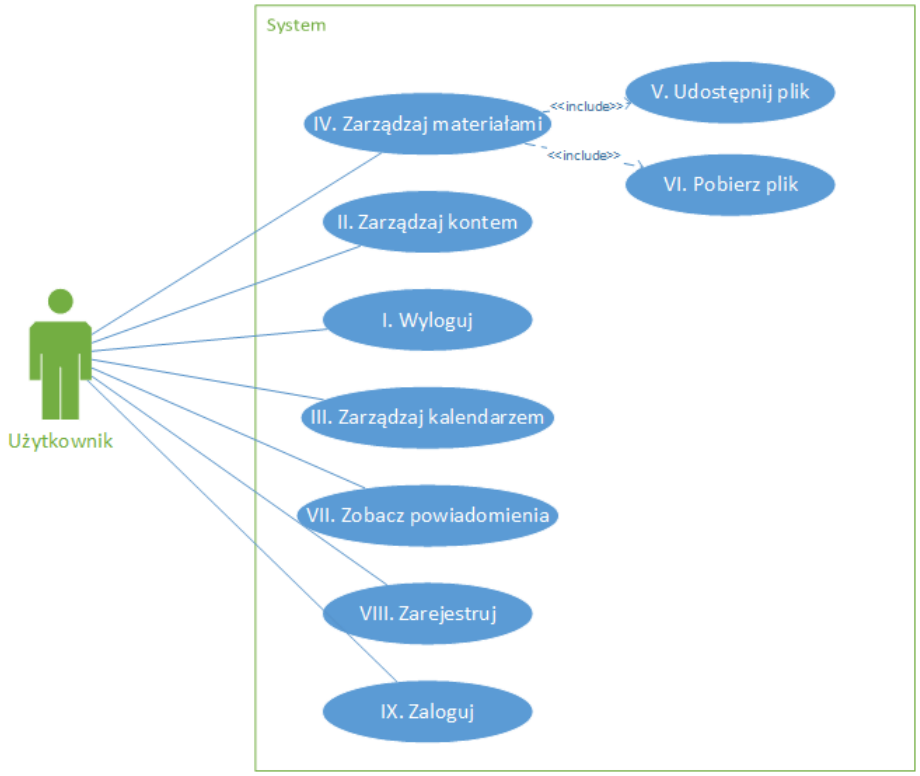
– Nieprzystosowanie serwera do dużego obciążenia,

– Nieobsłużenie ważnych błędów.

### 3.2.3. Przypadki użycia

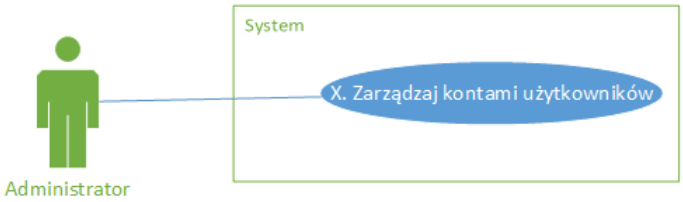
Dzięki przypadkom użycia możemy okeślić jak użytkownik będzie korzystał z systemu. Dzięki diagramowi możemy zobaczyć jakie czynności może wykonywać dany użytkownik [8]. Opisy czynności zawartych na diagramie znajdują się w scenariuszach przypadków użycia.

Poniższe diagramy prezentują możliwe interakcje użytkowników z systemem.




**Rys. 3.5.** Diagram przypadków użycia dla użytkownika

W poniższym diagramie pominięto przypadki użycia, które posiada zwykły użytkownik, lecz nie należy zapominać, że administrator również je posiada.




**Rys. 3.6.** Diagram przypadków użycia dla administratora

Scenariusze przypadków użycia:

**Tabela 3.1.** I. Wyloguj

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce się wylogować |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk “Wyloguj” |
| **Warunki wstępne:** | Użytkownik jest zalogowany |
| **Warunki końcowe dla sukcesu:** | Użytkownik jest wylogowany |
| **Warunki końcowe dla niepowodzenia:** | Użytkownik jest zalogowany |

**Scenariusz główny:**

1. Użytkownik wciska przycisk wyloguj widoczny w prawym górnym rogu strony.
2. Pojawia się okno z napisem: “Czy na pewno chcesz się wylogować?”
3. Użytkownik wciska przycisk “tak”.
4. Użytkownik jest wylogowany i przekierowany na stronę główną systemu.

**Scenariusz alternatywany:**

3a. Użytkownik wciska “nie”.

3a.1. Okno z napisem znika, użytkownik dalej jest zalogowany.

**Tabela 3.2.** II. Zarządzaj kontem

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce zmienić login lub hasło |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk “Zarządzaj kontem” |
| **Warunki wstępne:** | Użytkownik jest zalogowany |
| **Warunki końcowe dla sukcesu:** | Login lub hasło użytkownika są zmienione |
| **Warunki końcowe dla niepowodzenia:** | Login lub hasło użytkownika nie są zmienione |

**Scenariusz główny:**

1. Użytkownik wciska przycisk: “Zarządzaj kontem”.

2. System przenosi użytkownika do strony zarządzania kontem, na której ma do wyboru dwie opcje: “Zmień login” i “Zmień hasło”

3. Użytkownik wybiera opcje “Zmień login”.

4. System przenosi użytkownika do strony zmiany loginu, na której znajduje się formularz z polem tekstowym na nowy login.

5. Użytkownik wpisuje nowy login i wciska “Zatwierdź”.

6. System sprawdza czy nowy login nie występuje w bazie danych.

7. System wyświetla okno o pomyślnej zmianie loginu i przekierowuje użytkownika do strony zarządzania kontem.

**Scenariusz alternatywny:**

3a. Użytkownik wybiera opcje “Zmień hasło”.

3a.1. System przenosi użytkownika do strony zmiany hasła, na której znajduje się formularz z polem tekstowym na nowe hasło.

3a.2. Użytkownik wpisuje nowe hasło w pola “hasło” oraz “potwierdź hasło” i wciska “Zatwierdź”.

3a.3. System sprawdza czy wartości z pola hasło i potwierdź hasło są takie same.

3a.4. System sprawdza czy hasło ma odpowiednią długość i zawiera duże znaki.

3a.5. System wyświetla komunikat o poprawnej zmianie hasła i przenosi użytkownika do strony zarządzania kontem.

**Scenariusz alternatywny:**

3a.3a Wartość z pola hasło i potwirdź hasło nie są takie same.

3a.3a.1 System wyświetla odpowiedni komunikat błędu.

**Scenariusz alternatywny:**

7b. System wyświetla komunikat błędu: “Podany login już istnieje w systemie”.

7b.1. Powrót do pkt.4 scenariusza głównego.

**Tabela 3.3.** III. Zarządzaj kalendarzem

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce dodać lub usunąć wydarzenie w kalendarzu |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk “Zarządzaj kalendarzem” |
| **Warunki wstępne:** | Użytkownik jest zalogowany |
| **Warunki końcowe dla sukcesu:** | Użytkownik dodał, edytował lub usunął wydarzenie |
| **Warunki końcowe dla niepowodzenia:** | Użytkownik nie dodał, nie edytował lub nie usunął wydarzenia |

**Scenariusz główny:**

1. Użytkownik wciska przycisk “Zarządzaj kalendarzem”.
2. System przekierowuje użytkownika na stronę zarządzania kalendarzem.
3. Użytkownik wybiera datę w kalendarzu.
4. System podświetla wybrany przez użytkownika dzień i wyświetla przycisk “dodaj wydarzenie”, “usuń wydarzenie” oraz “edytuj wydarzenie”, jeżeli w danym dniu jest jakieś zapisane.
5. Użytkownik wciska przycisk “dodaj wydarzenie”.
6. System przenosi użytkownika na stronę dodawania wydarzenia, na której znajduje się data, pole tekstowe na nazwe wydarzenia oraz na godzinę wydarzenia i czas trwania.
7. Użytkownik wypełnia formularz i wciska przycisk “zatwierdź”.
8. System sprawdza poprawność wpisanych danych.
9. Wydarzenie jest dodane do kalendarza.

**Scenariusz alternatywny:**

1b. System wyświetla komunikat błędu: “Błąd serwera”.
1b.1. Powrót do pkt.1 scenariusza głównego.

**Scenariusz alternatywny:**

8a. System wyświetla komunikat błędu: “Pole “czas trwania” musi być wypełnione liczbą.”
8a.1. Powrót do pkt. 4 scenariusza głównego.

**Scenariusz alternatywny:**

4a. Użytkownik wciska przycisk “usuń wydarzenie”.
4a.1. System przenosi użytkownika na stronę usuwania wydarzenia, na której znajduje się lista wydarzeń

zaplanowanych na wybrany dzień oraz przyciski “usuń” obok każdego z nich.

4a.2. Użytkownik wciska przycisk “usuń” obok jednego z wydarzeń.

4a.3. System wyświetla okno z pytaniem czy na pewno usunąć wydarzenie.

4a.4. Użytkownik wciska przycisk “tak”.

4a.5. System usuwa wydarzenie i wyświetla komunikat o pomyślnym usunięciu wydarzenia.

4a.6. System przekierowuje użytkownika na stronę zarządzania kalendarzem.

**Scenariusz alternatywny:**

4a.3a. Użytkownik wciska przycisk “nie”.

4a.3a.1. Powrót do pkt. 4 scenariusza głównego.

**Scenariusz alternatywny:**

4b. Użytkownik wciska przycisk “edytuj wydarzenie”.

4b.1. System przenosi użytkownika na stronę edycji wydarzenia, na której znajduje się lista wydarzeń zaplanowanych na wybrany dzień oraz przyciski “edytuj” obok każdego z nich.

4a.2. Użytkownik wciska przycisk “edytuj” obok jednego z wydarzeń.

4a.3. System wyświetla okno z pytaniem czy na pewno usunąć wydarzenie.

4a.4. Użytkownik wciska przycisk “tak”.

4a.5. System usuwa wydarzenie i wyświetla komunikat o pomyślnym usunięciu wydarzenia.

4a.6. System przekierowuje użytkownika na stronę zarządzania kalendarzem.

**Tabela 3.4.** IV. Zarządzaj materiałami

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce dodać, usunąć, pobrać, lub udostępnić plik |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk "Zarządzaj materiałami" |
| **Warunki wstępne:** | Użytkownik jest zalogowany |
| **Warunki końcowe dla sukcesu:** | Użytkownik dodał, usunął, pobrał, lub udostępnił plik |
| **Warunki końcowe dla niepowodzenia:** | Użytkownik pozostał na dotychczasowej stronie. |

**Scenariusz główny:**

1. Użytkownik wciska przycisk "zarządzaj materiałami".
2. System przekierowuje użytkownika na stronę zarządzania materiałami.
3. Użytkownik wciska przycisk "dodaj plik".
4. System wyświetla nowe okno z eksplorerem plików w celu wybrania pliku z lokalnego dysku użytkownika.
5. Użytkownik wybiera plik i zatwierdza wybór.
6. System sprawdza czy plik nie jest zbyt duży.
7. System dodaje plik do bazy i wyświetla informacje o pomyślnym dodaniu pliku do bazy.
8. Nowy plik pojawia się w oknie zarządzania materiałami.

**Scenariusz alternatywny:**

6a. System wyświetla komunikat błędu: "Plik zbyt duży."

**Scenariusz alternatywny:**

1a. System wyświetla komunikat "Błąd serwera".

1a.1. Użytkownik pozostaje na dotychczasowej stronie.

**Scenariusz alternatywny:**

2a. Użytkownik wybiera plik z listy plików użytkownika.

2a.1. System otwiera stronę zarządzania plikiem z informacją o wybranym pliku, oraz trzema przyciskami: "Usuń", "Pobierz" oraz "Udostępnij".

2a.2. Użytkownik wybiera opcję "Usuń".

2a.3. System wyświetla okno "Czy na pewno chcesz usunąć plik?".

2a.4. Użytkownik wciska przycisk "tak".

2a.5. Plik zostaje usunięty z systemu.

2a.6. System przekierowuje użytkownika do strony zarządzania materiałami.

**Scenariusz alternatywny:**

2a.3a. Użytkownik wybiera opcje “nie”.

2a.3a.1. Powrót do pkt 2 scenariusza głównego.

**Tabela 3.5.** V. Udostępnij plik

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce udostępnić plik |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk “udostępnij plik” |
| **Warunki wstępne:** | Użytkownik jest zalogowany. Użytkownik jest na stronie zarządzania materiałami |
| **Warunki końcowe dla sukcesu:** | Plik został udostępniony |
| **Warunki końcowe dla niepowodzenia:** | Plik nie został udostępniony |

**Scenariusz główny:**

1. Użytkownik wybiera plik z listy.

2. System otwiera stronę zarządzania plikiem z informacją o wybranym pliku, oraz trzema przyciskami: “Usuń”, “Pobierz” oraz “Udostępnij”.

3. Użytkownik wybiera opcję “Udostępnij”.

4. System otwiera stronę udostępniania pliku, na której znajduje się formularz, z polem na nazwę użytkownika oraz przycisk “zatwierdź”.

5. Użytkownik wpisuję nazwę użytkownika i wciska przycisk “zatwierdź”.

6. System po znalezieniu użytkownika w bazie udostępnia plik wybranemu użytkownikowi.

7. System wyświetla wiadomość o pomyślnym udostępnieniu pliku.

**Scenariusz alternatywny:**

5a. System nie znajduje użytkownika w bazie.

5a.1. System wyświetla komunikat o niepowodzeniu.

**Tabela 3.6.** VI. Pobierz plik

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce pobrać plik |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk “pobierz plik” |
| **Warunki wstępne:** | Użytkownik jest zalogowany. Użytkownik jest na stronie zarządzania materiałami |
| **Warunki końcowe dla sukcesu:** | Plik został pobrany |
| **Warunki końcowe dla niepowodzenia:** | Plik nie został pobrany |

**Scenariusz główny:**

1. Użytkownik wybiera plik z listy.
2. System otwiera stronę zarządzania plikiem z informacją o wybranym pliku, oraz trzema przyciskami: “Usuń”, “Pobierz” oraz “Udostępnij”.
3. Użytkownik wybiera opcję “Pobierz”.
4. System rozpoczyna pobieranie pliku do domyślnego folderu na lokalnej maszynie użytkownika.
5. System zakończył pobieranie pliku i wyświetla komunikat o powodzeniu.

**Scenariusz alternatywny:**

4a. System nie może pobrać pliku, ponieważ nastąpiło rozłączenie z serwerem.

4a.1. System wyświetla komunikat o niepowodzeniu.

**Tabela 3.7.** VII. Zobacz powiadomienia

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce zobaczyć powiadomienia |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk “Zobacz powiadomienia” |
| **Warunki wstępne:** | Użytkownik jest zalogowany |
| **Warunki końcowe dla sukcesu:** | Użytkownik znajduje się na stronie powiadomień |
| **Warunki końcowe dla niepowodzenia:** | Użytkownik pozostał na dotychczasowej stronie |

**Scenariusz główny:**

1. Użytkownik wciska przycisk “Zobacz powiadomienia”.

2. System otwiera stronę powiadomień.

**Scenariusz alternatywny:**

1a. System wyświetla błąd.

1a.1. Użytkownik pozostaje na dotychczasowej stronie.

**Tabela 3.8.** VIII. Zarejestruj

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Niezarejestrowany użytkownik chce się zarejestrować |
| **Zdarzenie wyzwalające:** | Niezarejestrowany użytkownik wciska przycisk Źarejestruj" |
| **Warunki wstępne:** | Brak |
| **Warunki końcowe dla sukcesu:** | Niezarejestrowany użytkownik znajduje się na stronie głównej |
| **Warunki końcowe dla niepowodzenia:** | Konto nie zostaje stworzone, a użytkownik zostaje poinformowany o niepowodzeniu oraz podane zostają przyczyny błędu np. adres email jest już wykorzystany lub login jest już zajęty |

**Scenariusz główny:**

1. System wyświetla formularz rejestracyjny.

2. Użytkownik wypełnia formularz.

3. Użytkownik wciska zatwierdź.

4. System sprawdza poprawność danych i unikalność loginu.

5. Konto użytkownika zostaje utworzone

**Scenariusz alternatywny:**

4a. System znajduje konto o podanym loginie.

4a.1. System wyświetla ponownie formularz z informacją, że konto o podanym loginie już istnieje.

4a.2. Powrót do punktu 2 scenariusza głównego.

**Scenariusz alternatywny:**

4b. System wykrył polskie znaki w haśle.

4b.1 System wyświetla ponownie formularz z informacją, że nie może być polskich znaków w haśle.

4b.2. Powrót do punktu 2 scenariusza głównego.

**Scenariusz alternatywny:**

2a. Użytkownik odświeża stronę.

2a.1. Użytkownik przekierowany jest na stronę główną serwisu.

**Tabela 3.9.** IX. Zaloguj

| | |
|---|---|
| **Aktorzy:** | Użytkownik, Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Użytkownik chce się zalogować |
| **Zdarzenie wyzwalające:** | Użytkownik wciska przycisk źaloguj" |
| **Warunki wstępne:** | Użytkownik musi posiadać konto w systemie |
| **Warunki końcowe dla sukcesu:** | Użytkownik znajduje się na stronie głównej |
| **Warunki końcowe dla niepowodzenia:** | Użytkownik jest niezalogowany |

**Scenariusz główny:**

1. System wyświetla formularz logowania.
2. Użytkownik wypełnia formularz.
3. Użytkownik wciska zatwierdź.
4. System sprawdza poprawność danych.
5. Użytkownik zostaje zalogowany.

**Scenariusz alternatywny:**

4a. System odrzuca wprowadzone dane, ponieważ login lub hasło są nieprawidłowe.

4a.1. System wyświetla ponownie formularz z informacją o błędnym loginie lub haśle.

4a.2. Powrót do punktu 2 scenariusza głównego.

**Tabela 3.10.** X. Zarządzaj kontami użytkowników

| | |
|---|---|
| **Aktorzy:** | Administrator |
| **Zakres:** | System |
| **Poziom:** | Sytemowowy |
| **Udziałowcy i ich cele:** | Administrator chce usunąć, lub edytować konto użytkownika |
| **Zdarzenie wyzwalające:** | Administrator wciska przycisk “zarządzaj kontami” |
| **Warunki wstępne:** | Administrator jest zalogowany |
| **Warunki końcowe dla sukcesu:** | Konto użytkownika zostało usunięte lub edytowane |
| **Warunki końcowe dla niepowodzenia:** | Konto użytkownika nie zostało usunięte lub edytowane |

**Scenariusz główny:**

1. Administrator wciska przycisk “zarządzaj kontami”.
2. System przekierowuje na stronę zarządzania kontami użytkowników, gdzie wylistowane są loginy wszystkich użytkowników, a obok nazwy są dwa przyciski: “usuń” oraz “edytuj” oraz bad listą pole na wpisanie loginu i przycisk do zatwierdzania.
3. Administrator klika “usuń” przy wybranym loginie.
4. System wyświetla komunikat o treści “Czy na pewno chcesz usunąć użytkownika z bazy danych?” oraz dwie opcje odpowiedzi: “tak” i “nie”.
5. Administrator wciska “tak”
6. System usuwa użytkownika z bazy danych.
7. System wyświetla komunikat o udanej operacji.

**Scenariusz alternatywny:**

2a. Administrator wciska “edytuj”.
2a.1. System przekierowuje administratora na stronę edycji użytkownika z formularzem do edycji.
2a.2. Administrator wprowadza nowe dane i potwierdza zmianę przyciskiem “zatwierdź”.
2a.3. System zmienia dane użytkownika w bazie.
2a.4. System wyświetla komunikat o udanej operacji.
2a.5. Powrót do pkt 2 scenariusza głównego.

**Scenariusz alternatywny:**

2b. Administrator wpisuje login w odpowiednie pole i wciska przycisk szukaj
2b.1. System wyświetla tylko użytkownika o podanym loginie.

# 4. Projekt architektury systemu

W tym rozdziale omówiona zostanie architektura implementowanego systemu. W pierwszym podrozdziale znajduje się ogólny opis używanej archikterktury, w następnym model bazy danych, a w ostatnim model klas.

## 4.1. Opis architektutry

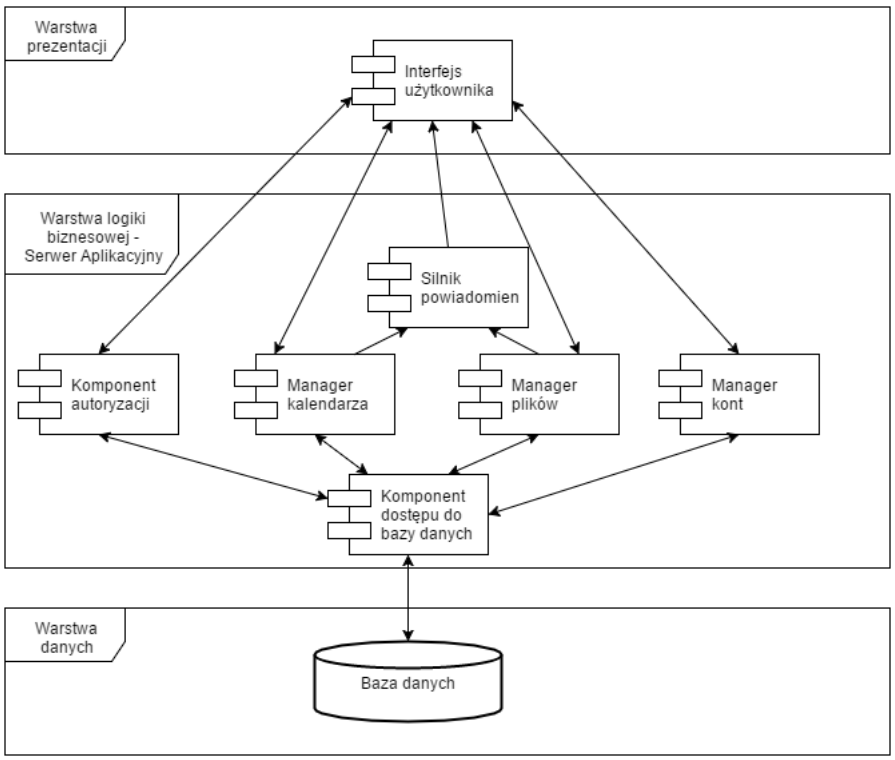
Implementowany projekt jest oparty na architekturze SOA (wyjaśnionej w podrozdziale 2.1), która została wybrana przede wszystkim ze względu na podział aplikacji na niezależne od siebie moduły. Dzięki modularności w łatwy sposób można dodawać nowe funkcjonalności do projektu poprzez wstawianie kolejnych usług. Oprócz opisanej wcześniej architektury do implementacji projektu został użyty wzorzec projektowy MVC (*ang. Model View Controller*), który jak sama nazwa wskazuje odpowiada za podzielenie aplikacji na trzy części:

- Model - część odpowiedzialna za zarządzanie danymi czyli za połączenie z bazą danych,

- View - widok odpowiada za interakcję użytkownika z systemem czyli graficzny interfejs użytkownika,

- Controller - kontroler oddziałuje pomiędzy modelem a widokiem i kontroluje przepływ danych pomiędzy nimi [9].

Opisywaną architekturę przedstawia poniższy model komponentów:




**Rys. 4.1.** Diagram komponentów

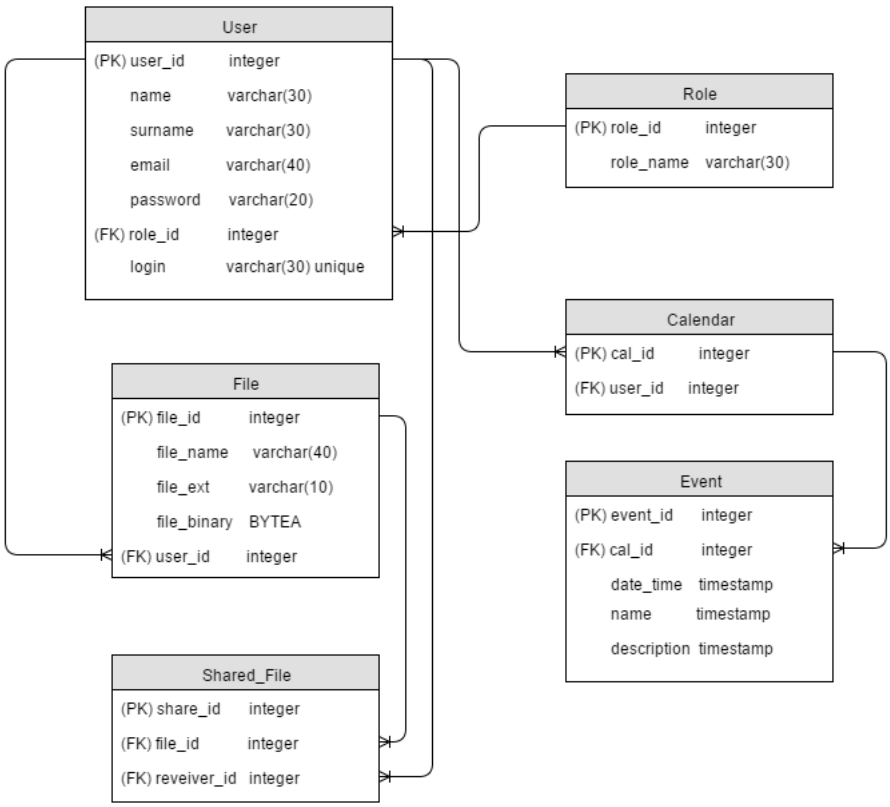
Każdy z komponentów zostanie opisany dokładnie w rozdziale implementacyjnym.

## 4.2. Model bazy danych

Baza danych składa się z sześciu tabel:

- User - odpowiedzialna za przechowywanie danych o użytkownikach,
- Role - Wyznaczająca rolę użytkownika: *user* lub *admin*, dzięki którym można rozróżnić uprawnienia,
- File - Reprezentuje plik,
- Calendar - odpowiedzialny za przechowywanie kalendarza użytkownika,
- Event - prezentuje wydarzenie,
- Shared_File - dzięki tej tabeli użytkownik może dawać dostęp do swojego pliku innym użytkownikom.

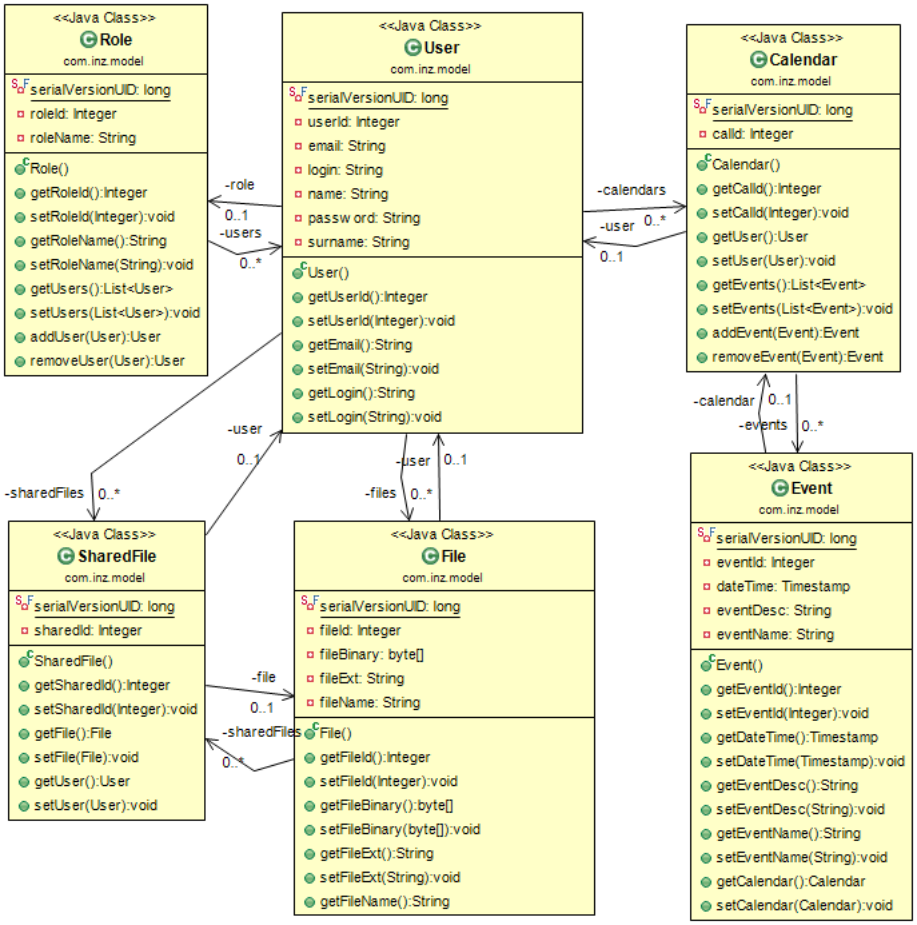
Model bazy danych przedstawiony jest poniżej na diagramie ERD:




**Rys. 4.2.** Diagram ERD

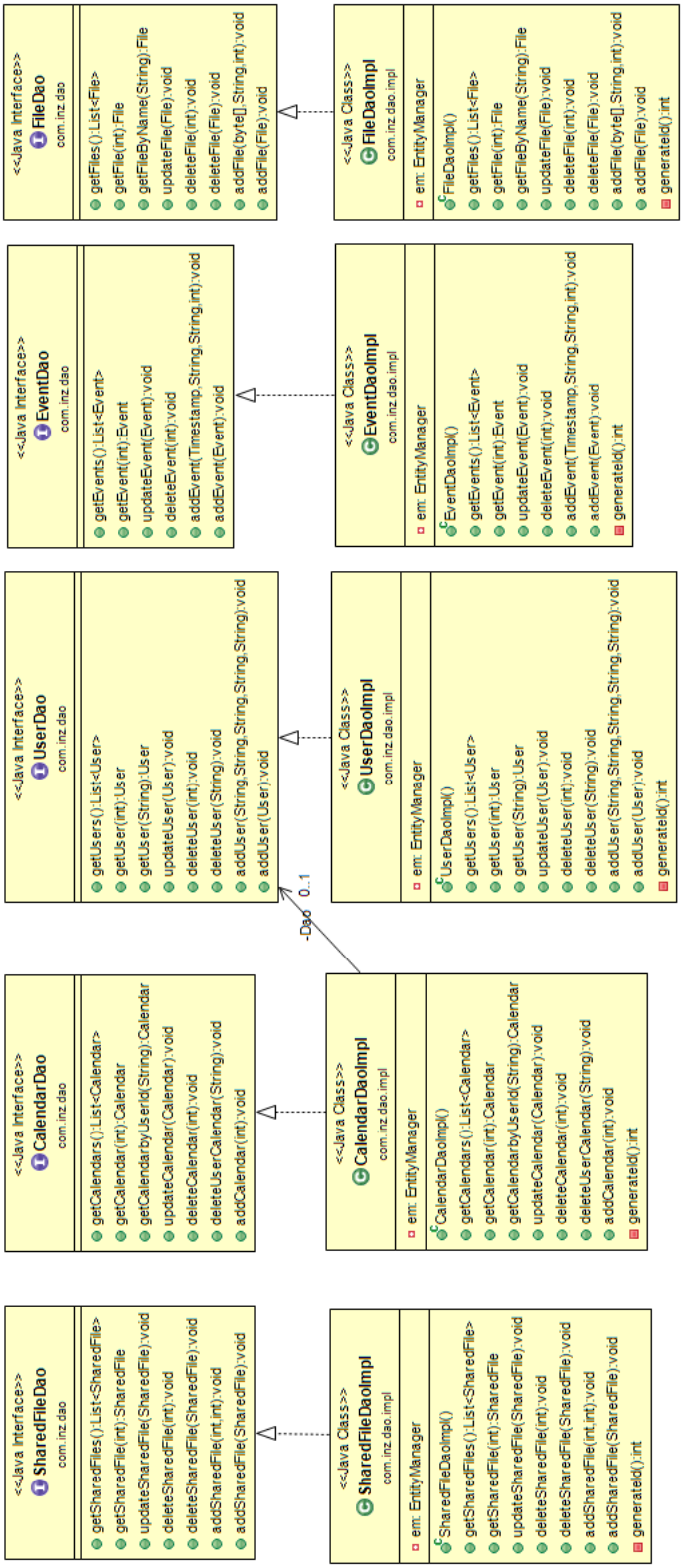
## 4.3. Model klas

Model klas w postaci diagramu UML reprezentujący tabele z bazy danych:




**Rys. 4.3.** Diagram Klas Encji

Za połączenie z bazą danych odpowiada wzorzec projektowy DAO (omówiony w 5.2.1), który przedstawiony jest na poniższym diagramie.




**Rys. 4.4.** Diagram Klas DAO

# 5. Implementacja

Rozdział ten opisuje najważniejsze szczegóły implementacyjne projektu. W pierwszym podrozdziale znajduje się krótki opis narzędzi użytych do implementacji oraz sposób w jaki ich użyto. W drugim znajduje się opis podprojektów, a w trzecim metody zabezpieczeń systemu.

## 5.1. Użyte narzędzia

Aplikacja została zaimplementowana w środowisku programowania **Eclipse IDE for Java EE Developers**, ponieważ posiada wszystkie niezbędne narzędzia do tworzenia aplikacji JEE. Innym plusem użytego środowiska jest także dodatkowy plugin JBossTools odpowiadający za zarządzanie serwerem aplikacyjnym JBoss.

Dodatkowym bardzo przydatnym narzędziem użytym do implementacji projektu jest **Apache Maven**, który jest odpowiedzialny za budowanie aplikacji. Tworzenie aplikacji przy użyciu narzędzia maven opiera się na pliku pom.xml (POM – ang. Project Object Model). Jest on miejscem, w którym znajdują się wszystkie najważniejsze informację definiujące projekt, jego strukturę, to w jaki sposób będzie budowany oraz zależności [10]. Dużą zaletą Mavena jest łątwy sposób dodawania potrzebnych bibliotek do projektu, co odbywa się za pomocą przyłączania kolejnych zależności do pliku pom.xml, które są automatycznie ściągane z repozytorium mavena. Przykład dodania biblioteki JSF w projekcie:

```
<dependency>
   <groupId>org.jboss.spec.javax.faces</groupId>
   <artifactId>jboss-jsf-api_2.1_spec</artifactId>
   <scope>provided</scope>
</dependency>
```

Maven posiada archetypy czyli dodatkowe narzędzie do generowania szablonów projektów. Przy tworzeniu opisywanej aplikacji skorzystano z tej właśnie funkcjonalności. Szablon tworzy się za pomocą wywołania komendy:

```
mvn archetype:generate
```

Następnie z pośród tysięcy dostępnych szablonów trzeba wybrać ten, który nas interesuje. W przypadku tego projektu jest to archetyp który buduje szablon aplikacjii JavaEE kompatybilny z serwerem aplikacyjnym JBoss AS 7.1:

```
remote -> org.jboss.spec.archetypes:jboss-javaee6-webapp-ear-archetype (An archetype
     that generates a starter Java EE 6 webapp project for JBoss AS 7.1 (by default)
     or EAP 6 (if the "enterprise" property is true). The project is an EAR, with an
     EJB-JAR and WAR)
```

Po wybraniu archetypu następuje definicja nazwy projektu oraz nazwy domyślnego pakietu. Wybrany pakiet dzieli projekt na trzy podprojekty:

– EJB - Odpowiedzialny za połączenie z bazą danych. Tutaj znajdują się klasy reprezentujące tabele w bazie danych oraz klasy Bean, które są odpowiedzialne za pobieranie i wstawianie danych z bazy. Podprojekt EJB eksportowany jest w postaci plików JAR (ang. Java archive),

– WEB - W tym podprojekcie znajdują się wszystkie strony html tworzące interfejs użytkownika, oraz klasy Managed Bean, które są odpowiedzialne za połączenie ze stronami html. Podczas budowy aplikacji podprojekt WEB zapisywany jest jako WAR (ang. Web archiwe),

– EAR - Aplikacja eksportowana jest w postaci plików EAR (enterprise archiwe), które zawierają wszystkie podprojekty implementowanej aplikacji [11].

Zbudowany projekt EAR jest lokowany na serwerze aplikacyjnym JBoss.

## 5.2. Oprogramowanie

Projekt jak to zostało opisane w poprzednim podrozdziale jest podzielony na trzy podprojekty, z których implementacja znajduje się tylko w częściach WEB i EJB. W tym podrozdziale przybliżona zostanie ich zawartość.

### 5.2.1. EJB

W opisywanym projekcie znajdują się klasy, które zostały wygenerowane dzięki narzędziu JPA Tools (ang. Java Persistance API) z tabel bazy danych opisanych w punkcie 4.2. Przykładowa wygenerowana klasa reprezentująca użytkownika:

```
package com.inz.model;

import java.io.Serializable;
import javax.persistence.*;
import java.util.List;

/**
 * The persistent class for the users database table.
 *
 */
@Entity
```

```
@Table(name="users")
@NamedQueries({
        @NamedQuery(name="user.findAll", query="SELECT u FROM User u"),
        @NamedQuery(name="user.byId", query="select u from User as u where u.userId=:userId"),
        @NamedQuery(name="user.byLogin", query="select u from User as u where u.login=:login"),
})
public class User implements Serializable {
        private static final long serialVersionUID = 1L;

        @Id
        @Column(name="user_id", columnDefinition = "serial")
        private Integer userId;
        private String email;
        private String login;
        private String name;
        private String password;
        private String surname;

        //bi-directional many-to-one association to Calendar
        @OneToMany(mappedBy="user")
        private List<Calendar> calendars;

        //bi-directional many-to-one association to File
        @OneToMany(mappedBy="user")
        private List<File> files;

        //bi-directional many-to-one association to SharedFile
        @OneToMany(mappedBy="user")
        private List<SharedFile> sharedFiles;

        //bi-directional many-to-one association to Role
        @ManyToOne
        @JoinColumn(name="role_id")
        private Role role;

        public User() {
        }

        /* Metody get i set dla wszystkich atrubutów */
}
```

W powyższym przykładzie ominięte zostały metody get i set odpowiadające za pobieranie i ustawianie atrybutów klasy. Właściwości tabeli w wygenerowanej klasie są zachowane dzięki adnotacjom (@Adnotacja), gdzie @Id definiuje klucz główny tabeli, @OneToMany i @ManyToOne relacje zachodzące między tabelami i @Column podający dodatkowe właściwości kolumny, które nie mogą być przedstawione przy pomocy zmiennej w języku Java. Ponadto ponad definicją klasy znajdują się ad-

notacje @Entity, która oznacza klasę jako reprezentującą tabelę z bazy danych, @Table pozwalająca ustawić nazwę tabeli do której się odwołuje oraz @NamedQueries, która pozwala na zdefiniowanie w prosty sposób zapytań do bazy danych. Dzięki frameworkowi JPA i wygenerowanym przez niego klasom nie trzeba się odwoływać do tabel z bazy, lecz do powstałych klas.

Aby ułatwić innym komponentom dostęp do bazy danych skorzystano z wzorca projektowego DAO (ang. Data Access Object), który tworzy jednolity interfejs odpowiedzialny za połączenie z bazą danych. DAO sprawia, że warstwa odpowiadająca za logike aplikacji nie musi znać szczegółów implementacyjnych udostępnionego interfejsu co uniezależna tą warstwę od sposobu dostępu do bazy danych. Poniżej zostanie przedstawiony przykładowy interfejs i jego implementacja odpowiedzialne za zarządzanie tabelą użytkownika.

Interfejs:

```
package com.inz.dao;

import java.util.List;
import com.inz.model.User;

public interface UserDao {
        List<User> getUsers();
        User getUser(int id);
        User getUser(String login);
        void updateUser(User user);
        void deleteUser(int id);
        void deleteUser(String login);
        void addUser(String login, String name, String surname, String email, String
            password);
        void addUser(User user);
}
```

Implementacja:

```
package com.inz.dao.impl;

import java.util.LinkedList;
import java.util.List;

import javax.ejb.Local;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import com.inz.dao.UserDao;
import com.inz.model.User;
```

```
@Stateless
@Local(UserDao.class)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public class UserDaoImpl implements UserDao {

        @PersistenceContext(name="primary")
        private EntityManager em;

        @Override
        public List<User> getUsers() {
                return em.createNamedQuery("user.findAll").getResultList();
        }

        @Override
        public User getUser(int id) {
                return (User) em.createNamedQuery("user.byId")
                        .setParameter("userId", id).getSingleResult();
        }

        @Override
        public User getUser(String login) {
                return (User) em.createNamedQuery("user.byLogin")
                        .setParameter("login", login).getSingleResult();
        }

        @Override
        public void updateUser(User user) {
                em.merge(user);
        }

        @Override
        public void deleteUser(int id) {
                User u = getUser(id);
                em.remove(u);
        }

        @Override
        public void deleteUser(String login) {
                User u = getUser(login);
                em.remove(u);
        }

        @Override
        public void addUser(String login, String name, String surname, String email,
                        String password) {
                User u = new User();
                u.setLogin(login);
```

```
                u.setName(name);
                u.setSurname(surname);
                u.setEmail(email);
                u.setPassword(password);
                u.setUserId(generateId());
                em.persist(u);
        }

        @Override
        public void addUser(User user) {
                if(user.getUserId().equals(null)){
                        user.setUserId(generateId());
                }
                em.persist(user);
        }

        private int generateId(){
                List<User> users = getUsers();
                int id=0;
                List<Integer> ints = new LinkedList<Integer>();
                for(User usr:users){
                        ints.add(usr.getUserId());
                }
                int i=1;
                while(true){
                        if(ints.contains(i)) i++;
                        else{
                                id=i;
                                break;
                        }
                }
                return id;
        }
}
```

Najważniejszą zmienną odpowiedzialną za wykonywanie operacji na bazie danych jest instancja klasy EntityManager, która jest "wstrzyknięta"do klasy za pomocą adnotacji @PersistenceContext. Adnotacja @Stateless oznacza, że wstrzykniwany Bean (komponent wielokrotnego użytku w różnych miejscach aplikacji) będzie bezstanowy. Komponent ten jak sama nazwa wskazuje nie posiada żadnego stanu więc wszystkie takie komponenty wywoływane przez różnych użytkowników są identyczne. Adnotacja @Local mówi o tym, że implementowany interfejs znajduje się w tym samym środowisku. Adnotacja @TransactionAttribute(TransactionAttributeType.REQUIRED) wymusza na każdej metodzie, aby była wykonana w ramach tranzakcji już działającej, a jeżeli żadna nie jest rozpoczęta to musi być utworzona nowa. Stworzony bean wstrzykuje się w następujący sposób:

```
@EJB private UserDao dao;
```

### 5.2.2. WEB

Użytkownik korzysta z aplikacją za pomocą przeglądarki internetowej, więc interfejsem są strony www. W implementowanej aplikacji są to pliki z rozszerzeniem xhtml. Głównym frameworkiem pomocnym przy tworzeniu tego interfejsu jest JSF, który udostępnia dodatkowe tagi pomocne przy tworzeniu stron oraz klasy Managed Bean dzięki którym można się połączyć z tymi stronami.
Przykładowy formularz odpowiedzialny za rejestracje nowego użytkownika:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR
    /xhtml1/DTD/xhtml1-transitional.dtd">
<html   xmlns="http://www.w3.org/1999/xhtml"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:f="http://java.sun.com/jsf/core">
<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Rejestracja</title>
</h:head>
<h:body>
        Zarejestruj się:
        <h:form>
        <br />
           Login: <h:inputText id="login" value="#{userMan.login}"
                               required="true"
                               validator="#{userMan.validateLogin}"
                               requiredMessage="Pole Login jest wymagane."/>
                               <h:message for="login" style="color:red" /> <br />
           Hasło: <h:inputSecret id="password" value="#{userMan.password}"
                                       required="true"
                               validatorMessage="Błąd walidacji: Hasło nie może
                                   zawierać znaków specjalnych oraz musi mieć długo
                                   ść od 5 do 30 znaków."
                               requiredMessage="Pole Hasło jest wymagane.">
                               <f:validateRegex
                                       pattern="[A-Za-z0-9]*{5,30}$" />
                           </h:inputSecret>
                           <h:message for="password" style="color:red" /> <br />
                Imię: <h:inputText id="name" value="#{userMan.name}"
                               required="true"
                               requiredMessage="Pole Imię jest wymagane."/>
                               <h:message for="name" style="color:red" /> <br />
        Nazwisko: <h:inputText id="surname" value="#{userMan.surname}"
                               required="true"
                               requiredMessage="Pole Nazwisko jest wymagane."/>
                               <h:message for="surname" style="color:red" /> <br />
           Email: <h:inputText id="email" value="#{userMan.email}"
                               required="true"
```

```
                                validatorMessage="Błąd walidacji: Email musi być
                                    postaci użytkownik@domena."
                                requiredMessage="Pole Email jest wymagane.">
                                <f:validateRegex
                                        pattern="^[_A-Za-z0-9-\+]+(\.[_A-Za-z0-9-]+)
                                            *@[A-Za-z0-9-]+(\.[A-Za-z0-9]+)*(\.[A-Za
                                            -z]{2,})$" />
                        </h:inputText>
                        <h:message for="email" style="color:red" /> <br />
                <h:commandButton value="Dodaj mnie" action="#{userMan.addUser()}" />
        </h:form>
</h:body>
</html>
```

Klasa Managed Bean odpowiedzialna za obsługę powyższego formularza:

```
package com.inz.jsf;

import java.util.List;
import javax.ejb.EJB;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;
import com.inz.dao.UserDao;
import com.inz.model.User;

@ManagedBean(name = "userMan", eager = true)
@SessionScoped
public class UserManager {
        @EJB private UserDao dao;
        private String login;
        private String name;
        private String surname;
        private String password;
        private String email;

        public void ManagedBean(){};

/* Metody get i set */

        public String addUser(){
                dao.addUser(login, name, surname, email, password);
                return "index";
        }
```

```
        public void validateLogin(FacesContext context, UIComponent component,
            Object value) throws ValidatorException {
                String login = (String) value;
                String message = "Błąd walidacji: ";
                boolean err = false;
                if ( login.length() > 30 || login.length() < 5 ) {
                        message=message+"Login musi mieś długość od 5 do 30 znaków
                            .";
                        err=true;
                } else if(validateLoginHelper(login)){
                        message=message+"Użytkownik o podanym loginie znajduje się
                             już w bazie danych.";
                        err=true;
                }
                if(err){
                        FacesMessage msg = new FacesMessage(message);
                    throw new ValidatorException(msg);
                }
            }

        private boolean validateLoginHelper(String login){
                List<User> users = dao.getUsers();
                for(User u:users){
                        if(u.getLogin().equals(login)) return true;
                }
                return false;
        }
}
```

ManagedBean to klasa z frameworku JSF, która jest dostępna ze stron używających tagów JSF. Definiowana jest za pomocą adnotacji @ManagedBean. Adnotacja @SessionScoped definiuje, że Bean żyje tak długo jak sesja HTTP.

Aby użytkownik nie mógł wprowadzać nieprawidłowych danych przy tworzeniu formularza zastosowano mechanizm walidacji wprowadzanych danych. Użytkownik nie może zostawić pola nieuzupełnionego (required="true"), a ponadto do pola login, hasło i email są sprawdzane pod względem wprowadzanego tekstu. W polach email i hasło sprawdzany jest warunek długości oraz treści wprowadzanego tekstu za pomocą wyrażenia regularnego na stronie JSF (f:validateRegex pattern="warunek"). Pole login walidowane jest za pomocą MangedBeana pod względem długości tekstu (od 5 do 30 znaków) oraz unikalności loginu w systemie. Jeżeli któryś z wyżej opisanych warunków nie zostanie spełniony to po prawej stronie pola pojawi się odpowiedni komunikat w kolorze czerwonym.

## 5.3. Zabezpieczenia

Wszystkie usługi będą udostępniane tylko zalogowanym użytkownikom. Możliwość zarządzania plikami oraz kalendarzem wszystkich użytkowników posiada tylko administrator, co jest autoryzowane za pomocą bazy danych oraz podsystemu bezpieczeństwa serwera aplikacyjnego JBoss.

W celu stworzenia logowania do systemu skorzystano z pomocy modułu logowania *Database* dostarczanego przez serwer aplikacyjny JBoss. Aby zrealizować autoryzację w pierwszej kolejności trzeba dodać odpowiednie elementy do pliku konfiguracyjnego serwera JBoss (standalone.xml):

```
<security-domain name="postgresql-domain" cache-type="default">
        <authentication>
                <login-module code="Database" flag="required">
                        <module-option name="dsJndiName" value="java:jboss/
                            datasources/StudentServisDataSource"/>
                        <module-option name="principalsQuery" value="select password
                             from users where login=?"/>
                        <module-option name="rolesQuery" value="select role_name, '
                            Roles' as Roles from role r join users u on (r.role_id=u
                            .role_id) where u.login=?"/>
                </login-module>
        </authentication>
</security-domain>
```

Powyższy wpis definiuje moduł logowania korzystający z weryfikacji za pomocą bazy danych zdefiniowanej w źródle danych *StudentServisDataSource*. Następnie przygotowany jest specjalny formularz logowania, po czym w pliku xml między znacznikami <security-constraint> definiowane są strony, do których dostęp ma tylko zalogowany użytkownik. Dodatkowo w pliku web.xml definiowane są strony logowania do których niezalogowany użytkownik jest przekierowywany gdy chce uzyskać dostęp do stron objętych ochroną:

```
<form-login-config>
        <form-login-page>/login.xhtml</form-login-page>
        <form-error-page>/login_error.xhtml</form-error-page>
</form-login-config>
```

Aby przydzielić niektóre funkcje systemu tylko administratorowi nad ManagedBeanem wstawiane są następujące adnotacje:

```
@RolesAllowed("admin")
@SecurityDomain("postgresql-domain")
```

Gdzie adnotacja @SecurityDomain definiuje baze danych na podstawie której użytkownik rola użytkownika jest weryfikowana.

# 6. Podsumowanie

Celem pracy była implementacja serwisu internetowego przeznaczonego dla studentów. Główną funkcjonalnością sytemu miała być możliwość dodawania plików do systemu oraz udostępnianie tych plików znajomym użytkownikom. Ponadto każdy użytkownik miał mieć prywatny kalendarz w celu dodawania ważnych wydarzeń. Ostatnią zaplanowaną funkcjonalnością miała być strona powiadomień przypominająca o nadchodzących wydarzeniach, także powiązanych ze stroną uczelni oraz udostępnionych plikach.

Cel pracy został osiągnięty, w wyniku implementacji powstał serwis internetowy zaimplementowany w architekturze SOA. Serwis posiada zaplanowaną funkcjonalność. Przechowywanie plików zostało zrealizowane za pomocą zapisywania plików w postaci binarnej w bazie danych. Interfejs użytkownika są strony internetowe utworzone za pomocą frameworku JSF. Niestety nie udało się zrealizować funkcjonalności pobierania aktualności z portalu Facebook z oficjalnych grup założonych przez uczelnie do której uczęszcza użytkownik.

W przypadku kontunuowania pracy przy projekcie poprawiony by został interfejs użytkownika za pomocą skorzystania z odpowiednich bibliotek lub narzędzi pomagających w jego tworzeniu. Ponadto gdyby serwis miałby być używany na szeroką skalę trzeba by wziąć pod uwagę kwestię lepszego zabezpieczenia danych użytkownika oraz lepszy system przechowywania plików.

Projekt dzięki architekturze SOA można rozwijać poprzez dodawanie kolejnych modułów dodających nową funkcjonalność. W przypadku rozwijania aplikacji dobrym pomysłem było by dodanie integracji w portalami społecznościowymi w celu zwiększenia zainteresowania serwisem oraz ułatwienia do niego dostępu. Kolejną funkcjonalnością mogącą przynieść serwisowi większe zainteresowanie było by przygotowanie aplikacji mobilnej na system Adroid.

# Bibliografia


[1] Claus T. Jensen. *SOA Design Principles for Dummies*. New Jersey: John Wiley and Sons, Inc., 2013.

[2] *The History of Java Technology*. `http://www.oracle.com/technetwork/java/javase/overview/javahistory-index198355.html`. Oracle.

[3] *JEE - wprowadzenie do Javy Enterprise Edition*. `http://javastart.pl/static/jee/jeewprowadzeniedojavyenterprise-edition/`.

[4] D. Krzemień; R.Rusiecki. *Hibernate + Java*. `http://student.agh.edu.pl/drusieck/hibernate/`. Akademia Górniczo-Hutnicza.

[5] R. Matusik. *Serwery aplikacji*. `http://math.uni.lodz.pl/ radmat/serwery/Wyklad_d_01.pdf`. Uniwersytet Łudzki.

[6] P. Szwed. *Szablon dokumentacji projektowej*. `http://home.agh.edu.pl/ pszwed/wiki/doku.php?id=amo:rup_tailored`. Akademia GórniczoHutnicza. 2015.

[7] K. Zmitrowicz. *Analiza wymagań funkcjonalnych*. `http://wymagania.net/bazawiedzy/36bazawiedzy/88analizawymagan-funkcjonalnych`.

[8] H. Wesołowska. *Jak opisywać przypadki użycia?* `http://analizait.pl/2012/jakopisywacprzypadkiuzycia/`. 2012.

[9] *Design Patterns - MVC Pattern*. `http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm`.

[10] M. Łaguna. *Wprowadzenie do Maven'a*. `http://blog.atena.pl/wprowadzeniedomavena`. 2011.

[11] *Enterprise application projects*.
`http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.jst.j2ee.doc.user%2Ftopics%2Fcjearproj.html`. Eclipse.