



Politechnika Łódzka

Instytut Informatyki

## RAPORT Z PROJEKTU KOŃCOWEGO

## STUDIÓW PODYPLOMOWYCH

# **NOWOCZESNE APLIKACJE BIZNESOWE JAVA/JAKARTA EE**

## **SYSTEM INFORMATYCZNY WSPOMAGAJĄCY TWORZENIE RAPORTÓW BEZPIECZEŃSTWA PRODUKTÓW KOSMETYCZNYCH**

**Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej**  
**Opiekun:** dr inż. Mateusz Smoliński  
**Sluchacz:** Aleksandra Kemona

Łódź, 13.11.2021



Instytut Informatyki

90-924 Łódź, ul. Wólczańska 215, budynek B9  
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

# Spis treści

1 Cel i zakres projektu.....	3
2 Założenia projektu.....	4
2.1 Wersje zastosowanych technologii i narzędzi.....	4
2.2 Wymagania funkcjonalne.....	4
2.3 Wymagania niefunkcjonalne.....	5
2.4 Przypadki użycia dla różnych poziomów dostępu.....	5
2.4.1 Tabela krzyżowa poziomów dostępu i przypadków użycia.....	5
2.4.2 Diagramy przypadków użycia.....	6
2.4.3 Opis przypadków użycia.....	8
3 Realizacja projektu.....	25
3.1 Warstwa składowania danych.....	25
3.1.1 Model relacyjnej bazy danych.....	25
3.1.2 Mapowanie obiektowo relacyjne.....	26
3.2 Realizacja przykładowego CRUD'a.....	28
3.2.1 Wyświetlanie listy kosmetyków.....	29
3.2.2 Tworzenie kosmetyku.....	31
3.2.3 Edycja kosmetyku.....	34
3.2.4 Usuwanie kosmetyku.....	36
3.3 Warstwa logiki biznesowej.....	38
3.3.1 Sesyjne komponenty EJB.....	38
3.3.2 Mechanizm ochrony spójności danych.....	39
3.3.3 Rejestrowanie zdarzeń w dziennikach, kontrola odpowiedzialności.....	41
3.3.3 Kontrola dostępu.....	42
3.3.4 Obsługa błędów.....	42
3.4 Warstwa widoku.....	43
3.4.1 Wzorzec projektowy DTO.....	44
3.4.2 Ujednolicony interfejs użytkownika.....	44
3.4.3 Internacjonalizacja.....	45
3.4.4 Uwierzytelnienie i autoryzacja.....	47
3.4.5 Walidacja danych.....	48
3.4.6 Obsługa błędów.....	48
3.5 Instrukcja wdrożenia.....	48
3.5.1 Utworzenie bazy danych.....	49
3.5.2 Konfiguracja obszaru bezpieczeństwa na serwerze.....	49
3.5.3 Wgranie aplikacji na serwer aplikacyjny i uruchomienie aplikacji.....	50
3.5.4 Umieszczenie w bazie struktur baz danych i danych inicjujących.....	51
3.6 Podsumowanie.....	51
4 Źródła.....	53

# 1 Cel i zakres projektu

Celem pracy jest stworzenie wielodostępnego systemu informatycznego wspomagającego tworzenie raportów bezpieczeństwa produktów kosmetycznych. Proces przygotowania oceny bezpieczeństwa kosmetyku z wykorzystaniem tego systemu rozpoczynać będzie wprowadzenie przez Handlowca produktu kosmetycznego (zwanego dalej kosmetykiem) do oceny. Każdy kosmetyk przypisany ma numer zamówienia, przy czym na jedno zamówienie może składać się kilka kosmetyków do oceny. Ponadto każdy kosmetyk w momencie wprowadzania go musi mieć podaną unikalną nazwę, przypisaną kategorię oraz podany skład (lista surowców wymienionych po przecinku). Na podstawie wybranej kategorii system automatycznie przypisywał będzie do kosmetyku wymagane analizy, a na podstawie składu generowany będzie opis toksykologiczny. Wprowadzony kosmetyk będzie mógł zostać zaakceptowany do oceny przez Safety Assessora (zwanego dalej Assessorem), który będzie mógł też zrezygnować z oceny. Przygotowanie raportu bezpieczeństwa w dedykowanej formacie, oraz przesłanie go klientowi odbywać się będzie poza systemem.

System informatyczny będzie złożony ze stanowej aplikacji internetowej wykonanej w technologii Java Enterprise Edition oraz relacyjnej bazy danych. Zastosowany zostanie model trójwarstwowy umożliwiający rozdzielenie warstw widoku, logiki biznesowej oraz składowania danych w relacyjnej bazie danych. Wszystkie strony aplikacji będą posiadały ujednolicony graficzny interfejs użytkownika. Środowiskiem programistycznym użytym do stworzenia systemu w języku Java będzie program NetBeans IDE, środowiskiem uruchomieniowym serwer Payara, jako system zarządzania bazami danych wybrano JavaDB.

Funkcjonalności dostarczone użytkownikowi przez system będą się różniły w zależności od przypisanego do konta użytkownika poziomu dostępu. Aby móc skorzystać z aplikacji, użytkownik musi dysponować współczesną przeglądarką internetową oraz posiadać indywidualne konto założone w systemie informatycznym. Obsługa programu będzie możliwa z wykorzystaniem interfejsu użytkownika w polskiej lub angielskiej wersji językowej w zależności od preferencji językowych ustawionych w przeglądarce internetowej. System będzie umożliwiał jednoczesny dostęp do danych wielu uwierzytelnionym użytkownikom, dzięki zastosowaniu przetwarzania transakcyjnego OLTP (ang. On-Line Transactional Processing) oraz blokad optymistycznych.

Zakres projektu:

- Opracowanie wymagań biznesowych tworzonego systemu.
- Wybór stosu technologicznego.
- Identyfikacja ról i przypadków użycia.
- Zaprojektowanie modelu relacyjnej bazy danych i wykonanie statycznych struktur.
- Zaplanowanie architektury systemu.
- Przygotowanie danych inicjujących dla relacyjnej bazy danych.
- Zaprojektowanie i implementacja aplikacji.
- Uruchomienie systemu w środowisku serwera aplikacyjnego i systemu zarządzania bazami danych, weryfikacja poprawności działania systemu.

## 2 Założenia projektu

W rozdziale przedstawione zostały założenia projektowe z uwzględnieniem wersji stosowanych technologii i narzędzi, założeń funkcjonalnych i niefunkcjonalnych systemu, oraz przypadków użycia wraz z poziomami dostępu.

### 2.1 Wersje zastosowanych technologii i narzędzi

Stos technologiczny użyty przy realizacji projektu:

- język programowania Java 11 [4];
- ziarna CDI (ang. Context and Dependency Injection) w wersji 1.2;
- Kontener EJB (ang. Enterprise JavaBeans) 3.2 [1];
- JSF (ang. Java Server Faces) w wersji 2.3;[7]
- JTA (ang. Java Transaction API) 1.2;
- BootsFaces w wersji 1.5.0;[11]
- Project Lombok w wersji 1.18.20;[10]
- JPA (ang. Java Persistence API) 2.1 [3];

Środowisko rozwojowe:

- NetBeans IDE w wersji 12 [2];
- Maven w wersji 3.6;
- platforma programistyczna JEE (ang. Java Enterprise Edition) w wersji 7.0;
- Modelio Open Source w wersji 4.1

Środowisko uruchomieniowe:

- serwer aplikacyjny Payara w wersji 5.183 [8];
- Java DB (Apache Derby) odpowiedzialna za składowanie danych w wersji 10.14.2.0;

### 2.2 Wymagania funkcjonalne

Realizowany system informatyczny będzie posiadał 5 poziomów dostępu. W założeniach przyjęto, że każdy użytkownik systemu musi dysponować indywidualnym kontem, posiadającym przypisany dokładnie jeden poziom dostępu. Uwierzytelnienia będzie odbywało się poprzez poświadczenia: login i hasło. Każdemu z poziomów zostały przypisane inne zbiory przypadków użycia.

- **Gość** – ma możliwość utworzenia własnego konta, zalogowania się oraz zresetowania hasła.
- **Administrator** – zarządza kontami wszystkich użytkowników, może je aktywować i dezaktywować. Ponadto dla nowo utworzonych kont weryfikuje dane użytkowników na podstawie dokumentu tożsamości i potwierdza je w systemie. W momencie potwierdzenia konta w systemie, Administrator przypisuje mu typ poziomu dostępu. Nowe konto Administratora tworzy użytkownik z poziomem dostępu Administrator.
- **Handlowiec** – zarządza zleceniami na raporty, może wyświetlić listę kosmetyków dodać kosmetyk do oceny, edytować go lub usunąć. Ma także dostęp do szczegółów kosmetyku.

- **Laborant** – ma dostęp do listy kosmetyków wprowadzonych przez Handlowca oraz do szczegółów każdego kosmetyku. Może edytować wyniki badań. Ma również dostęp do listy analiz, które może dodawać i edytować.
- **Assessor** - ma dostęp do listy kosmetyków wprowadzonych przez Handlowca oraz do szczegółów każdego kosmetyku. Może przyjąć kosmetyk do oceny lub zrezygnować z oceny wcześniej wybranego kosmetyku. Zarządza kategoriami produktu oraz surowcami. Może wyświetlać listy tych obiektów, dodawać je i edytować. Tak jak Laborant ma dostęp do listy analiz, które może dodawać i edytować.

## 2.3 Wymagania niefunkcjonalne

System będzie spełniał następujące wymagania niefunkcjonalne:

- Aplikacja będzie zbudowana jako monolit w architekturze trójwarstwowej, z podziałem na warstwę widoku, warstwę logiki biznesowej i warstwę składowania danych,
- System będzie wymagał uwierzytelnienia, które będzie odbywało się poprzez poświadczenie: login i hasło.
- Wzorce do uwierzytelniania będą przechowywane w relacyjnej bazie danych. Hasła będą przechowywane w formie skrótów obliczonych algorytmem SHA-256. Wymagana będzie unikalność loginów dla kont użytkownika.
- Dane biznesowe aplikacji przechowywane będą w relacyjnej bazie danych.
- Użytkownik będzie mógł korzystać z konta jeżeli będzie ono posiadało przypisany poziom dostępu, będzie potwierdzone i aktywne.
- Model relacyjny odwzorowany zostanie z wykorzystaniem standardu mapowania obiektowo-relacyjnego JPA.
- Spójność danych w systemie zapewniona zostanie za pomocą mechanizmów transakcji i blokad optymistycznych.
- System będzie zapewniał mechanizm zgłaszania komunikatów o występujących zdarzeniach i gromadzenia ich w dzienniku zdarzeń.
- Interfejs użytkownika zapewni dynamicznie generowane strony WWW (ang. World Wide Web) dostępne przez współczesną przeglądarkę internetową.
- System będzie zapewniał internacjonalizację interfejsu użytkownika oraz zgłaszanych komunikatów. Dostępne języki to polski i angielski.

## 2.4 Przypadki użycia dla różnych poziomów dostępu

Poniższy rozdział prezentuje szczegółowe informacje na temat występujących w aplikacji przypadków użycia w zależności od poziomu dostępu przypisanego do konta użytkownika systemu.

### 2.4.1 Tabela krzyżowa poziomów dostępu i przypadków użycia.

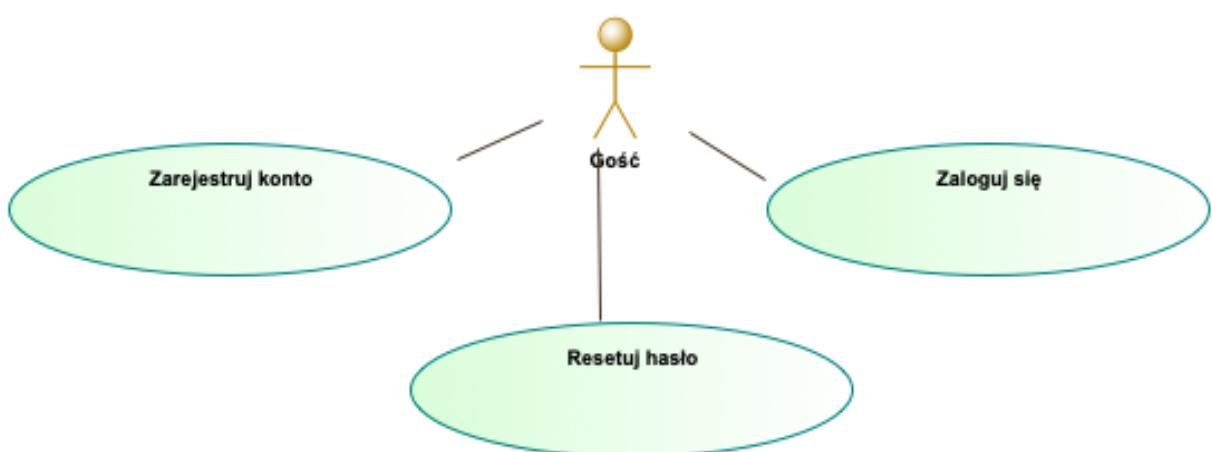
W tabeli 1 zaprezentowano występujące w aplikacji przypadki użycia wraz z ich przypisaniem do poziomów dostępu.

Tabela 1: Przypadki użycia w zależności od poziomu dostępu użytkownika systemu.

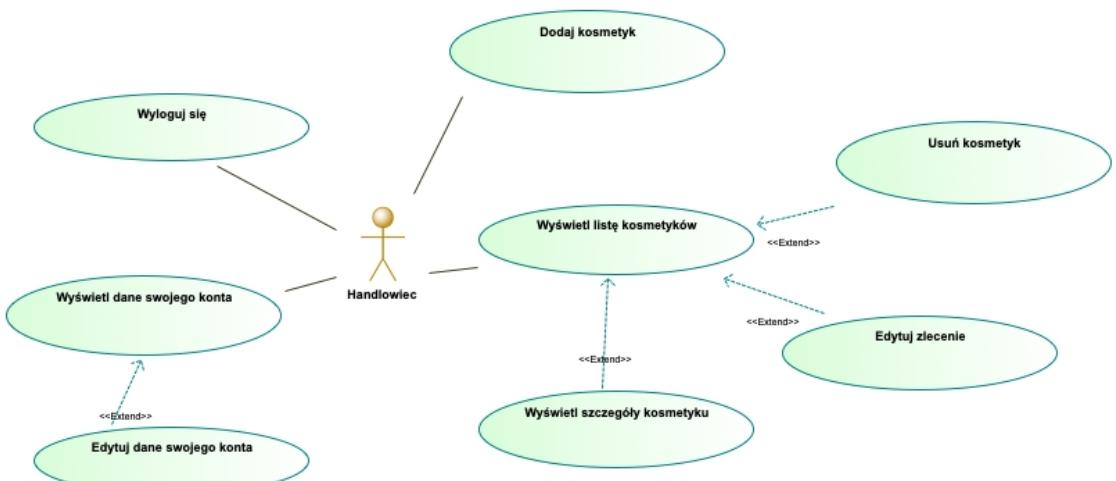
Lp.	Przypadek użycia	Gość	Handlowiec	Laborant	Assessor	Administrator
<b>Obsługa kont</b>						
1	Zarejestruj konto	x				
2	Zaloguj się	x				
3	Resetuj hasło	x				
4	Wyloguj się		x	x	x	x
5	Wyświetl dane swojego konta		x	x	x	x
6	Edytuj dane swojego konta		x	x	x	x
7	Utwórz konto użytkownika					x
8	Wyświetl listę kont użytkowników					x
9	Potwierdź konto użytkownika					x
10	Aktywuj konto użytkownika					x
11	Dezaktywuj konto użytkownika					x
12	Edytuj konto użytkownika					x
<b>Obsługa kosmetyków do oceny bezpieczeństwa</b>						
13	Wyświetl listę kosmetyków	x		x	x	
14	Dodaj kosmetyk	x				
15	Edytuj kosmetyk	x				
16	Usuń kosmetyk	x				
15	Przyjmij kosmetyk do oceny					x
16	Zrezygnuj z oceny					x
17	Wyświetl szczegóły kosmetyku	x		x	x	
<b>Szczegóły oraz obsługa kosmetyku</b>						
18	Wyświetl listę kategorii kosmetyków					x
19	Dodaj kategorię kosmetyku					x
20	Edytuj kategorię kosmetyku					x
21	Wyświetl listę analiz				x	x
22	Dodaj analizę			x		x
23	Edytuj analizę			x		x
24	Edytuj wynik badania			x		
25	Wyświetl listę surowców					x
26	Dodaj surowiec					x
27	Edytuj surowiec					x

#### 2.4.2 Diagramy przypadków użycia

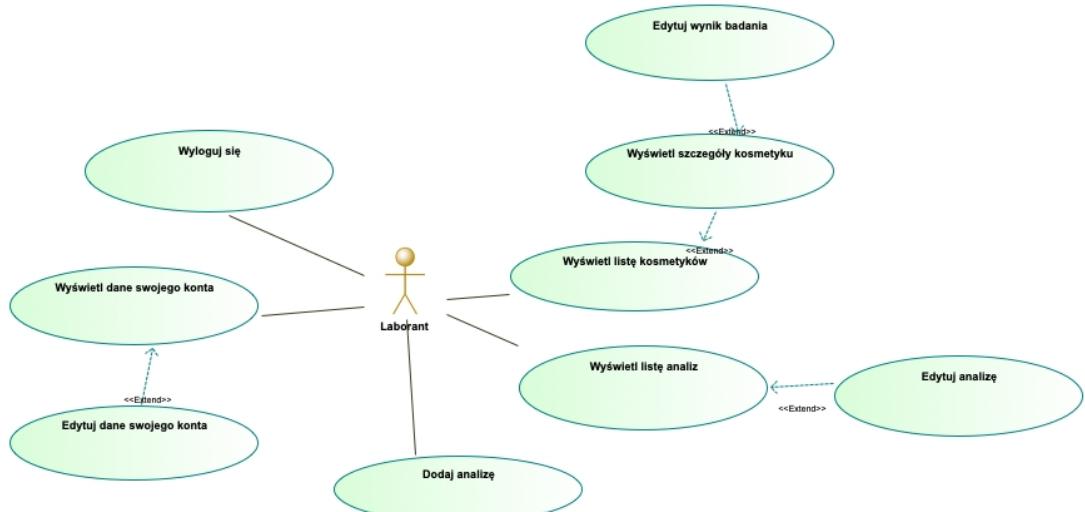
Funkcjonalności oferowane przez system informatyczny przedstawiono w formie diagramów przypadków użycia dla poziomów dostępu: Gość (rysunek 1), Handlowiec (rysunek 2), Laborant (rysunek 3), Assessor (rysunek 4), Administrator (rysunek 5).



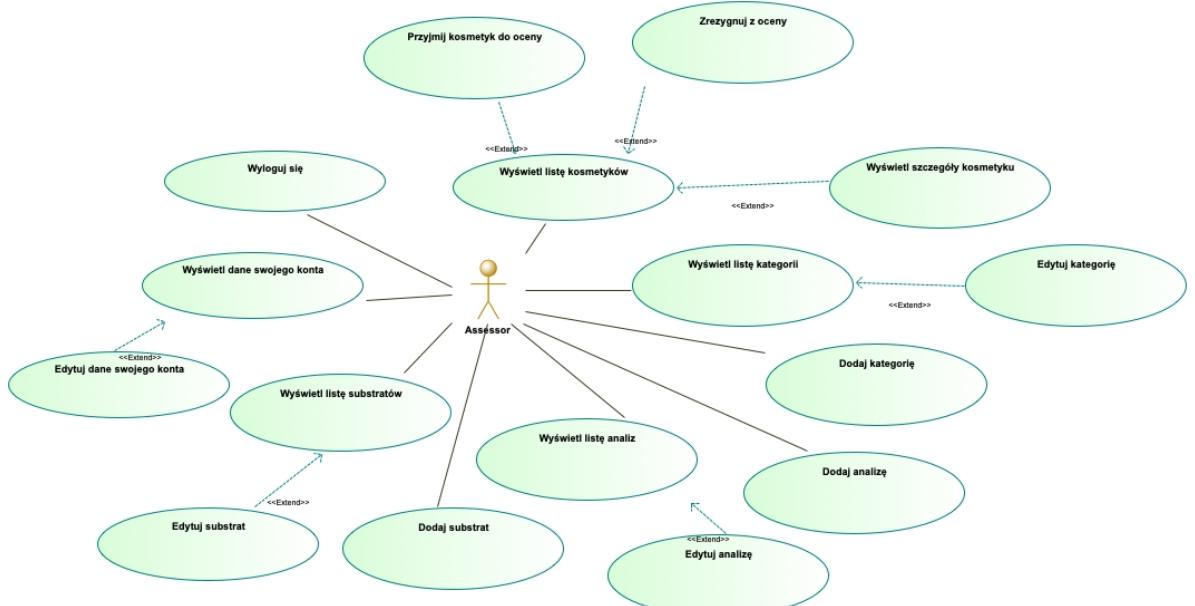
Rysunek 1: Diagram przypadków użycia dla poziomu dostępu: Gość



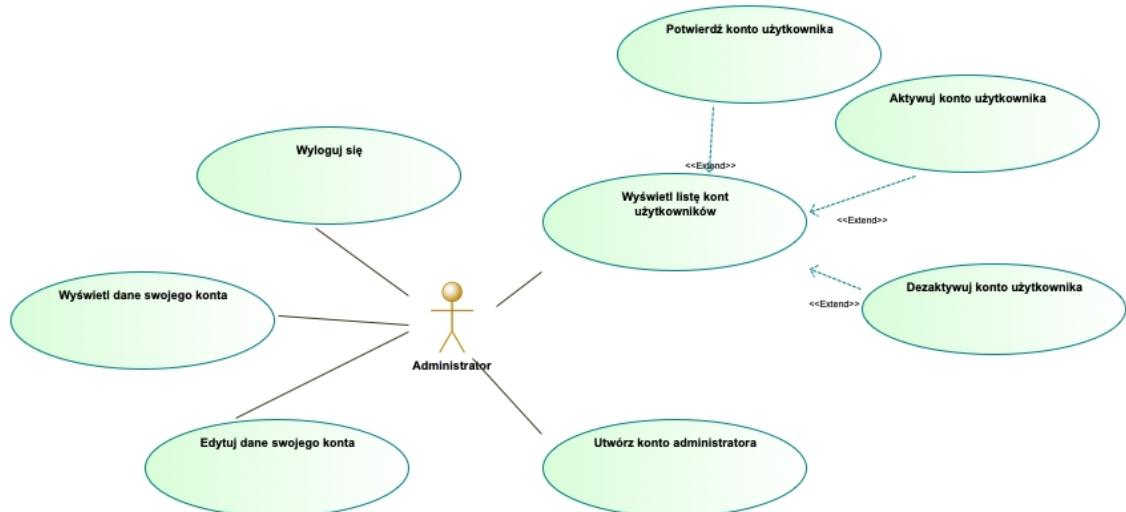
Rysunek 2: Diagram przypadek użycia dla poziomu dostępu: Handlowiec



Rysunek 3: Diagram przypadek użycia dla poziomu dostępu: Laborant



Rysunek 4: Diagram przypadek użycia dla poziomu dostępu: Assessor



Rysunek 5: Diagram przypadków użycia dla poziomu dostępu: Administrator

### 2.4.3 Opis przypadków użycia

Poniższy rozdział zawiera opis wszystkich przypadków użycia występujących w aplikacji. Każdy z przypadków zawiera charakterystykę przebiegu scenariusza głównego oraz scenariuszy błędów, które wynikają z naruszenia ustalonych reguł biznesowych. Na rysunkach przedstawiających formularze interfejsu użytkownika, etykiety wymaganych pól oznaczone zostały za pomocą gwiazdki.

**Zarejestruj konto** – jest to przypadek użycia dostępny tylko dla nieuwierzytelnionego użytkownika (Gość). Dane niezbędne do zarejestrowania konta wprowadzane są do formularza, a następnie po naciśnięciu przycisku „Rejestruj konto” następuje walidacja wprowadzonych danych. Jeśli weryfikacja potwierdzi poprawność wprowadzonych informacji, są one zapisywane w bazie. Na tym etapie konto nie ma przydzielonego poziomu dostępu.

Aktor	Użytkownik nieuwierzytelniony (Gość)
Scenariusz główny	Użytkownik wypełnia formularz poprawnie i konto zostaje utworzone.
Scenariusze błędów	<ol style="list-style-type: none"> <li>Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz rejestracji zgłasza błąd „Pole nie może być puste”.</li> <li>Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz rejestracji zgłasza błąd: „Login nie może być krótszy niż 6 znaków”, „Dozwolone są tylko znaki alfanumeryczne”, „Hasło nie może być krótsze niż 8 znaków, itp.”</li> <li>Wprowadzony zostaje login, który istnieje już w bazie – formularz rejestracji zgłasza błąd: „Podany login już istnieje”.</li> <li>Użytkownik w polu Adres e-mail wprowadza adres e-mail, który istnieje już w bazie danych – formularz zgłasza błąd „Podany e-mail już istnieje”.</li> </ol>

5. Użytkownik wprowadza w polu „Powtóż hasło” inny ciąg znaków niż w polu „Hasło” – formularz rejestracji zgłasza błąd: „Podane hasła różnią się”.

6. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby rejestracji w późniejszym czasie.

**Zaloguj się** – przypadek użycia polega na podaniu przez użytkownika unikalnego loginu oraz odpowiedniego dla danego konta użytkownika, aktualnego hasła. Po kliknięciu przycisku „Zaloguj” aplikacja sprawdza, czy konto o podanym loginie istnieje w bazie danych, czy jest aktywne oraz czy podane hasło zgadza się z istniejącym w bazie, poprzez porównanie skrótu SHA-256 wprowadzonego hasła z wzorcem skrótu hasła przechowywanym w bazie. Spełnienie wszystkich warunków pozwala użytkownikowi uzyskać dostęp do funkcjonalności przypisanych do jego poziomu dostępu.

Aktor	Użytkownik nieuwierzytelny (Gość)
Scenariusz główny	Użytkownik wypełnia poprawnie pola „Login” oraz „Hasło”, po czym zostaje zalogowany do aplikacji.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Podano niepoprawny login – formularz logowania zgłasza błąd: „Niepoprawna nazwa użytkownika (login) lub hasło”.</li><li>2. Wprowadzone zostało niepoprawne hasło – formularz logowania zgłasza błąd: „Niepoprawna nazwa użytkownika (login) lub hasło”.</li><li>3. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li></ol>

**Zresetuj hasło** – jest funkcjonalnością dostępną dla nieuwierzytelnego użytkownika, który zapomniał swojego aktualnego hasła i chce ustawić nowe hasło do posiadanej w systemie konta. Proces przebiega w dwóch etapach, w pierwszym użytkownik podaje swój login i naciska przycisk „Resetuj hasło”. Aplikacja weryfikuje w bazie czy podany login istnieje, po czym następuje przekierowanie do strony zawierającej pytanie weryfikacyjne. Użytkownik musi wprowadzić odpowiedź na pytanie, która podana została przy tworzeniu konta. Następnie wprowadza hasło, które musi zostać powtórzone, tak jak przy rejestracji konta.

Aktor	Użytkownik nieuwierzytelny (Gość)
Scenariusz główny	Użytkownik wprowadza poprawny login, po czym udziela właściwej odpowiedzi na pytanie oraz wpisuje poprawnie hasło i je powtarza. Po wciśnięciu przycisku „Resetuj hasło” następuje zapisanie nowego hasła w bazie.

- Scenariusze błędów
1. Podano niepoprawny login – formularz resetowania hasła zgłasza błąd: „Konto o podanym loginie nie zostało odnalezione”.
  2. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz resetowania hasła zgłasza błąd „Pole nie może być puste”.
  3. Odpowiedź na pytanie nie jest zgodna z odpowiedzią zapisaną w bazie – formularz resetowania hasła zgłasza błąd: „Odpowiedź niepoprawna”.
  4. Użytkownik wprowadza w polu „Powtórz hasło” inny ciąg znaków niż w polu „Hasło” – formularz resetowania hasła zgłasza błąd: „Podane hasła różnią się”.
  5. Hasło wprowadzone przez użytkownika jest niezgodne z walidacją pól formularza – formularz resetowania hasła zgłasza błąd: „Hasło musi zawierać co najmniej 8 znaków, w tym przynajmniej po jednej cyfrze, znaku specjalnym, małej i wielkiej literze”.
  6. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Wyloguj się** – zalogowany użytkownik po naciśnięciu przycisku „Wyloguj się”, staje się użytkownikiem nieuwierzytelnionym i traci dostęp do funkcjonalności przypisanych do przydzielonego mu poziomu dostępu, jednocześnie zyskując funkcjonalności odpowiadające użytkownikowi nieuwierzytelnionemu. Po 15 minutach braku aktywności następuje automatyczne wylogowanie zalogowanego użytkownika.

- |                    |  |
|--------------------|--|
| Aktor              | Użytkownik zalogowany (Administrator, Assessor, Laborant, Handlowiec)  |
| Scenariusz główny  | Użytkownik wylogowuje się z aplikacji.   |
| Scenariusze błędów | <ol style="list-style-type: none"> <li>1. Użytkownik nie wylogowuje się, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li> </ol> |

**Wyświetl dane swojego konta** – przypadek użycia pozwala zalogowanemu użytkownikowi na wgląd w informacje o własnym koncie. Dane wyświetlane są w oparciu o poziom dostępu przypisany do danego typu konta.

Aktor	Użytkownik zalogowany (Administrator, Assessor, Laborant, Handlowiec)
Scenariusz główny	Użytkownik wyświetla dane swojego konta.
Scenariusze błędów	1. Dane konta nie zostają wyświetlane, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Edytuj dane swojego konta** – przypadek użycia umożliwiający zmianę edytowalnych danych własnego konta.

Aktor	Użytkownik zalogowany (Administrator, Assessor, Laborant, Handlowiec)
Scenariusz główny	Użytkownik pobiera dane swojego konta, aktualizuje dane wypełniając formularz poprawnie. Edytowane dane zapisywane są w bazie.
Scenariusze błędów	<ol style="list-style-type: none"> <li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz zgłasza błąd „Pole nie może być puste”.</li> <li>2. Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz edycji konta zgłasza błąd, np.: „Dozwolone są tylko znaki alfanumeryczne”.</li> <li>3. Użytkownik w polu Adres e-mail wprowadza adres e-mail, który istnieje już w bazie danych – formularz zgłasza błąd „Podany e-mail już istnieje”.</li> <li>4. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłoszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li> </ol>

**Utwórz konto Użytkownika** – przypadek użycia dostępny jest wyłącznie dla poziomu dostępu Administrator. Polega na wprowadzeniu do odpowiedniego formularza danych, które są niezbędne do utworzenia konta Użytkownika. Naciśnięcie przycisku „Utwórz” prowadzi do zapisania danych nowego konta do bazy danych.

Aktor	Użytkownik z nadaną rolą Administrator.
Scenariusz główny	Użytkownik wypełnia formularz poprawnie i zostaje utworzone nowe konto Użytkownika.

- Scenariusze błędów
1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz rejestracji zgłasza błąd „Pole nie może być puste”.
  2. Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz rejestracji zgłasza błąd: „Login nie może być krótszy niż 6 znaków”, „Dzwolone są tylko znaki alfanumeryczne”, itp.
  3. Wprowadzony zostaje login, który istnieje już w bazie – formularz rejestracji zgłasza błąd: „Podany login już istnieje”.
  4. Użytkownik wprowadza w polu „Powtórz hasło” inny ciąg znaków niż w polu „Hasło” – formularz rejestracji zgłasza błąd: „Podane hasła różnią się”.
  5. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszaną jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Wyświetl listę kont użytkowników** - przypadek użycia dostępny jest wyłącznie dla poziomu dostępu Administrator. Po naciśnięciu przycisku „Panel administratora” wszystkie konta użytkowników pobierane są z bazy i wyświetlane w formie listy (rysunek 6). Administrator może wybrać konto z listy i wykonać akcję z nim związaną np.: potwierdzić niepotwierdzone konto oraz aktywować lub dezaktywować konto.

Lista kont użytkowników									
Strona główna									
Pokaż		10		pozycji		Szukaj:			
Login	↑	Imię	↑	Nazwisko	↑	Email	↑	Typ konta	↑
administrator1		Jan		Kowalski		jan.kowalski@mail.com		Administrator	
assessor1		Tomek		Malinowski		tomasz.malinowski@mail.com		Assessor	
sales1		Anna		Nowak		anna.nowak@mail.com		Sales	
laboratory1		Katarzyna		Brzozowska		katarzyna.Brzozowska@mail.com		LabTechnician	
administrator2		Magdalena		Jabłczyńska		magdalena.jabłczyńska@mail.com		Administrator	
assessor2		Paweł		Rybicki		pawel.rybicki@mail.com		Zweryfikuj konto	

Pozycje od 1 do 6 z 6 łącznie

Poprzednia 1 Następna

Rysunek 6: Lista kont użytkowników

Aktor	Użytkownik z nadaną rolą Administrator.
Scenariusz główny	Użytkownik wyświetla listę kont.
Scenariusze błędów	1. Lista kont nie zostaje wyświetlona, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Potwierdź konto użytkownika** – utworzone przez użytkownika konto nie ma przypisanego poziomu dostępu i jest niepotwierdzone. W tym przypadku użycia użytkownik z poziomem dostępu Administrator weryfikuje dane użytkownika oraz przypisuje poziom dostępu do konta. Po naciśnięciu przycisku „Potwierdź konto” użytkownik przechodzi do strony wyświetlającej wszystkie dane użytkownika oraz pole wyboru typu poziomu dostępu. Jeśli dane wyświetlane na stronie zgadzają się ze stanem faktycznym, użytkownik naciska przycisk „Zatwierdź”, co prowadzi do aktualizacji danych. Potwierdzone konto zostaje automatycznie aktywowane.

Aktor	Użytkownik z nadaną rolą Administrator.
Scenariusz główny	Użytkownik potwierdza konto w systemie, przypisując mu jednocześnie poziom dostępu.
Scenariusze błędów	<ol style="list-style-type: none"> <li>Użytkownik naciska przycisk „Potwierdź konto”, gdy konto zostało potwierdzone przez innego Administratora – formularz zgłasza błąd: „Konto zostało już potwierdzone”.</li> <li>Użytkownik naciska przycisk „Zatwierdź”, gdy konto zostało potwierdzone przez innego Administratora – formularz zgłasza błąd: „Konto zostało już potwierdzone”.</li> <li>Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłoszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li> </ol>

**Aktywuj/Dezaktywuj konto użytkownika** – aktywacja/dezaktywacja konta jest funkcjonalnością dostępną wyłącznie dla użytkownika z poziomem dostępu Administrator. Pozwala na zablokowanie użytkownikowi dostępu do aplikacji. Po zakończeniu sesji przez użytkownika, którego konto zostało zdezaktywowane, nie ma on możliwości ponownego zalogowania się do aplikacji. Po ponownej aktywacji konta przez Administratora, użytkownik ponownie ma możliwość zalogowania się. Przyciski „Aktywuj” i „Dezaktywuj” mają działanie naprzemienne i na ekranie w danym momencie może być widoczny tylko jeden z nich, w zależności od stanu konta, w związku z czym nie jest możliwe ponowne wcisnięcie tego samego przycisku.

Aktor	Użytkownik z nadaną rolą Administrator.
Scenariusz główny	Użytkownik aktualizuje wybrane konto z listy zmieniając dane w formularzu przyciskiem „Aktywuj”/„Dezaktywuj”. Aktualne dane zapisywane są w bazie.
Scenariusze błędów	<p>1. Użytkownik naciska przycisk „Aktywuj”, gdy konto zostało aktywowane przez innego Administratora – formularz zgłasza błąd: „Konto zostało już aktywowane”.</p> <p>2. Użytkownik naciska przycisk „Dezaktywuj”, gdy konto zostało dezaktywowane przez innego Administratora – formularz zgłasza błąd: „Konto zostało już dezaktywowane”.</p> <p>3. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</p>
<b>Edytuj konto użytkownika</b> – przypadek użycia dostępny dla poziomu dostępu Administrator. Funkcja stosowana jest, gdy niezbędne jest wprowadzenie zmian w danych osobowych dowolnego użytkownika. Formularz daje możliwość zmiany zarówno danych konta jak i hasła użytkownika, gdy z jakiegoś powodu użytkownik nie może tego zrobić samodzielnie. Zmiana hasła wymaga jego dwukrotnego wpisania.	
Aktor	Użytkownik z nadaną rolą Administrator.
Scenariusz główny	Użytkownik pobiera dane konta, aktualizuje je wypełniając poprawnie formularz i zapisuje w bazie.
Scenariusze błędów	<p>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz zgłasza błąd „Pole nie może być puste”.</p> <p>2. Dane wprowadzone przez użytkownika są niezgodne z waliadcją pól formularza – formularz edycji konta zgłasza błąd, np.: „Dozwolone są tylko znaki alfanumeryczne”.</p> <p>3. Użytkownik w polu Adres e-mail wprowadza adres e-mail, który istnieje już w bazie danych – formularz zgłasza błąd „Podany e-mail już istnieje”.</p> <p>4. Użytkownik wprowadza w polu „Powtórz hasło” inny ciąg znaków niż w polu „Hasło” – formularz rejestracji zgłasza błąd: „Podane hasła różnią się”.</p> <p>5. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</p>

czasie.

**Wyświetl listę kosmetyków** – przypadek użycia dostępny jest dla poziomów dostępu Handlowiec, Laborant i Assessor. Po naciśnięciu przycisku „Lista produktów do oceny” w menu rozwijanym „Produkty” wszystkie kosmetyki pobierane są z bazy i wyświetlane w formie listy. Użytkownik może wybrać produkt z listy i wykonać akcję z nim związaną np.: wyświetlić szczegóły produktu, edytować go lub usunąć.

Aktor	Użytkownik z nadaną rolą Handlowiec, Laborant i Assessor.
Scenariusz główny	Użytkownik wyświetla listę kosmetyków.
Scenariusze błędów	1. Lista kosmetyków nie zostaje wyświetlona, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

Numer zamówienia	Nazwa kosmetyku	Wybierz kategorię	Skład	Utworzone przez	Ocenione przez	Akcje
123556	Krem A	Krem	Aqua, Glycerin, Cetearyl Alcohol	A. Nowak	T. Malinowski	<span>Edytuj</span> <span>Szczegóły</span>
31247	Krem na noc	Krem	Aqua, Glycerin, Xantan Gum	A. Nowak		<span>Edytuj</span> <span>Szczegóły</span>

Rysunek 7: Lista produktów kosmetycznych do oceny

**Dodaj kosmetyk** – przypadek użycia dostępny dla poziomu dostępu Handlowiec. Po naciśnięciu przycisku „Utwórz kosmetyk” użytkownik przechodzi do formularza tworzenia kosmetyku widocznego na rysunku 8. Po naciśnięciu przycisku „Ok”, nastąpi przekierowanie do strony weryfikacji wprowadzonych danych. Po zatwierdzeniu poprawności danych, nowy kosmetyk zostanie zapisany w bazie.

Aktor	Użytkownik z nadaną rolą Handlowiec.
Scenariusz główny	Użytkownik uzupełnia poprawnie formularz, weryfikuje wprowadzone dane, a następnie po naciśnięciu przycisku „Utwórz” do bazy zostaje dodany nowy kosmetyk.
Scenariusze błędów	1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz rejestracji zgłasza błąd „Pole nie może być

puste".

2. Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz tworzenia kosmetyku zgłasza błąd: „Numer zamówienia musi być wartością liczbową”, itp.
3. Użytkownik wprowadza nazwę kosmetyku, który już istnieje w bazie – formularz tworzenia kosmetyku zgłasza błąd: „Kosmetyk o podanej nazwie już istnieje”.
4. Kosmetyk nie zostaje utworzony, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.



**Utwórz kosmetyk**

Numer zamówienia: \*

Nazwa kosmetyku: \*

Wybierz kategorię: \*

Krem

Skład: \*

Utwórz Anuluj

Rysunek 8: Formularz dodawania nowego kosmetyku

**Edytuj kosmetyk** – przypadek użycia dostępny z poziomu listy kosmetyków dla Handlowca, umożliwiający zmianę edytowalnych danych kosmetyku (rysunek 9).

Aktor	Użytkownik z nadaną rolą Handlowiec.
Scenariusz główny	Użytkownik pobiera dane kosmetyku, aktualizuje dane wypełniając formularz poprawnie. Edytowane dane zapisywane są w bazie.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz edycji kosmetyku zgłasza błąd „Pole nie może być puste”.</li><li>2. Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz edycji kosmetyku zgłasza błąd,</li></ol>

np.: „Dozwolone są tylko znaki alfanumeryczne”.

3. Użytkownik wprowadza nazwę kosmetyku, który już istnieje w bazie – formularz tworzenia kosmetyku zgłasza błąd: „Kosmetyk o podanej nazwie już istnieje”.
4. Podczas edycji kosmetyku inny użytkownik zmienił jego dane – formularz edycji kosmetyku zgłasza błąd: „Zmiany nie zostały zapisane, ponieważ edytowane dane są nieaktualne”.
5. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Edytuj kosmetyk**

• Nie można zapisać ponieważ kosmetyk został zedytowany

Numer zamówienia:  
**123556**

Nazwa:  
Krem na noc

Kategoria:  
Krem

Skład:  
Aqua, Glycerin, Cetearyl Alcohol

**OK**   **Porzuć**

Rysunek 9: Formularz edycji kosmetyku

**Usuń kosmetyk** – przypadek użycia dostępny z poziomu listy kosmetyków dla użytkownika z przypisanym poziomem dostępu Handlowiec. Przed usunięciem kosmetyku następuje przekierowanie na stronę z pytaniem potwierdzającym usunięcie kosmetyku widoczną na rysunku 10.

Aktor	Użytkownik z nadaną rolą Handlowiec.
Scenariusz główny	Użytkownik usuwa wybraną pozycję z listy kosmetyków poprzez naciśnięcie przycisku „Usuń”.
Scenariusze błędów	1. Użytkownik usuwa kosmetyk, który został wcześniej usunięty przez innego użytkownika – strona zgłasza błąd „Kosmetyk nie został odnaleziony”.

2. Podczas usuwania kosmetyku, inny użytkownik go edytował – formularz zgłasza błąd: „Kosmetyk nie może być usunięty, ponieważ jego dane są nieaktualne”.

3. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

## Potwierdź usunięcie kosmetyku

Numer zamówienia:  
2135

Nazwa:  
Krem A

Kategoria:  
Krem

Skład:  
woda, gliceryna

Usuń kosmetyk

Porzuć

Rysunek 10: Strona potwierdzenia usunięcia kosmetyku

**Przyjmij kosmetyk do oceny/Zrezygnuj z kosmetyku do oceny** – przyjęcie kosmetyku do oceny jest funkcjonalnością dostępną wyłącznie dla użytkownika z poziomem dostępu Assessor. Po naciśnięciu przycisku „Oceń” sygnatura Assessora przypisywana jest do produktu. Wybrany do oceny produkt przestaje być dostępny do przyjęcia dla innych Assessorów. Po zrezygnowaniu z oceny kosmetyk jest ponownie dostępny do przyjęcia dla wszystkich Użytkowników z odpowiednim poziomem dostępu. Przyciski „Oceń” oraz „Zrezygnuj” mają działanie naprzemienne i na ekranie w danym momencie może być widoczny tylko jeden z nich, w zależności od stanu konta, w związku z czym nie jest możliwe ponowne wcisnięcie tego samego przycisku, przy czym przycisk rezygnacji widoczny jest tylko dla oceniającego produkt Assessora.

Aktor	Użytkownik z nadaną rolą Assessor.
Scenariusz główny	Użytkownik aktualizuje wybrany z listy kosmetyk zmieniając dane w formularzu przyciskiem „Oceń”/„Zrezygnuj”. Aktualne dane zapisywane są w bazie.
Scenariusze błędów	<ol style="list-style-type: none"><li>Użytkownik naciska przycisk „Oceń”, gdy kosmetyk został przyjęty do oceny przez innego Assessora – formularz zgłasza błąd: „Kosmetyk został już przypisany”.</li><li>Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li></ol>

**Wyświetl szczegóły kosmetyku** – przypadek użycia pozwala użytkownikowi z poziomem dostępu Handlowiec, Laborant lub Assessor na wgląd w szczegółowe informacje o produkcie. Szczegóły kosmetyku zawierają zakres informacji rozszerzony względem tabeli zbiorczej o wymagane analizy, wyniki badań oraz opis toksykologiczny (rysunek 11).

## Krem B

Numer zamówienia:  
**123556**

Kategoria:  
**Krem**

Skład:  
**Aqua, Glycerin, Cetearyl Alcohol**

Analiza	Wynik badania	Minimum	Maksimum	Czy w normie
GC-MS	10 <button style="border: none; padding: 0 5px;">Edytuj</button>	1	100	Wynik w normie
Mikrobiologia	20 <button style="border: none; padding: 0 5px;">Edytuj</button>	1	100	Wynik w normie
HPLC	200 <button style="border: none; padding: 0 5px;">Edytuj</button>	2	100	<b>Wynik poza skalą</b>

[Excel](#) [PDF](#)

Surowiec	Opis toksykologiczny
Cetearyl Alcohol	rozpuszczalnik
Glycerin	rozpuszczalnik
Aqua	rozpuszczalnik polarny

[Excel](#) [PDF](#)

[Wróć do listy](#)

Rysunek 11: Strona szczegółów kosmetyku

Aktor	Użytkownik z poziomem dostępu Assessor, Laborant, Handlowiec
Scenariusz główny	Użytkownik wyświetla szczegółowe dane kosmetyku.
Scenariusze błędów	1. Dane kosmetyku nie zostają wyświetlone, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Edytuj wynik badania** - przypadek użycia dostępny dla poziomu dostępu Laborant. Pozwala na wprowadzenie/zmienienie wyniku badania. Domyślnym wynikiem badania jest 0. Po naciśnięciu przycisku „Edytuj” pole wyniku staje się polem edytowalnym. Po wprowadzeniu nowej wartości i naciśnięciu przycisku „Aktualizuj” następuje aktualizacja danej w bazie.

Aktor	Użytkownik z nadaną rolą Laborant.
Scenariusz główny	Użytkownik naciska przycisk „Edytuj” wprowadza poprawnie nową wartość w polu, po naciśnięciu przycisku „Aktualizuj” następuje aktualizacja danej w bazie.
Scenariusze błędów	<ol style="list-style-type: none"> <li>Pole wartości wyniku badania nie zawiera żadnej wartości – formularz edycji kosmetyku zgłasza błąd „Pole nie może być puste”.</li> <li>Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz edycji kosmetyku zgłasza błąd, np.: „Dzwolone są tylko wartości liczbowe”.</li> <li>Podczas edycji wyniku inny użytkownik zmienił jego wartość – formularz edycji kosmetyku zgłasza błąd: „Wprowadzone zmiany nie zostały zapisane, ponieważ edytowane dane są nieaktualne”.</li> <li>Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby.</li> </ol>

**Wyświetl listę kategorii kosmetyków** - przypadek użycia dostępny jest dla poziomu dostępu Assessor. Po naciśnięciu przycisku „Lista kategorii” w menu rozwijanym „Produkty” wszystkie kategorie pobierane są z bazy i wyświetlane w formie listy (rysunek 12). Użytkownik może wybrać kategorię z listy i wykonać akcję z nią związaną: edytować ją lub usunąć.

Lista kategorii			
<input type="button" value="Utwórz kategorię"/>		Szukaj: <input type="text"/>	
Nazwa kategorii	Analizy wymagane	Akcje	
Krem	GC-MS, Mikrobiologia, HPLC	<input type="button" value="Edytuj"/>	
Tonik	Mikrobiologia, Dermatologia	<input type="button" value="Edytuj"/>	
Serum	Mikrobiologia, HPLC	<input type="button" value="Edytuj"/>	
Płyn micelarny	GC-MS, Mikrobiologia, Dermatologia, HPLC	<input type="button" value="Edytuj"/>	
nowa kategoria	Mikrobiologia, HPLC	<input type="button" value="Edytuj"/>	

Pozycje od 1 do 5 z 5 łącznie

Poprzednia 1 Następna

Rysunek 12: Lista kategorii kosmetyku

Aktor	Użytkownik z nadaną rolą Assessor.
Scenariusz główny	Użytkownik wyświetla listę kategorii.
Scenariusze błędów	<ol style="list-style-type: none"> <li>Lista kategorii nie zostaje wyświetlona, brak połączenia z bazą</li> </ol>

danych – do dzienników zdarzeń aplikacji zgłoszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Dodaj kategorię kosmetyku** – przypadek użycia dostępny dla poziomu dostępu Assessor. Po naciśnięciu przycisku „Dodaj kategorię” użytkownik przechodzi do formularza tworzenia kategorii. Po wprowadzeniu danych i naciśnięciu przycisku „Ok”, nastąpi przekierowanie do strony weryfikacji wprowadzonych danych. Po zatwierdzeniu poprawności danych, nowa kategoria zostanie zapisana w bazie.

Aktor	Użytkownik z nadaną rolą Assessor.
Scenariusz główny	Użytkownik uzupełnia poprawnie formularz, weryfikuje wprowadzone dane, a następnie po naciśnięciu przycisku „Utwórz” do bazy zostaje dodana nowa kategoria.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz rejestracji zgłasza błąd „Pole nie może być puste”.</li><li>2. Użytkownik wprowadza nazwę kategorii, która już istnieje w bazie – formularz tworzenia analizy zgłasza błąd: „Kategoria o podanej nazwie już istnieje”.</li><li>3. Kategoria nie zostaje utworzona, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li></ol>

**Edytuj kategorię kosmetyku** – przypadek użycia dostępny z poziomu listy kategorii dla Assessora, umożliwiający zmianę edytowalnych danych kategorii.

Aktor	Użytkownik z nadaną rolą Assessor.
Scenariusz główny	Użytkownik pobiera dane analizy, aktualizuje je wypełniając formularz poprawnie. Edytowane dane zapisywane są w bazie.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz edycji kosmetyku zgłasza błąd „Pole nie może być puste”.</li><li>2. Użytkownik wprowadza nazwę kategorii, która już istnieje w bazie – formularz tworzenia analizy zgłasza błąd: „Kategoria o podanej nazwie już istnieje”.</li><li>3. Podczas edycji kategorii inny użytkownik zmienił jej dane –</li></ol>

formularz edycji analizy zgłasza błąd: „Wprowadzone zmiany nie zostały zapisane, ponieważ edytowane dane są nieaktualne”.

4. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłoszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby.

**Wyświetl listę analiz** – przypadek użycia dostępny jest dla poziomów dostępu Laborant i Assessor. Po naciśnięciu przycisku „Lista analiz” w menu rozwijanym „Produkty” wszystkie analizy pobierane są z bazy i wyświetlane w formie listy. Użytkownik może wybrać analizę z listy i ją edytować.

Aktor	Użytkownik z nadaną rolą Laborant i Assessor.
Scenariusz główny	Użytkownik wyświetla listę analiz.
Scenariusze błędów	1. Lista analiz nie zostaje wyświetlona, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Dodaj analizę** – przypadek użycia dostępny dla poziomów dostępu Laborant i Assessor. Po naciśnięciu przycisku „Dodaj analizę” użytkownik przechodzi do formularza tworzenia analizy. Po wprowadzeniu danych i naciśnięciu przycisku „Ok”, nastąpi przekierowanie do strony weryfikacji wprowadzonych danych. Po zatwierdzeniu poprawności danych, nowa analiza zostanie zapisana w bazie.

Aktor	Użytkownik z nadaną rolą Laborant i Assessor.
Scenariusz główny	Użytkownik uzupełnia poprawnie formularz, weryfikuje wprowadzone dane, a następnie po naciśnięciu przycisku „Utwórz” do bazy zostaje dodana nowa analiza.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz zgłasza błąd „Pole nie może być puste”.</li><li>2. Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz tworzenia analizy zgłasza błąd: „Wartość minimalna musi być wartością liczbową”, itp.</li><li>3. Użytkownik wprowadza nazwę analizy, która już istnieje w bazie – formularz tworzenia analizy zgłasza błąd: „Analiza o podanej nazwie już istnieje”.</li><li>4. Analiza nie zostaje utworzona, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis</li></ol>

o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.

**Edytuj analizę** - przypadek użycia dostępny z poziomu listy analiz dla Laboranta i Assessora, umożliwiający zmianę edytowalnych danych analizy.

Aktor	Użytkownik z nadaną rolą Laborant lub Assessor.
Scenariusz główny	Użytkownik pobiera dane analizy, aktualizuje je wypełniając formularz poprawnie. Edytowane dane zapisywane są w bazie.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz edycji kosmetyku zgłasza błąd „Pole nie może być puste”.</li><li>2. Dane wprowadzone przez użytkownika są niezgodne z walidacją pól formularza – formularz tworzenia analizy zgłasza błąd: „Wartość minimalna musi być wartością liczbową”, itp.</li><li>3. Użytkownik wprowadza nazwę analizy, która już istnieje w bazie – formularz edycji analizy zgłasza błąd: „Analiza o podanej nazwie już istnieje”.</li><li>4. Podczas edycji analizy inny użytkownik zmienił jej dane – formularz edycji analizy zgłasza błąd: „Wprowadzone zmiany nie zostały zapisane, ponieważ edytowane dane są nieaktualne”.</li><li>5. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłaszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby.</li></ol>

**Wyświetl listę surowców** – przypadek użycia dostępny jest dla poziomu dostępu Assessor. Po naciśnięciu przycisku „Lista surowców” w menu rozwijanym „Produkty” wszystkie surowce pobierane są z bazy i wyświetlane w formie listy. Użytkownik może wybrać surowiec z listy i wykonać akcję z nim związaną: edytować go lub usunąć.

Aktor	Użytkownik z nadaną rolą Assessor.
Scenariusz główny	Użytkownik wyświetla listę surowców.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Lista surowców nie zostaje wyświetlona, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li></ol>

**Dodaj surowiec** – przypadek użycia dostępny dla poziomu dostępu Assessor. Po naciśnięciu przycisku „Dodaj surowiec” użytkownik przechodzi do formularza tworzenia surowca. Po wprowadzeniu danych i naciśnięciu przycisku „Ok”, nastąpi przekierowanie do strony weryfikacji wprowadzonych danych. Po zatwierdzeniu poprawności danych, nowy surowiec zostanie zapisany w bazie.

Aktor	Użytkownik z nadaną rolą Assessor.
Scenariusz główny	Użytkownik uzupełnia poprawnie formularz, weryfikuje wprowadzone dane, a następnie po naciśnięciu przycisku „Utwórz” do bazy zostaje dodany nowy surowiec.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz zgłasza błąd „Pole nie może być puste”.</li><li>2. Użytkownik wprowadza nazwę surowca, która już istnieje w bazie – formularz tworzenia analizy zgłasza błąd: „Surowiec o podanej nazwie już istnieje”.</li><li>3. Surowiec nie zostaje utworzony, brak połączenia z bazą danych – do dzienników zdarzeń aplikacji wprowadzany jest zapis o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby w późniejszym czasie.</li></ol>

**Edytuj surowiec** – przypadek użycia dostępny z poziomu listy surowców dla Assessora, umożliwiający zmianę edytowalnych danych surowca.

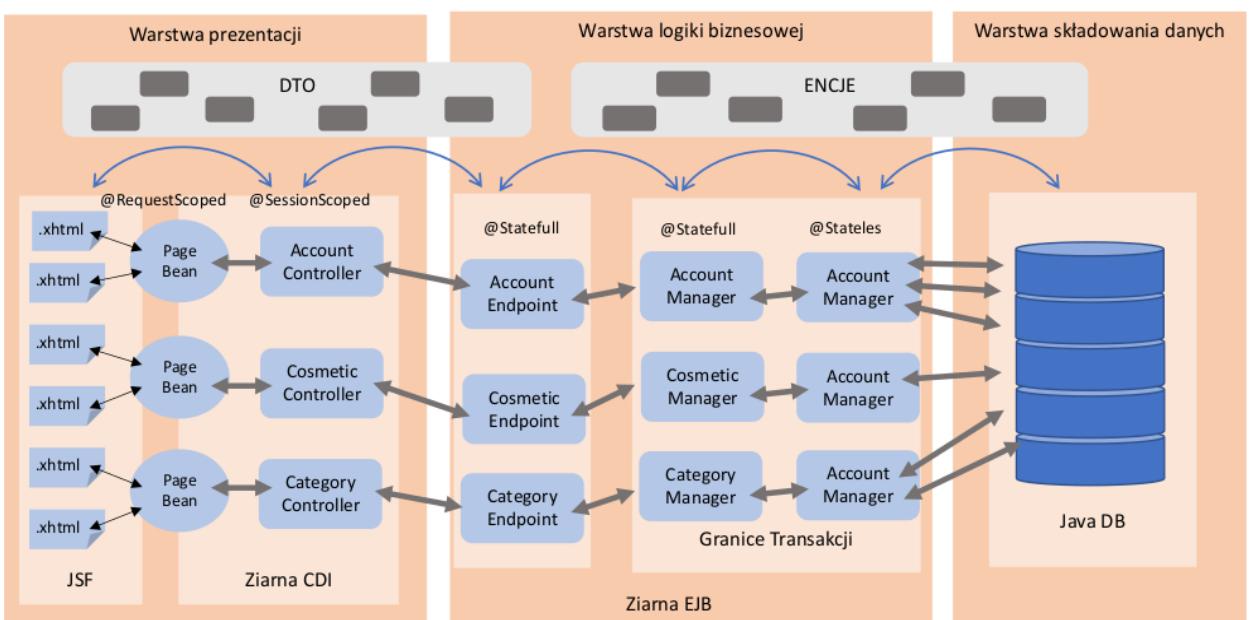
Aktor	Użytkownik z nadaną rolą Assessor.
Scenariusz główny	Użytkownik pobiera dane surowca, aktualizuje je wypełniając formularz poprawnie. Edytowane dane zapisywane są w bazie.
Scenariusze błędów	<ol style="list-style-type: none"><li>1. Nie wszystkie wymagane (oznaczone gwiazdką) pola zostały uzupełnione – formularz zgłasza błąd „Pole nie może być puste”.</li><li>2. Użytkownik wprowadza nazwę surowca, która już istnieje w bazie – formularz edycji analizy zgłasza błąd: „Surowiec o podanej nazwie już istnieje”.</li><li>3. Podczas edycji surowca inny użytkownik zmienił jego dane – formularz edycji surowca zgłasza błąd: „Wprowadzone zmiany nie zostały zapisane, ponieważ edytowane dane są nieaktualne”.</li><li>4. Brak połączenia z bazą danych – do dzienników zdarzeń aplikacji zgłoszana jest informacja o problemach z połączeniem z bazą. Na stronie wyświetlany jest komunikat z prośbą o kontakt z Administratorem systemu oraz ponowienie próby.</li></ol>

### 3 Realizacja projektu

Poniższe rozdziały prezentują zastosowany w aplikacji, powszechny dla aplikacji biznesowych model trójwarstwowy (widoczny na rysunku 13) obejmujący:

- Warstwę prezentacji, która udostępnia interfejs użytkownika, to w tej warstwie użytkownik wprowadza dane oraz wykonuje akcje aplikacji.
- Warstwę logiki biznesowej, która implementuje reguły przetwarzania danych zgodnie z modelem biznesowym, który aplikacja odwzorowuje.
- Warstwę składowania danych, implementującą funkcjonalność składowania danych z wykorzystaniem statycznych struktur relacyjnej bazy danych.

Podział na warstwy ma charakter hierarchiczny, co oznacza, że każda warstwa może być wywoływana bezpośrednio przez warstwę położoną wyżej w hierarchii oraz wywoływać warstwę znajdująca się poniżej w hierarchii.



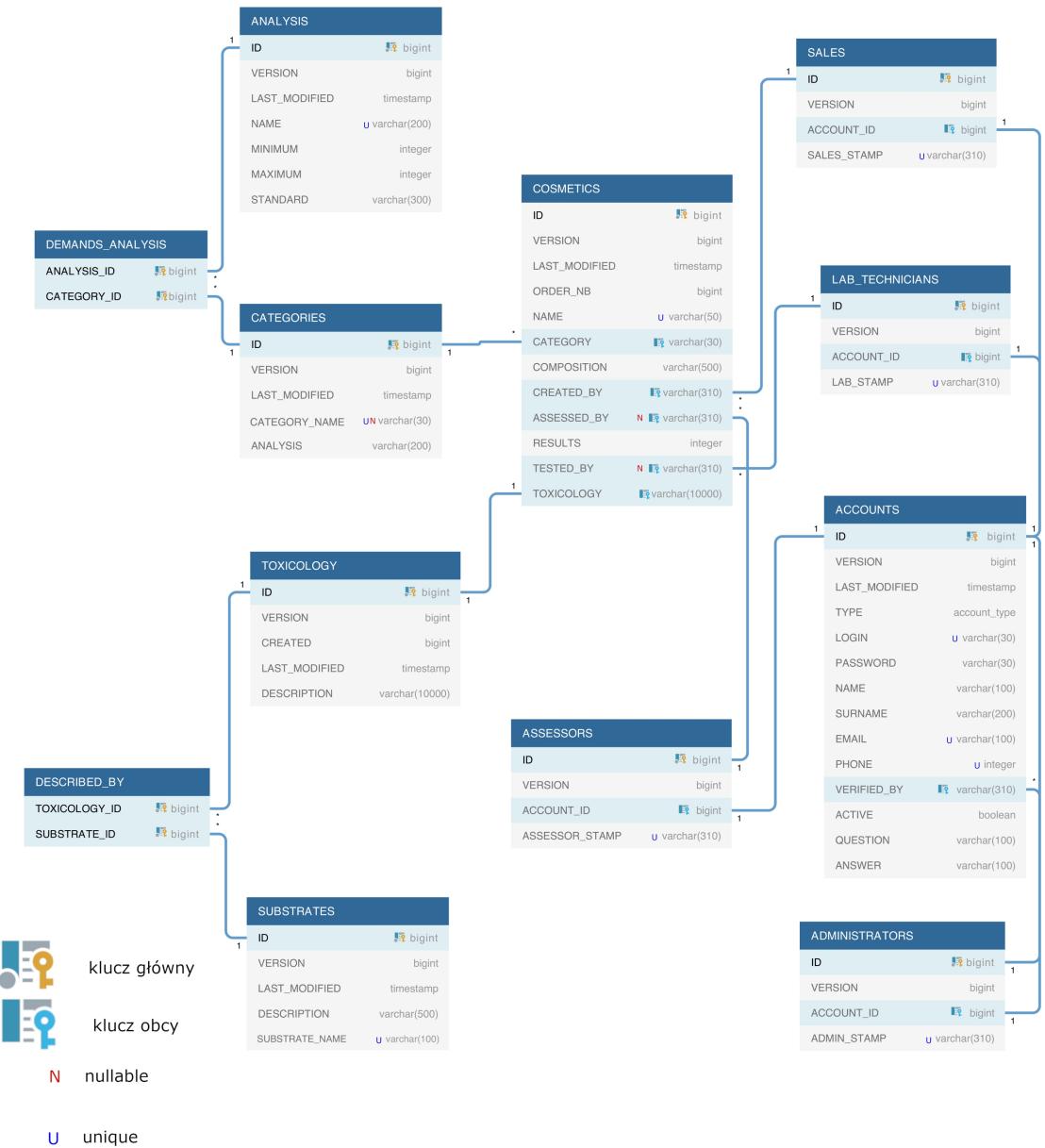
Rysunek 13: Diagram komponentów aplikacji dla wybranych przypadków użycia.

#### 3.1 Warstwa składowania danych

Do składowania danych w tworzonym systemie informatycznym wykorzystana została relacyjna baza danych, która osadzona została w systemie zarządzania bazami danych JavaDB. Baza zbudowana jest z tabel danych powiązanych ze sobą za pomocą unikalnych pól kluczy głównych i pól kluczy obcych poszczególnych rekordów w tabelach. Niektóre tabele powiązane są ze sobą za pomocą dodatkowej tabeli parującej klucze główne.

##### 3.1.1 Model relacyjnej bazy danych

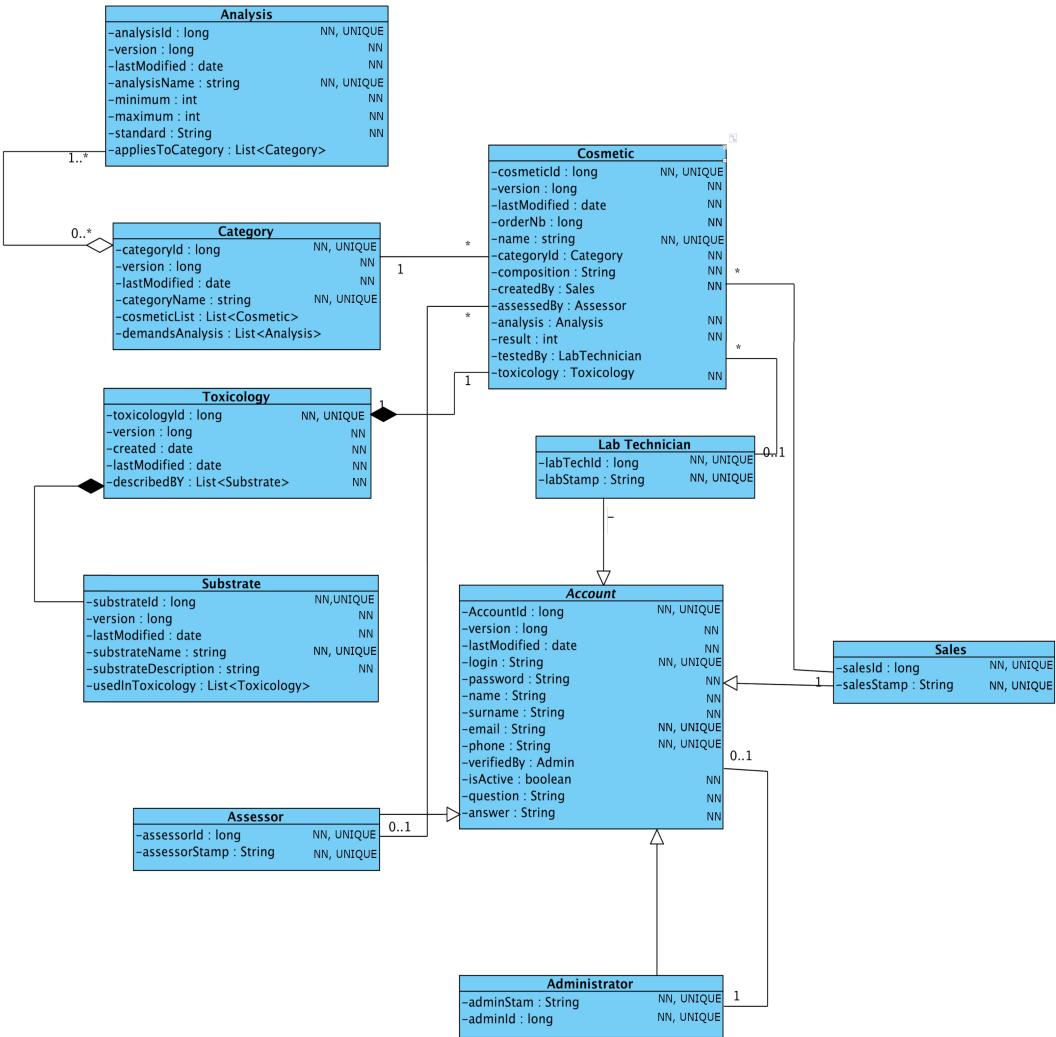
Na schemacie na rysunku 14 przedstawiono struktury bazodanowe wraz z relacjami i ograniczeniami bazodanowymi. Ponieważ znaczna część atrybutów ma ograniczenie @NotNull, na diagramie oznaczone zostały atrybuty, które mogą przyjmować wartość null.



Rysunek 14: Model relacyjnej bazy danych

### 3.1.2 Mapowanie obiektowo relacyjne

Obiektową architekturę systemu informatycznego można odwzorować na bazę o relacyjnym charakterze za pomocą mapowania obiektowo-relacyjnego ORM (ang. Object-Relational Mapping). Oznacza to, że struktury relacyjnej bazy danych odwzorowane zostają na odpowiednie obiekty Java. Mapowanie to, w serwerze aplikacyjnym Payara zapewnione jest przez standard JPA (Java Persistence API). Diagram klas encyjnych tworzonych systemu przedstawiony został na rysunku 15. Na diagramie przedstawiono atrybuty klas, dla każdego pola stworzono odpowiednie metody dostępowe. Ograniczenia bazodanowe generowane są w sposób automatyczny, natomiast mapowanie odbywa się za pomocą adnotacji, które zaprezentowane zostały na przykładzie fragmentu klasy encyjnej Cosmetic w listingu 9.



Rysunek 15: Diagram klas encyjnych

```

@Id
@Column(name = "COSMETIC_ID", updatable = false)
@GeneratedValue(strategy = GenerationType.AUTO)
private Long cosmeticId;

@Digits(integer = 12, fraction = 0)
@Column(name = "ORDER_NB")
private BigInteger orderNb;

@NotNull
@Size(min=3,max=32,message="{constraint.string.length.notinrange}")
@Column(name = "NAME", unique = true, nullable = false, updatable = true)
private String name;

@NotNull(message="{constraint.notNull}")
@JoinColumn(name = "CATEGORY_ID", nullable = true)
@ManyToOne(optional = true)
private Category categoryId;
@NotNull(message="{constraint.notNull}")
@Pattern(regexp="[a-zA-Z, ]",message="{constraint.string.incorrectchar}")
@Column(name = "COMPOSITION")
private String composition;

```

```

@JoinColumn(name = "CREATED_BY", nullable = true)
@ManyToOne(optional = true, cascade = {CascadeType.MERGE})
private BigInteger createdBy;

@JoinColumn(name = "ASSESSED_BY", nullable = true)
@ManyToOne(optional = true, cascade = {CascadeType.MERGE})
private BigInteger AssessedBy;

@JoinColumn(name = "TESTED_BY", nullable = true)
@ManyToOne(optional = true, cascade = {CascadeType.MERGE})
private BigInteger testedBy;

@OneToOne(cascade = {CascadeType.MERGE})
@JoinColumn(name = "TOXICOLOGY_ID")
private Toxicology toxicologyId;

```

*Listing 1: Fragment kodu klasy encyjnej Cosmetic*

W bazie danych zastosowano operacje kaskadowe na encjach, które są wykonywane automatycznie na obiektach JPA lub krotkach, które są związane z obiektem lub krotką, na których pierwotnie wykonywana jest dana operacja. Operacje kaskadowe można zdefiniować na poziomie klas encyjnych JPA, stosując atrybut *cascade* dla anotacji mapujących związki pomiędzy encjami JPA (listing 1).

## 3.2 Realizacja przykładowego CRUD'a

Aby dane przetwarzane w systemie mogły być poprawnie składowane w bazie danych, a także nie utraciły spójności, wiarygodności oraz mogły zostać zaprezentowane użytkownikowi, niezbędne jest ich spójne przekazywanie pomiędzy warstwami i komponentami. Przykładem przepływu danych w systemie jest realizacja CRUD (ang. Create, Read, Update, Delete) [5], czyli podstawowych funkcji umożliwiających zarządzanie danymi w pamięci trwałej:

- tworzenie - dodawanie nowych danych do bazy (create),
- odczytywanie/wyświetlanie istniejących informacji (read),
- modyfikowanie – edycja istniejących informacji (update),
- usuwanie istniejących informacji (delete).

Poniższy rozdział przedstawia opis realizacji CRUD'a dla klasy encyjnej Cosmetic, której obiekt reprezentuje zbiór danych rekordu z tabeli COSMETICS.

Widoki interfejsu użytkownika w aplikacji internetowej stanowią dokumenty XHTML. Funkcję modelu spełniają ziarna CDI (ang. Context and Dependency Injection) o różnych zasięgach: @ViewScoped, @RequestScoped oraz @SessionScoped. Ziarna aplikacji zawierają pola służące do przechowywania danych, jak również metody obsługi akcji wywoływanych przez przyciski strony. Obiekty sesyjne (Kontrolery) służą do ustalania informacji o danych, przetwarzanych dłużej niż jedno żądanie, np. wczytanie obiektu do edycji na jednym formularzu, edycja danych na kolejnym oraz zapisanie danych. Warstwa logiki biznesowej jest zrealizowana w postaci aplikacji internetowej zawierającej komponenty EJB (ang. Enterprise Java Beans) i osadzonej w kontenerze EJB, który odpowiedzialny jest za zarządzanie tymi komponentami.

Dane odczytywane z tabel w bazie danych stanowią zbiór encji JPA (ang. Java Persistence API), zarządzanych przez Zarządcę Encji (ang. Entity Manager). Zarządca Encji jest pozyskiwany przez wstrzyknięcie @PersistenceContext do bezstanowych komponentów EJB – fasad. W opisywanym przykładzie przypadków użycia CRUD jest to CosmeticFacade. Zarządca encji zarządza encjami oraz dostarcza metody, które umożliwiają wykonywanie na nich operacji. Fasady mogą zgłaszać wyjątki związane z odczytem danych z bazy a także mechanizmem blokad optymistycznych. Na tym etapie następuje również przechwycenie wyjątków

i przekształcenie ich na wyjątki aplikacyjne, żeby uniknąć przekazania ich obsługi do kontenera. Przekazanie to uniemożliwiłoby poprawną obsługę wyjątków komponentach EJB, ze względu na stosowaną przez kontener domyślną procedurę obsługi dla przechwyconych wyjątków nieaplikacyjnych.

Po odczytaniu danych z bazy następuje ich przekazanie wewnątrz kontenera EJB do stanowych komponentów EJB – menadżerów (CosmeticManager), oraz punktów dostępowych - endpointów (CosmeticEndpoint). Stanowość komponentów pozwala na zachowanie stanu encji i po przetworzeniu danych w warstwie prezentacji utrwalenie jej w bazie z wykorzystaniem bezstanowych fasad. Jest to szczególnie istotne w przypadku edycji danych, w której przechowywanie stanu pozwala zachować obiekt encji JPA wraz z numerem wersji, który wykorzystywany jest przez mechanizm blokad optymistycznych. W komponentach EJB stanowiących punkty końcowe warstwy logiki biznesowej realizowana jest logika biznesowa aplikacji. Komponenty EJB mogą być wstrzykiwane do ziaren CDI kontenera.

### 3.2.1 Wyświetlanie listy kosmetyków

Wyświetlenie listy kosmetyków to przypadek użycia odpowiedzialny za pobieranie listy kosmetyków z bazy danych oraz wyświetleniu ich w formie tabeli w widoku użytkownika (rysunek 8). Jest to przypadek użycia dostępny dla użytkownika z poziomami dostępu Handlowiec, Laborant i Assessor. Strona odpowiedzialna za wyświetlenie sformatowanych danych przedstawiona została w listingu 2.

```
<ui:define name="content">
    <b:container fluid="true">
        <h2 align="center" style="color: navy; font-size:18px">Lista produktów kosmetycznych do oceny</h2>
        <b:column>
            <b:form>
                <b:commandButton iconAwesome="refresh" action="#{listCosmeticsBean.init()}" />
                <b:commandButton value="Utwórz kosmetyk" action="#{createCosmeticBean.begin()}" />
                <b:dataTable var="row" value="#{listCosmeticsBean.cosmetics}" >
                    <h:column id="orderNb">
                        <f:facet name="header">Numer zamówienia</f:facet>
                        <h:outputText value="#{row.orderNb}" />
                    </h:column>
                    <h:column id="name">
                        <f:facet name="header">Nazwa</f:facet>
                        <h:outputText value="#{row.name}" />
                    </h:column>
                    <h:column id="category">
                        <f:facet name="header">Kategoria</f:facet>
                        <h:outputText value="#{row.category.getCategoryName()}" />
                    </h:column>
```

```

<h:column id="composition">
    <f:facet name="header">Skład</f:facet>
    <h:outputText value="#{row.composition}" />
</h:column>
<h:column id="createdBy">
    <f:facet name="header">Utworzony przez</f:facet>
    <h:outputText value="#{row.createdBy}" />
</h:column>
<h:column id="assessedBy">
    <f:facet name="header">Ocenione przez</f:facet>
    <h:outputText value="#{row.assessedBy}" />
</h:column>
<h:column id="actions">
    <f:facet name="header">Akcje</f:facet>
    <b:commandButton iconAwesome="trash"
                      action="#{listCosmeticsBean.downloadCosmeticForDeletion(row)}" />
    <b:commandButton value="Edytuj" action="#{listCosmeticsBean.edit(row)}" />
    <b:commandButton value="Szczegóły" action="#{listCosmeticsBean.showDetails(row)}" />
</h:column>
</b:dataTable>
</b:form>
</b:column>
</b:container>
</ui:define>
```

*Listing 2: Fragment kodu strony listCosmetics.xhtml*

Klasa *ListCosmeticsBean* o zasięgu żądania @ViewScoped wywołuje metodę inicjującą *init* (listing 3), która wywołuje metodę komponentu stanowego EJB *CosmeticEndpoint* (listing 4) poprzez metodę klasy *CosmeticController* (listing 5).

```
@PostConstruct
public void init() {
    cosmetics = cosmeticController.listAllCosmetics();
}
```

*Listing 3: Fragment kodu klasy ListCosmeticBean*

```
public List<CosmeticDTO> listAllCosmetics() throws AppBaseException{
    return CosmeticConverter.createListCosmeticDTOFromEntity(cosmeticManager.downloadAllCosmetics());
}
```

*Listing 4: Fragment kodu klasy CosmeticEndpoint*

```
public List<CosmeticDTO> listAllCosmetics() {
    try{
        return cosmeticEndpoint.listAllCosmetics();
    }catch(AppBaseException abe){
        Logger.getLogger(CosmeticController.class.getName()).log(Level.SEVERE,
            "Zgłoszenie w metodzie akcji createSubstrate wyjątku typu: ", abe.getClass());
        if (ContextUtils.isInternationalizationKeyExist(abe.getMessage())){
            ContextUtils.emitInternationalizedMessage(null, abe.getMessage());
        }
        return null;
    }}
```

*Listing 5: Fragment kodu klasy CosmeticController*

```
public List<T> findAll() {
    javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
    cq.select(cq.from(entityClass));
    return getEntityManager().createQuery(cq).getResultList();
}
```

*Listing 6: Metoda findAll z interfejsu AbstractFacade*

W metodzie `listAllCosmetics` klasy `CosmeticController` została wykorzystana klasa `CosmeticConverter`. Jest to klasa narzędziowa służąca do przekształcania obiektów typu DTO (ang. Data Transfer Object) w obiekty encji oraz obiektów encji w obiekty typu DTO. Pobieranie obiektów encji z bazy danych następuje przez wykonanie metody `findAll` pochodzącej z interfejsu `AbstractFacade` (*listing 6*).

### 3.2.2 Tworzenie kosmetyku

Dodanie nowego kosmetyku – ten przypadek użycia prowadzi do utworzenia nowego kosmetyku do oceny bezpieczeństwa przez dodanie nowej pozycji w tabeli COSMETICS w bazie danych. W tym celu Handlowiec wypełnia formularz przedstawiony na rysunku 9. Formularz generowany jest przez dokument `createNewCosmetic.xhtml`, którego fragment przedstawia *listing 7*. Następnie po naciśnięciu przycisku „Utwórz” następuje przekierowanie do strony weryfikacji wprowadzonych danych, na której naciśnięcie przycisku „Utwórz” wywołuje metodę `createCosmetic` z klasy `CreateNewCosmeticBean`, którą przedstawia *listing 8*.

```
<b:form id="editCosmetic">
    <h:outputLabel value="#{msg['cosmetic.orderNb']}:" for="orderNb"/>
    <b:inputText id="orderNb" value="#{createCosmeticBean.newCosmetic.orderNb}" required="true"/>
    <h:messages for="name" class="error" errorStyle="color:red"/>

    <h:outputLabel value="#{msg['cosmetic.name']}:" for="name"/>
    <b:inputText id="name" value="#{createCosmeticBean.newCosmetic.name}" required="true" />
    <h:messages for="name" class="error" errorStyle="color:red"/>

    <h:outputLabel value="#{msg['cosmetic.categoryChoice']}:" for="categoryName"/>
    <b:selectOneMenu id="categoryName" value="#{createCosmeticBean.categoryName}">
        <f:selectItems value="#{createCosmeticBean.listCategoryDTO}" var="categoryName"
            itemValue="#{category.categoryName}" itemLabel="#{category.categoryName}" />
    </b:selectOneMenu>

    <h:outputLabel value="#{msg['cosmetic.composition']}:" for="composition"/>
    <b:inputText id="composition" value="#{createCosmeticBean.newCosmetic.composition}"
        required="true"/>
    <h:messages for="composition" class="error" errorStyle="color:red"/>

    <b:commandButton value="#{msg['action.create']}"
        action="#{createCosmeticBean.confirmCosmetic()}" />
    <b:commandButton value="#{msg['action.cancel']}" action="#{createCosmeticBean.abort()}"
        immediate="true"/>
</b:form>
```

*Listing 7: Fragment strony `createNewCosmetic.xhtml` - formularz tworzenia kosmetyku.*

Klasa `CreateCosmeticBean` jest ziarem CDI o zasięgu sesji (@SessionScoped). W metodzie `begin` pobierana jest lista kategorii kosmetyków, która pozwala na wybór kategorii z listy rozwijanej na stronie WWW. W klasie tej następuje wywołanie metody `createCosmetic` na rzecz klasy `CosmeticController` (*listing 9*), będącej ziarem CDI o zasięgu sesji. Metoda ta jako parametry przyjmuje obiekt typu DTO (`CosmeticDTO`) oraz obiekt typu String (`categoryName`).

```

@SessionScoped
@Named
public class CreateCosmeticBean implements Serializable {
(...)

@Getter
@Setter
private List<CategoryDTO> listCategoryDTO;
(...)

public String begin() {
    newCosmetic = new CosmeticDTO();
    categoryName = new String();
    listCategoryDTO = categoryEndpoint.listAllCategories();
    return "createCosmetic";
}

(...)

public String confirmCosmetic() {
    if (null != newCosmetic && null != newCosmetic.getName()) {
        CategoryDTO categoryId = CategoryConverter.createCategoryDTOFromEntity
            (categoryFacade.findByName(categoryName));
        newCosmetic.setCategoryId(categoryId);
        categoryController.chooseCategory(newCosmetic);
        return "confirmCosmetic";
    } else {
        return "listCosmetics";
    }
}

public String createCosmetic() {
    cosmeticController.createCosmetic(newCosmetic, categoryName);
    return "listCosmetics";
}

```

*Listing 8: Fragment kodu klasy CreateCosmeticBean*

```

public String createCosmetic(CosmeticDTO newCosmetic, String categoryName) {
    try {
        createdCosmetic = newCosmetic;
        cosmeticEndpoint.createCosmetic(createdCosmetic, categoryName);
        createdCosmetic = null;
        return "listCosmetics";
    } catch (CosmeticException ce) {
        if (CosmeticException.KEY_COSMETIC_NAME_EXISTS.equals(ce.getMessage())) {
            ContextUtils.emitInternationalizedMessage("createNewCosmeticForm:name",
                CosmeticException.KEY_COSMETIC_NAME_EXISTS);
        } else {
            Logger.getLogger(CosmeticController.class.getName()).log(Level.SEVERE,
                "Zgłoszenie w metodzie akcji createSubstrate wyjątku: ", ce);
        }
        return null;
    } catch (AppBaseException abe) {
        (...)}

```

*Listing 9: Metoda createCosmetic w klasie CosmeticController*

W kontrolerze wywoływana jest metoda `createCosmetic` punktu dostępowego `CosmeticEndpoint` (listing 10), będącego komponentem stanowym EJB. Metoda ta jest odpowiedzialna za konwersję pomiędzy obiektami transferowymi DTO a obiektami encji JPA, poprzez wykorzystanie metody `createCosmeticEntityFromDTO` pochodzącej z klasy `CosmeticConverter`. W metodzie `createNewCosmetic` zastosowano mechanizm ponawiania transakcji. W deskryptorze `web.xml` ustalono limit na 3 próby

ponawiania odwołanej transakcji. System jest klasy OLTP (ang. On-Line Transakctional Processing), co oznacza, że przetwarzanie danych odbywa się w ramach transakcji realizowanych na bieżąco. W związku z tym transakcja jest zatwierdzana (ang. commit) tylko wtedy, gdy wszystkie operacje kończą się sukcesem. W przypadku jakiegokolwiek błędu (wyjątek, niespójność danych, limit czasu, błędy z poziomu bazy danych itp.) transakcja jest wycofywana (ang. rollback). Transakcje zostały dokładniej omówione w rozdziale 3.3.2 Mechanizm ochrony spójności danych. Metoda `createCosmetic` ma atrybut transakcyjny `NEVER`. Oznacza to, że transakcja rozpoczynana jest przez kontener EJB wraz z wywołaniem metody biznesowej menadżera EJB, którym w tym przypadku jest `CosmeticManager`.

```
@TransactionAttribute(TransactionAttributeType.NEVER)
@RolesAllowed({"SALES"})
public void createCosmetic(CosmeticDTO cosmetic, String name) throws AppBaseException {
    Cosmetic cosmetic = new Cosmetic();
    cosmetic.setOrderNb(newCosmetic.getOrderNb());
    cosmetic.setName(newCosmetic.getName());
    cosmetic.setCategoryId(CategoryConverter.createCategoryEntityFromDTO(newCosmetic.getCategoryId()));
    cosmetic.setComposition(newCosmetic.getComposition());
    boolean rollbackTX;
    int retryTXCounter = txRetryLimit;
    do {
        try {
            cosmeticManager.createCosmetic(cosmetic, categoryName);
            rollbackTX = cosmeticManager.isLastTransactionRollback();
        } catch (AppBaseException | EJBTransactionRolledbackException ex) {
            Logger.getGlobal().log(Level.SEVERE, "Próba " + retryTXCounter
                + " wykonania metody zakończona wyjątkiem klasy:" + ex.getClass().getName());
            rollbackTX = true;
        }
    } while (rollbackTX && --retryTXCounter > 0);
    if (rollbackTX && retryTXCounter == 0) {
        throw CosmeticException.createCosmeticExceptionWithTxRetryRollback();
    }
}
```

*Listing 10: Metoda `createCosmetic` klasy `CosmeticEndpoint`*

Następuje tu wywołanie metody `createCosmetic` w klasie `CosmeticManager` (listing 11). `CosmeticManager` ma wartość atrybutu adnotacji `@Transactional` ustawiony na `REQUIRES_NEW`. Oznacza to, że metoda `createCosmetic` wykonana jest w zakresie nowo utworzonej transakcji. Komponent stanowy `CosmeticManager` pośredniczy pomiędzy komponentem stanowym EJB typu Endpoint a komponentem bezstanowym EJB typu `Facade`.

```
@Stateful
public class CosmeticManager extends AbstractManager{
    ...
    @RolesAllowed({"SALES"})
    public String createCosmetic(Cosmetic newCosmetic, String categoryName) throws AppBaseException {
        Category category = categoryFacade.findByCategoryName(categoryName);
        newCosmetic.setCategoryId(category);
        categoryName = category.getCategoryName();
        cosmeticFacade.create(newCosmetic);
        return categoryName;
    }
}
```

*Listing 11: Fragment klasy `CosmeticManager`*

Fasady (ang. Facade) to bezstanowe komponenty EJB, które pośredniczą w przeprowadzaniu operacji CRUD na encjach JPA. Fasady mają nadany atrybut transakcyjny Mandatory. Oznacza to, że metoda biznesowa komponentu musi zostać wykonana w zakresie istniejącej transakcji komponentu EJB, z którego nastąpiło wywołanie. Inaczej zostaje zgłoszony wyjątek `javax.ejb.TransactionRequiredException`. W klasie `CosmeticFacade` następuje przeciążenie metody `create` widocznej na *listingu 12* poprzez referencję do klasy nadzędnej `AbstractFacade`, w której na obiekcie klasy `EntityManager` wywoływana jest metoda `persist` prowadząca do utworzenia nowej krotki w tabeli `COSMETICS`.

```
@Override  
public void create(Cosmetic entity) throws AppBaseException {  
    try {  
        super.create(entity);  
        em.flush();  
    } catch (PersistenceException | DatabaseException ex) {  
        final Throwable cause = ex.getCause();  
        final Throwable causeCause = ex.getCause().getCause();  
        if (cause instanceof DatabaseException  
            &&  
            causeCause.getMessage().contains(DB_UNIQUE_CONSTRAINT_FOR_COSMETIC_NAME)) {  
            throw CosmeticException.createWithDbCheckConstraintKey(entity, cause);  
        }  
    }  
}
```

*Listing 12: Metoda create w klasie CosmeticFacade*

W fasadach zgłaszane są wyjątki aplikacyjne. Przechwycenie wyjątków aplikacyjnych wynikających z błędów jakie DBMS (ang. DataBase Management System) zgłosił w trakcie realizacji kwerend SQL (ang. Structured Querry Language) wymaga wcześniejszego wymuszenia wykonania tych kwerend, jeszcze zanim bieżąca transakcja zostanie zakończona. Wymuszenie to zapewnia metoda `em.flush()`.

### 3.2.3 Edycja kosmetyku

Edycja kosmetyku jest to przypadek użycia, który umożliwia zmianę nazwy, kategorii oraz składu wprowadzonego kosmetyku. Przycisk edycji kosmetyku znajduje się na stronie *listCosmetics.xhtml* widocznej na rysunku 8. Przycisk ten wywołuje metodę `editCosmetic`, która odpowiedzialna jest za pobranie kosmetyku do edycji (`editedCosmetic`), zapamiętanie stanu kosmetyku (`cosmeticState`) oraz wywołanie strony `editCosmetic.xhtml`. `EditedCosmetic` jest obiektem pobranym z bazy i zapamiętywanym w polu stanowego komponentu EJB. Po naciśnięciu przycisku „Ok” na stronie `editCosmetic.xhtml` wywoływana jest metoda `saveCosmeticAfterEdition` z klasy `EditCosmeticBean`. Metoda ta służy do przekazania obiektu typu DTO do komponentu CDI `CosmeticController` o zasięgu sesji (*listing 13*), gdzie zaimplementowany jest segment obsługi wyjątków, co szerzej opisane jest w rozdziale 3.3.4 Obsługa błędów.

Tak jak w przypadku dodawania nowego kosmetyku komponent stanowy `CosmeticEndpoint` korzystając z klasy `CosmeticConverter` przeprowadza konwersję obiektów transferowych DTO na encję, co przedstawia *listing 14*. W metodzie `saveCosmeticAfterEdition` do zapamiętanego stanu kosmetyku `cosmeticState` przypisywane są podane przez użytkownika

nowe wartości. W przypadku gdy wersja obiektu `cosmeticState` jest niezgodna z wersją obiektu aktualnie znajdującej się w bazie zgłoszony zostaje wyjątek `javax.persistence.OptimisticLockException`. Następnie wywoływana jest metoda `editCosmetic` z klasy `CosmeticManager`.

```
@Named("cosmeticSession")
@SessionScoped
public class CosmeticController implements Serializable {
(...)
public String saveCosmeticAfterEdition(CosmeticDTO cosmeticDTO, String categoryName){
    try{
        categoryName = editCosmetic.getCategory().getCategoryName();
        cosmeticEndpoint.saveCosmeticAfterEdition(cosmeticDTO, categoryName);
        return "listCosmetics";
    } catch (CosmeticException ce) {
        if (CosmeticException.KEY_COSMETIC_NAME_EXISTS.equals(ce.getMessage())) {
            ContextUtils.emitInternationalizedMessage("createNewCosmeticForm:name",
                CosmeticException.KEY_COSMETIC_NAME_EXISTS);
        } else if(CosmeticException.KEY_OPTIMISTIC_LOCK.equals(ce.getMessage())){
            ContextUtils.emitInternationalizedMessage(null, CosmeticException.KEY_OPTIMISTIC_LOCK);
        } else {
            Logger.getLogger(CosmeticController.class.getName()).log(Level.SEVERE,
                "Zgłoszenie w metodzie akcji editSubstrate wyjątku: ", ce);
        }
        return null;
    } catch (AppBaseException abe) {
        Logger.getLogger(SubstrateController.class.getName()).log(Level.SEVERE,
            "Zgłoszenie w metodzie akcji editSubstrate wyjątku typu: ", abe.getClass());
        if (ContextUtils.isInternationalizationKeyExist(abe.getMessage())) {
            ContextUtils.emitInternationalizedMessage(null, abe.getMessage());
        }
        return null;
    }
}
(...)}
Listing 13: Fragment kodu klasy CosmeticController – edycja kosmetyku
```

```
@Stateful
public class CosmeticEndpoint {
(...)
public void saveCosmeticAfterEdition(CosmeticDTO cosmeticDTO, String categoryName) throws
AppBaseException {
    if (null == cosmeticState) {
        throw CosmeticException.createExceptionWrongState(cosmeticState);
    }
    cosmeticState.setName(cosmeticDTO.getName());
    cosmeticState.setCategoryId(CategoryConverter.createCategoryEntityFromDTOForCosmetic(cosmeticDT
O.getCategory()));
    cosmeticState.setComposition(cosmeticDTO.getComposition());
    boolean rollbackTX;
    int retryTXCounter = txRetryLimit;
    do {
        try {
            cosmeticManager.saveCosmeticAfterEdition(cosmeticState, categoryName);
            rollbackTX = cosmeticManager.isLastTransactionRollback();
        } catch (CosmeticException ce) {
            ...
        }
    }
}
```

*Listing 14: Metoda saveCosmeticAfterEdition z klasy CosmeticEndpoint*

*CosmeticManager* przekazuje obiekt do metody *editCosmetic* bezstanowego komponentu EJB *CosmeticFacade*. Jest to metoda przeciążona, wywoływana w klasie nadziednej *AbstractFacade* i jako parametr przyjmuje obiekt klasy encyjnej. W klasie *AbstractFacade* zostaje wywołana metoda *merge*, która powoduje edycję encji w tabeli COSMETICS w bazie danych.

```

@Override
public void edit(Cosmetic entity) throws AppBaseException {
    try {
        super.edit(entity);
        em.flush();
    } catch (OptimisticLockException oe) {
        throw CosmeticException.createCosmeticExceptionWithOptimisticLockKey(entity, oe);
    } catch (PersistenceException | DatabaseException ex) {

        final Throwable cause = ex.getCause();
        final Throwable causeCause = ex.getCause().getCause();
        if (cause instanceof DatabaseException
            &&
        causeCause.getMessage().contains(DB_UNIQUE_CONSTRAINT_FOR_COSMETIC_NAME)) {
            throw CosmeticException.createWithDbCheckConstraintKey(entity, cause);
        }
    }
}

```

*Listing 15: Metoda edit w klasie CosmeticFacade*

### 3.2.4 Usuwanie kosmetyku

Przypadek użycia usuń kosmetyk – powoduje usunięcie z bazy danych wybranego z listy kosmetyku. W celu usunięcia kosmetyku należy nacisnąć przycisk z ikoną kosza widniejący przy każdej pozycji w tabeli na stronie *listCosmetics.xhtml*, co jest widoczne na rysunku 8. Dostęp do tej funkcji ma wyłącznie użytkownik z przypisanym poziomem dostępu Handlowiec. Po naciśnięciu przycisku usuwania wywoływana jest metoda *deleteCosmetic*, z klasy *DeleteCosmeticBean*, wywołująca stronę *deleteCosmetic.xhtml*, która prosi o potwierdzenie usunięcia kosmetyku, widoczną na rysunku 10. Po naciśnięciu przycisku „Usuń kosmetyk” następuje wywołanie metody *confirmDeleteCosmetic* z klasy *DeleteCosmeticBean*, która przekazuje obiekt typu DTO do komponentu CDI klasy *CosmeticController* o zasięgu sesji (listing 16).

```

@SessionScoped
public class CosmeticController implements Serializable {
    public String confirmDeleteCosmetic(CosmeticDTO cosmeticDTO) throws AppBaseException {
        try {
            cosmeticEndpoint.confirmDeleteCosmetic(cosmeticDTO);
            return "cosmeticList";
        } catch (CosmeticException ce) {
            if (CosmeticException.KEY_COSMETIC_ALREADY_CHANGED.equals(ce.getMessage())) {
                ContextUtils.emitInternationalizedMessage(null,
                    CosmeticException.KEY_COSMETIC_ALREADY_CHANGED);
            } else if (CosmeticException.KEY_COSMETIC_NOT_FOUND.equals(ce.getMessage())) {
                ContextUtils.emitInternationalizedMessage(null,
                    CosmeticException.KEY_COSMETIC_NOT_FOUND);
            } else if (CosmeticException.KEY_COSMETIC_OPTIMISTIC_LOCK.equals(ce.getMessage())) {
                ContextUtils.emitInternationalizedMessage(null,
                    CosmeticException.KEY_COSMETIC_OPTIMISTIC_LOCK);
            }
        }
    }
}

```

```

    } else {
        Logger.getLogger(CosmeticController.class.getName()).log(Level.SEVERE,
            "Zgłoszenie w metodzie akcji deleteDiet wyjątku: ", ce);
    } return null;
} catch (AppBaseException abe) {
    Logger.getLogger(CosmeticController.class.getName()).log(Level.SEVERE,
        "Zgłoszenie w metodzie akcji deleteDiet wyjątku typu: ", abe.getClass());
    if (ContextUtils.isInternationalizationKeyExist(abe.getMessage())) {
        ContextUtils.emitInternationalizedMessage(null, abe.getMessage());
    } return null;
}
}

```

*Listing 16: Fragment klasy CosmeticController - usuwanie kosmetyku.*

Następnie obiekt *CosmeticDTO* przekazywany jest do warstwy logiki biznesowej, do komponentu EJB *CosmeticEndpoint* (*listing 17*).

```

@Stateful
public class CosmeticEndpoint {
    @TransactionAttribute(TransactionAttributeType.NEVER)
    public void confirmDeleteCosmetic(CosmeticDTO cosmeticDTO) throws AppBaseException {
        if (null == cosmeticState) {
            throw CosmeticException.createExceptionWrongState(cosmeticState);
        }
        boolean rollbackTX;
        int retryTXCounter = txRetryLimit;
        do {
            try {
                cosmeticManager.confirmDeleteCosmetic(cosmeticDTO.getId());
                rollbackTX = cosmeticManager.isLastTransactionRollback();
            } catch (AppBaseException | EJBTransactionRolledbackException ex) {
                Logger.getGlobal().log(Level.SEVERE, "Próba " + retryTXCounter
                    + " wykonania metody biznesowej zakończona wyjątkiem klasy:"
                    + ex.getClass().getName());
                rollbackTX = true;
            }
        } while (rollbackTX && --retryTXCounter > 0);
        if (rollbackTX && retryTXCounter == 0) {
            throw CosmeticException.createCosmeticExceptionWithTxRetryRollback();
        }
    }
}

```

*Listing 17: Fragment klasy CosmeticEndpoint – usuwanie kosmetyku*

W komponencie *CosmeticEndpoint* obiekt *CosmeticDTO* przekształcany jest za pomocą *CosmeticConvertera* do obiektu encji, który przekazywany jest do stanowego komponentu EJB *CometicManager* (*listing 18*), który wywołuje metodę *remove* klasy *CosmeticFacade* (*listing 19*). Metoda ta jest metodą przeciążoną, wywoływaną z klasy nadzędnej *AbstractFacade* i jako parametr przyjmuje obiekt klasy encyjnej. W klasie *AbstractFacade* na obiekcie klasy *EntityManager* wywoływana jest metoda *remove*, która powoduje usunięcie odpowiedniego rekordu z tabeli COSMETICS z bazy danych.

```

@Stateful
public class CosmeticManager extends AbstractManager{
    public void confirmDeleteCosmetic(Long id) throws AppBaseException{
        deletedCosmetic = cosmeticFacade.find(id);
        if (null == deletedCosmetic) {
            throw CosmeticException.createExceptionWrongState(deletedCosmetic);
        } else
            cosmeticFacade.remove(deletedCosmetic);
    }
}

```

*Listing 18: Fragment klasy CosmeticManager – usuwanie kosmetyku*

```

@Override
public void remove(Cosmetic entity) throws AppBaseException {
    try {
        super.remove(entity);
        em.flush();
    } catch (OptimisticLockException oe) {
        throw CosmeticException.createCosmeticExceptionWithOptimisticLockKey(entity, oe);
    }
}

```

*Listing 19: Fragment kodu klasy CosmeticFacade z metodą remove*

### 3.3 Warstwa logiki biznesowej

W warstwie logiki biznesowej dane przetwarzane są zgodnie z modelem biznesowym. Implementacja logiki biznesowej opiera się na ziarnach Enterprise JavaBeans.

#### 3.3.1 Sesyjne komponenty EJB

W pracy zastosowane zostały dwie grupy sesyjnych komponentów EJB. Pierwsza z nich, zawiera komponenty stanowe oznaczone adnotacją @Statefull, do których należą punkty dostępowe (ang. endpoint) oraz menadżery (ang. manager) umożliwiające komunikację warstwy widoku z warstwą logiki biznesowej. Komponenty te odpowiedzialne za transfer danych z obiektów DTO do obiektów encji JPA, dzięki czemu zwiększa się poziom bezpieczeństwa danych przechowywanych w bazie. Przechowują one również stan obiektów i ponawiają odwołane transakcje aplikacyjne.

Drugą stosowaną grupą komponentów EJB są komponenty bezstanowe z adnotacją @Stateless, reprezentowane przez fasady (ang. facade), zawierające metody realizujące podstawowe operacje na rekordach przechowywanych w strukturach relacyjnej bazy danych. Klasa *AbstractFacade* (listing 20) jest klasą nadzczną względem wszystkich fasad i udostępnia podstawowe metody do komunikacji z bazą danych. Wszystkie klasy, które dziedziczą z klasy *AbstractFacade* nadpisują jej metody oraz uzupełniają je o odpowiednie dla danego przypadku użycia wyjątki, co widoczne jest na listingu 12 przedstawiającym fragment klasy *CosmeticFacade*.

```

public abstract class AbstractFacade<T> {
    (...)

    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) throws AppBaseException {
        getEntityManager().persist(entity);
    }

    public void create(T entity) throws AppBaseException {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) throws AppBaseException {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) throws AppBaseException{
        getEntityManager().remove(getEntityManager().merge(entity));
    }

    public void refresh(T entity) {
        getEntityManager().refresh(entity);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    (...)}

```

*Listing 20: Fragment kodu klasy AbstractFacade*

### 3.3.2 Mechanizm ochrony spójności danych

Stworzony system jest systemem wielodostępnym, co oznacza, że transakcje mogą być przetwarzane równolegle, co z kolei może prowadzić do konfliktów w przypadku zapisu i odczytu danych. W tworzonym systemie zastosowano transakcje aplikacyjne i transakcje bazodanowe w celu kontrolowania współbieżnego dostępu do danych. Transakcja musi zostać wykonana bezbłędnie w całości, lub efekty jej zmian nie zostaną zatwierdzone (utrwalone). W wyniku jej wykonania nie może zostać naruszona spójność danych. Transakcja bazodanowa musi posiadać poziom izolacji, który wymusza separację danych pomiędzy współbieżnymi transakcjami. Musi również zostać zapewnione utrzymanie aktualnego stanu danych systemu np. w przypadku awarii. Poziom izolacji dla transakcji bazodanowych ustawiony został jako read-committed (listing 30). W aplikacji wykorzystano strategię CMT (ang. Container-Managed Transactions), co oznacza, że transakcje aplikacyjne zarządzane są przez kontener a adnotacje określające przebieg transakcji znajdują się nad deklaracjami klas i metod komponentów EJB. Atrybuty transakcyjne

umożliwiają określenie zasad przetwarzania zasobów biorących udział w transakcji aplikacyjnej, która docelowo kończy się wykonaniem odpowiedniej transakcji bazodanowej.

W celu rejestracji granic transakcji stanowy komponent EJB implementuje interfejs `javax.ejb.SessionSynchronisation`, a co za tym idzie udostępnia kontenerowi metody zwrotne, dzięki którym możliwy jest zapis oraz identyfikacja realizowanych transakcji aplikacyjnych. Metody zwrotne zaimplementowane zostały w klasie `AbstractManager` (listing 21). Są one wspólne dla wszystkich punktów dostępowych rozszerzających tą klasę. Przykładem takiej klasy jest `CosmeticEndpoint` przedstawiona na listingu 4, która dla wszystkich metod biznesowych ma przypisany atrybut transakcyjny `NEVER`. Wszystkie metody biznesowe klasy `CosmeticManager` (listing 11) posiadają atrybut transakcyjny ustawiony na `REQUIRES_NEW`, co oznacza, że transakcja aplikacyjna rozpoczętna jest wraz z rozpoczęciem metody komponentu EJB Managera. Fasady posiadają atrybut transakcyjny ustawiony na `MANDATORY`.

```
abstract public class AbstractManager {  
    @Resource  
    SessionContext sctx;  
    protected static final Logger LOGGER = Logger.getGlobal();  
    private String transactionId;  
    private boolean lastTransactionRollback;  
    public final int NB_ATTEMPTS_FOR_METHOD_INVOCATION = 3;  
    public boolean isLastTransactionRollback() {  
        return lastTransactionRollback;  
    }  
    public void afterBegin() {  
        transactionId = Long.toString(System.currentTimeMillis())  
            + ThreadLocalRandom.current().nextLong(Long.MAX_VALUE);  
        LOGGER.log(Level.INFO, "Transakcja TXid={0} rozpoczęta w {1},tożsamość: {2}",  
            new Object[]{transactionId, this.getClass().getName(),  
                sctx.getCallerPrincipal().getName()});  
    }  
    public void beforeCompletion() {  
        LOGGER.log(Level.INFO, "Transakcja TXid={0} przed zatwierdzeniem w {1},tożsamość {2}",  
            new Object[]{transactionId, this.getClass().getName(),  
                sctx.getCallerPrincipal().getName()});  
    }  
    public void afterCompletion(boolean committed) {  
        lastTransactionRollback = !committed;  
        LOGGER.log(Level.INFO, "Transakcja TXid={0} zakończona w {1} poprzez {3},tożsamość {2}",  
            new Object[]{transactionId, this.getClass().getName(),  
                sctx.getCallerPrincipal().getName(),  
                committed ? "ZATWIERDZENIE" : "ODWOŁANIE"});  
    }  
}
```

Listing 21: Klasa `AbstractManager`

Dzięki zastosowaniu mechanizmu blokad optymistycznych możliwe jest sprawdzenie czy dane zapisane w bazie nie uległy zmianie od czasu ich odczytu z bazy. Jest to kontrolowane przez zarządcę encji, który kontroluje pole zawierające numer wersji pochodzące z klasy `AbstractEntity` oznaczone adnotacją `@Version`. Obiekt pobraany z bazy zostaje zapamiętany w polu stanowym komponentu EJB. Jeśli wersja zapamiętanego obiektu nie jest zgodna z wersją obiektu znajdującego się w bazie zostaje zgłoszony wyjątek

`javax.persistence.OptimisticLockException`, którego obsługa zapewniona jest w metodach `edit` oraz `delete` w klasie fasady (listing 15 oraz 19). Na rysunku 9 przedstawiono komunikat, który wyświetlany jest użytkownikowi, w przypadku równoległej edycji tego samego kosmetyku.

### 3.3.3 Rejestrowanie zdarzeń w dziennikach, kontrola odpowiedzialności

Do realizacji rejestracji zdarzeń wykorzystano mechanizm interceptora z pakietu `javax.interceptor.Interceptors`. Mechanizm ten zawiera zestaw adnotacji i interfejsów, służących do definiowania klas oraz metod interceptora, a także do wiązania klas interceptora z odpowiednimi klasami systemu.

W tym celu zdefiniowano klasę `LoggingInterceptor`, która zawiera definicję metody oznaczonej adnotacją `@AroundInvoke`. Dzięki temu metoda ta jest wywoływana zawsze z metodami biznesowymi klasy, której definicja została poprzedzona adnotacją `@Interceptors` wskazującą definicję klasy interceptora. Powoduje to, że każde wywołanie metody biznesowej komponentu EJB posiadającego przypisany obiekt przechwytyujący (ang. interceptor) jest otaczane wywołaniem metody interceptora. W tym przypadku metoda odpowiedzialna jest za rejestrowanie poszczególnych wywołań metod biznesowych w dzienniku zdarzeń systemu. W listingu 22 przedstawiono definicję klasy `LoggingInterceptor`, natomiast listing 4 zawiera przykład adnotacji wiążącej obiekt przechwytyjący z klasą systemu.

```
public class LoggingInterceptor {  
    @Resource  
    private SessionContext sessionContext;  
    @AroundInvoke  
    public Object additionalInvokeForMethod(InvocationContext invocation) throws Exception {  
        StringBuilder sb = new StringBuilder("Wywołanie metody biznesowej ")  
            .append(invocation.getTarget().getClass().getName()).append('.').  
            .append(invocation.getMethod().getName());  
        sb.append(" z tożsamością: ").append(sessionContext.getCallerPrincipal().getName());  
        try {  
            Object[] parameters = invocation.getParameters();  
            if (null != parameters) {  
                for (Object param : parameters) {  
                    if (param != null) {  
                        sb.append(" z parametrem ").append(param.getClass().getName()).append('=').append(param);  
                    } else {  
                        sb.append(" bez wartości (null)");  
                    }  
                }  
            }  
            Object result = invocation.proceed();  
            if (result != null) {  
                sb.append(" zwrócono ").append(result.getClass().getName()).append('=').append(result);  
            } else {  
                sb.append(" zwrócono wartość null");  
            }  
            return result;  
        } catch (Exception ex) {  
            sb.append(" wystąpił wyjątek: ").append(ex);  
            throw ex;  
        } finally {  
            Logger.getLogger().log(Level.INFO, sb.toString());  
        }  
    }  
}
```

Listing 22: Definicja klasy `LoggingInterceptor`

Dodatkowo zaprezentowana definicja interceptora rejestrującego pozwala uzyskać zapisanie w dzienniku zdarzeń loginu użytkownika, który wywołał metodę wraz ze statusem transakcji. Jest to jeden z elementów kontroli odpowiedzialności działań użytkownika w systemie. Dzięki temu, że login użytkownika jest wartością unikalną, pozwala on na zidentyfikowanie użytkownika. *LoggingInterceptor* pozyskuje login wykorzystując klasy i metody pakietu *javax.ejb.SessionContext*, który jest wstrzykiwany jako zasób. Istnieje możliwość zapisania działań użytkownika, którego login uzyskiwany jest jako *sessionContext.getCallerPrincipal().getName()*.

Drugi element mechanizmu kontroli odpowiedzialności stanowi zapisywanie w bazie danych informacji o dacie utworzenia i/lub modyfikacji każdego z rekordów danych w systemie. Ich utrwalanie odbywa się przy pomocy metod klasy abstrakcyjnej (oznaczonych adnotacjami *@PrePersist* i *@Preupdate*), z której dziedziczą pozostałe klasy modelu obiektowego encji JPA.

### 3.3.3 Kontrola dostępu

Do zapewnienia autoryzacji w dostępne do metod biznesowych komponentów EJB wykorzystano mechanizmy dostępne z pakietu *javax.annotation.security*. Dzięki zastosowaniu adnotacji pochodzących z tego pakietu w fasadach oraz punktach dostępowych, możliwe jest przypisanie dostępu do metod biznesowych konkretnym poziomom typom kont. Przykład użycia adnotacji *@RolesAllowed* zaprezentowany został na listingu 10. Kontrola dostępu zdefiniowana została dla poszczególnych ról, na które mapowane są dane użytkowników dostarczane przez mechanizm uwierzytelniania. Szczegółowy opis mechanizmu uwierzytelniania zawarty został w rozdziale 3.4.4.

### 3.3.4 Obsługa błędów

W aplikacji zaimplementowano wielopoziomową obsługę zgłoszanych wyjątków. Wyjątki systemowe, które mogą zostać zgłoszone przez kontener EJB powodują odwołanie bieżącej transakcji, a ich obsługa sprowadza się do wyświetlenia strony błędu typu *java.lang.RuntimeException* zadeklarowanej w deskryptorze wdrożenia web.xml (listing 23).

```
<error-page>
    <error-code>403</error-code>
    <location>/faces/error/error403.xhtml</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/faces/error/error404.xhtml</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/faces/error/error.xhtml</location>
</error-page>
<error-page>
    <exception-type>java.lang.RuntimeException</exception-type>
    <location>/faces/error/error.xhtml</location>
</error-page>
```

Listing 23: Fragment pliku web.xml - deklaracja stron błędów *java.lang.RuntimeException*

W celu uproszczenia propagacji wyjątków aplikacyjnych stworzono nadklasę *AppBaseException*, z której dziedziczą wszystkie zaimplementowane klasy obsługi wyjątków m.in. *CosmeticException* widoczny na listingu 24. Klasa nadzędna opatrzona została adnotacją `@ApplicationException(rollback=true)`, dzięki czemu odwołanie transakcji aplikacyjnej i propagacja wyjątku nie prowadzi do zastosowania standardowej procedury obsługi wyjątków jaką stosuje kontener EJB, czyli komponent EJB nie zostanie zniszczony. Metody w kolejnych warstwach aplikacji m.in. zapewniają propagację wyjątku, dzięki zgłoszeniu go w sygnaturach ich metod (`throws`). Zgłoszony wyjątek jest przekazywany do warstwy widoku, gdzie następuje wyświetlenie odpowiadającego mu komunikatu, który informuje użytkownika o rodzaju zgłoszonego wyjątku oraz wymaganej od niego reakcji. Szczegółowy opis obsługi wyjątku w warstwie widoku znajduje się w rozdziale 3.4.6.

```
public class CosmeticException extends AppBaseException {

    static final public String KEY_COSMETIC_NAME_EXISTS =
"error.cosmetics.db.constraint.uniq.cosmeticName";
    static final public String KEY_COSMETIC_OPTIMISTIC_LOCK =
"error.cosmetics.optimisticlock";
    static final public String KEY_COSMETIC_NOT_READ_FOR_EDITION =
"error.cosmetic.cosmeticNotDownloadedForEdition";
(...)

    static public CosmeticException createCosmeticExceptionWithOptimisticLockKey(Cosmetic cosmetic,
OptimisticLockException oe) {
        CosmeticException ce = new CosmeticException(KEY_COSMETIC_OPTIMISTIC_LOCK, oe);
        ce.cosmetic = cosmetic;
        return ce;
    }

    public static CosmeticException createExceptionWrongState(Cosmetic cosmetic) {
        CosmeticException ae = new CosmeticException(KEY_COSMETIC_NOT_READ_FOR_EDITION);
        ae.cosmetic = cosmetic;
        return ae;
    }

    static public CosmeticException createCosmeticExceptionWithTxRetryRollback() {
        CosmeticException ce = new CosmeticException(KEY_TX_RETRY_ROLLBACK);
        return ce;
    }

    static public CosmeticException createWithDbCheckConstraintKey(Cosmetic cosmetic, Throwable
cause) {
        CosmeticException ce = new CosmeticException(KEY_COSMETIC_NAME_EXISTS, cause);
        ce.cosmetic = cosmetic;
        return ce;
    }
}
```

*Listing 24: Fragment kodu klasy CosmeticException*

## 3.4 Warstwa widoku

Do wykonania interfejsu użytkownika wykorzystano dynamicznie generowane strony WWW współczesnej przeglądarki internetowej. W jego przygotowaniu wykorzystano technologię Java Server Faces wraz z biblioteką *BootsFaces*.

### 3.4.1 Wzorzec projektowy DTO

Obiekt Transferu Danych (ang. *Data Transfer Object*) jest obiektem służącym do przenoszenia w obie strony danych pomiędzy warstwą logiki biznesowej oraz warstwą widoku. Dzięki zastosowaniu Obiektów Transferu Danych (klasy DTO) możliwa jest większa kontrola danych, które przekazywane są bezpośrednio do użytkownika systemu informatycznego. Większość pól dostępnych w klasach DTO jest zbieżna z polami klas encyjnych jednak, jak widać na listingu 25 zawierającym fragment klasy CosmeticDTO, do warstwy widoku nie trafiają wszystkie dane np. data modyfikacji czy numer wersji.

```
public class CosmeticDTO {  
  
    private Long id;  
    @NotNull(message="{constraint.notNull}")  
    @Digits(integer = 12, fraction = 0)  
    private BigInteger orderNb;  
    @NotNull(message="{constraint.notNull}")  
    @Size(min=3,max=32,message="{constraint.string.length.notInRange}")  
    private String name;  
    @NotNull(message="{constraint.notNull}")  
    private CategoryDTO categoryId;  
    @NotNull(message="{constraint.notNull}")  
    @Pattern(regexp="[a-zA-Z, ]",message="{constraint.string.invalidChar}")  
    private String composition;  
    private BigInteger createdBy;  
    private BigInteger assessedBy;  
    private BigInteger testedBy;  
    private Toxicology toxicologyId;  
  
    public CosmeticDTO(Long id, BigInteger orderNb, String name, CategoryDTO categoryId, String  
composition, BigInteger createdBy, BigInteger assessedBy, BigInteger testedBy, Toxicology toxicologyId) {  
        this.id = id;  
        this.orderNb = orderNb;  
        this.name = name;  
        this.categoryId = categoryId;  
        this.composition = composition;  
        this.createdBy = createdBy;  
        this.assessedBy = assessedBy;  
        this.testedBy = testedBy;  
        this.toxicologyId = toxicologyId;  
    }  
    (...)  
}
```

*Listing 25: Fragment kodu klasy CosmeticDTO*

### 3.4.2 Ujednolicony interfejs użytkownika

Wszystkie strony aplikacji posiadają ujednolicony interfejs graficzny, który nie jest zależny od poziomu dostępu użytkownika[6]. Do ich wykonania wykorzystano technologie JSF, HTML i CSS (ang. Cascading Style Sheets)[9] oraz bibliotekę BootsFaces. Interfejs użytkownika składa się z paska menu (menu), zawartości głównej (content) oraz stopki (footer), co zostało pokazane na rysunku 16.



Safety Assessment stworzone przez Aleksandę Kemonę

Rysunek 16: Wygląd strony z formularzem logowania

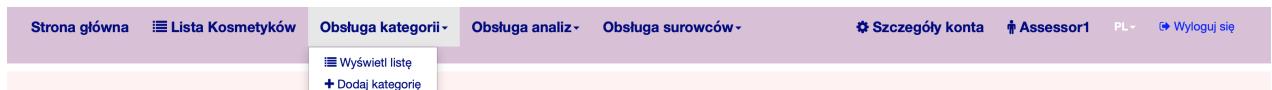
Zawartość formularzy oraz tabel pokazana została na rysunkach znajdujących się przy opisach przypadków użycia rozdziału 2.4.3 Opis przypadków użycia. Na poniższych rysunkach przedstawione przedstawiony został wygląd menu z uwzględnieniem dostępu do poszczególnych funkcji programu dla poziomów dostępu: Administrator (rysunek 17), Handlowiec (rysunek 18), Assessor (rysunek 19), Laborant (rysunek 20).



Rysunek 17: Wygląd paska menu dla poziomu dostępu Administrator



Rysunek 18: Wygląd paska menu dla poziomu dostępu Handlowiec



Rysunek 19: Wygląd paska menu dla poziomu dostępu Assessor



Rysunek 20: Wygląd paska menu dla poziomu dostępu Laborant

Każda strona wyświetlana w przeglądarce ma ujednoliconą strukturę i układ strony, zgodne z ustalonym dla interfejsu użytkownika szablonem strony.

### 3.4.3 Internacjonalizacja

Interfejs użytkownika systemu obsługuje dwie wersje językowe: polską i angielską, przy czym domyślną wersją jest wersja polska. Wykorzystano tu mechanizm internacjonalizacji, który pozwala zdefiniować w kodzie systemu klucze, które podczas prezentacji w interfejsie użytkownika odczytywane są jako napisy pochodzące z przygotowanych zasobów i ich wyświetlanie jest zgodne z ustawieniami językowymi wykorzystywanej przeglądarki. Klucze wraz z przypisanymi do nich wartościami zapisane są w plikach zasobów, których fragment przedstawiono na rysunku 21. Lokalizacja tych zasobów wskazana jest w pliku *faces-config.xml* (*listing 26*).

```

<application>
    <locale-config>
        <default-locale>pl</default-locale>
        <supported-locale>pl</supported-locale>
        <supported-locale>en</supported-locale>
    </locale-config>
    <resource-bundle>
        <base-name>i18n.messages</base-name>
        <var>msg</var>
    </resource-bundle>
    <message-bundle>
        i18n.jsf_messages
    </message-bundle>
    <resource-bundle>
        <base-name>/Bundle</base-name>
        <var>bundle</var>
    </resource-bundle>
</application>

```

*Listing 26: Fragment pliku faces-config.xml zawierający definicję zasobów potrzebnych do internacjonalizacji.*

Key	default language	pl - polski	en - angielski
app.name	[DEFAULT]Safety Assesment stwo...	Safety Assesment stworzone prze...	Safety Assesment created by Al...
action.cancel	[DEFAULT]Anuluj	Anuluj	Cancel
action.create	[DEFAULT]Utwórz	Utwórz	Create
action.delete	[DEFAULT]Usuń	Usuń	Delete
action.activate	[DEFAULT]Aktywuj	Aktywuj	Activate
action.deactivate	[DEFAULT]Deaktywuj	Deaktywuj	Deactivate
action.login	[DEFAULT]Zaloguj	Zaloguj	Login
main.page	[DEFAULT]Strona główna	Strona główna	Main page
menu.account	[DEFAULT]Konto	Konto	Account
menu.account.details	[DEFAULT]Szczegóły konta	Szczegóły konta	Account details
menu.products	[DEFAULT]Produkty	Produkty	Products
menu.products.list	[DEFAULT]Lista kosmetyków do oceny	Lista kosmetyków do oceny	List of assessed products
menu.products.categories	[DEFAULT]Kategorie produktów	Kategorie produktów	Product's categories
menu.products.analysis	[DEFAULT]Dostępne analizy	Dostępne analizy	Available analysis
menu.products.substrates	[DEFAULT]Dostępne surowce	Dostępne surowce	Available substrates

*Rysunek 21: Fragment tabeli zawierającej klucze do internacjonalizacji*

Mechanizm internacjonalizacji wykorzystany został nie tylko bezpośrednio w stronach \*.xhtml, ale również w prezentacji komunikatów pochodzących z warstwy logiki biznesowej i warstwy widoku. Komunikaty pochodzące z warstwy logiki biznesowej mogą być przedstawione w wersji językowej odpowiedniej do preferencji w przeglądarce dzięki metodom w klasie ContextUtils (listing 27).

```

public static boolean isInternationalizationKeyExist(String key) {
    return ContextUtils.getDefaultBundle().getString(key) != null
    && !"".equals(ContextUtils.getDefaultBundle().getString(key));
}

public static void emitInternationalizedMessage(String id, String key) {
    FacesMessage msg = new FacesMessage(ContextUtils.getDefaultBundle().getString(key));
    msg.setSeverity(FacesMessage.SEVERITY_ERROR);
    FacesContext.getCurrentInstance().addMessage(id, msg);
}

```

*Listing 27: Fragment klasy ContextUtils z metodami internacjonalizacji*

### 3.4.4 Uwierzytelnienie i autoryzacja

Uwierzytelnianie użytkownika w systemie odbywa się na podstawie podanego unikalnego loginu i hasła. Hasło ulega przekształceniu do skrótu SHA-256, a następnie wprowadzane dane porównywane są z wzorcami przechowywanymi w bazie danych: login w postaci jawniej natomiast hasło w postaci zakodowanego skrótu SHA-256. Dane te są wprowadzane do formularza logowania, którego lokalizacja określona jest w pliku web.xml. Za autoryzacje w systemie odpowiedzialne są wbudowane mechanizmy serwera Payara, gdzie definiowany jest tzw. obszar bezpieczeństwa (ang. security realm), w tym przypadku wykorzystany został JDBCRealm, który określa obszar autoryzacji (realm) w oparciu o dane przechowywane w bazie danych. Mechanizm ten służy również do udostępniania użytkownikowi zasobów w oparciu uzyskaną z bazy role w systemie.

Dostęp do poszczególnych funkcji aplikacji zdefiniowany został poprzez podział widoku (opisany w rozdziale 3.4.2 Ujednolicony interfejs użytkownika) w zależności od przypisanej użytkownikowi roli. Dynamiczne generowanie menu w oparciu o poziomy dostępu możliwe jest dzięki elementowi <f:subview> i atrybutowi rendered, które przedstawiono na listingu 28.

```
<f:subview id="menuKontoNotLoggedIn" rendered="#{empty request.remoteUser}">
    <b:navCommandLink (...) value="${msg['menu.register']}" action="#{createAccountBean.init()}" />
</f:subview>
<f:subview id="admin" rendered="#{request.isUserInRole('Administrator')}">
    <b:navCommandLink (...) value="${msg['menu.adminPanel']}" href="#" action="listAccounts"/>
    <b:navCommandLink (...) value="${msg['admin.createUserAccount']}" href="#" action="createAccount"/>
</f:subview>
<f:subview id="assessor" rendered="#{request.isUserInRole('Assessor')}">
    <b:navCommandLink (...) value="${msg['menu.products.list']}" href="#" action="listCosmetics"/>
    <b:dropMenu (...) value="${msg['menu.categories']}">
        <b:navCommandLink (...) value="${msg['menu.list']}" href="#" action="listCategories"/>
        <b:navCommandLink (...) value="${msg['category.add']}" href="#" action="#{createCategoryBean.begin()}" />
    </b:dropMenu>
    <b:dropMenu (...) value="${msg['menu.analysis']}">
        <b:navCommandLink (...) value="${msg['menu.list']}" href="#" action="listAnalysis"/>
        <b:navCommandLink (...) value="${msg['analysis.add']}" href="#" action="#{createAnalysisBean.begin()}" />
    </b:dropMenu>
</f:subview>
```

*Listing 28: Fragment kodu pliku menu.xhtml przedstawiający podział w widoku*

Role i ograniczenia dostępu do wybranych zasobów aplikacji zdeklarowane są w deskryptorze wdrożenia aplikacji web.xml (listing 29).

```

<security-role>
    <description/>
    <role-name>Administrator</role-name>
</security-role>
<security-role>
    <description/>
    <role-name>Assessor</role-name>
</security-role>
<security-role>
    <description/>
    <role-name>Sales</role-name>
</security-role>
<security-role>
    <description/>
    <role-name>LabTechnician</role-name>
</security-role>

```

*Listing 29: Fragment kodu deskryptora wdrożenia web.xml - uwierzytelnianie i przypisanie poziomów dostępu*

### 3.4.5 Walidacja danych

Dane, które są wprowadzane przez użytkownika do programu podlegają walidacji już w warstwie widoku poprzez zastosowanie *BeanValidation*, który daje możliwość prostego weryfikowania poprawności pól obiektów w oparciu o zdefiniowane ograniczenia, dzięki czemu możliwa jest niezależna od modelu encji JPA relacyjnej bazy danych kontrola poprawności danych wprowadzonych w formularzach przez użytkownika. Aby wprowadzić ograniczenia stosujemy odpowiednie adnotacje dodane nad polami lub metodami dostępowymi (listing 25).

### 3.4.6 Obsługa błędów

Segment obsługi wyjątków, dzięki któremu następuje wyświetlenie komunikatu użytkownikowi znajduje się w warstwie widoku, w klasie *CosmeticController* (listing 5). Zgłoszone błędy nie zwracają docelowego komunikatu dla użytkownika, lecz zdefiniowany klucz. Klucz ten jest ostatecznie zmieniany na komunikat w wersji językowej interfejsu użytkownika. Zostało to szczegółowo opisane w rozdziale 3.3.4 Obsługa błędów. Komunikaty wyświetlane są na stronach JSF dzięki użyciu elementów *<h:messages(...)>* widocznych na przykładzie strony *createNewCosmetic.xhtml* w listingu 7. Oprócz tego w warstwie widoku następuje wywołanie *Loggera*, który zgłasza wystąpienie wyjątku, który może być zarejestrowany w dzienniku zdarzeń. Liczba segmentów obsługi wyjątków występujących w pojedynczej metodzie zależy od liczby możliwych do zgłoszenia wyjątków aplikacyjnych w trakcie wykonywania metod biznesowych co widać w metodzie *createCosmetic* w listingu 5.

## 3.5 Instrukcja wdrożenia

Uruchomienie stworzonego systemu wymaga odpowiedniego przygotowania środowiska, które posiada działający system operacyjny oraz współczesną przeglądarkę internetową. Ponadto musi posiadać zainstalowany serwer aplikacyjny Payara w wersji 5.183 oraz system JavaDB (Apache Derby) w wersji 10.14.2.0. Serwer aplikacyjny Payara oraz system

zarządzania bazami danych JavaDB powinny zostać uruchomione, a następnie należy wykonać czynności opisane w dalszych podrozdziałach.

### 3.5.1 Utworzenie bazy danych.

Baza danych powinna zostać utworzona z pomocą programu narzędziowego ij dostarczonego wraz z Apache Derby. Konfiguracja bazy danych zdefiniowana została w pliku *glassfish-resources.xml* (listing 30).

```
<resources>
  <jdbc-resource enabled="true" jndi-name="jdbc/SafetyAssesementDS" object-type="user" pool-
name="SafetyAssesementPool">
    </jdbc-resource>
    <jdbc-connection-pool transaction-isolation-level="read-committed" (...)>
      <property name="URL" value="jdbc:derby://localhost:1527/SafetyAssesement"/>
      <property name="serverName" value="localhost"/>
      <property name="PortNumber" value="1527"/>
      <property name="DatabaseName" value="SafetyAssesementDB"/>
      <property name="User" value="safety"/>
      <property name="Password" value="safety"/>
    </jdbc-connection-pool>
  </resources>
```

*Listing 30: Fragment kodu pliku konfiguracyjnego glassfish-resources.xml*

W wierszu poleceń należy uruchomić serwer bazy danych zgodnie z instrukcją pokazaną na listingu 31. Następnie w drugim oknie uruchomić program narzędziowy ij i utworzyć bazę danych zgodnie z listkiem 32.

```
java -jar $DERBY_HOME/lib/derbyrun.jar server start
```

*Listing 31: Uruchomienie serwera bazy danych*

```
java -jar $DERBY_HOME/lib/derbyrun.jar ij
```

```
ij>CONNECT
'jdbc:derby://localhost1527//SafetyAssesementDB; create=true; user = safety; password = safety;
ij>exit;
```

*Listing 32: Tworzenie bazy danych w programie ij*

### 3.5.2 Konfiguracja obszaru bezpieczeństwa na serwerze.

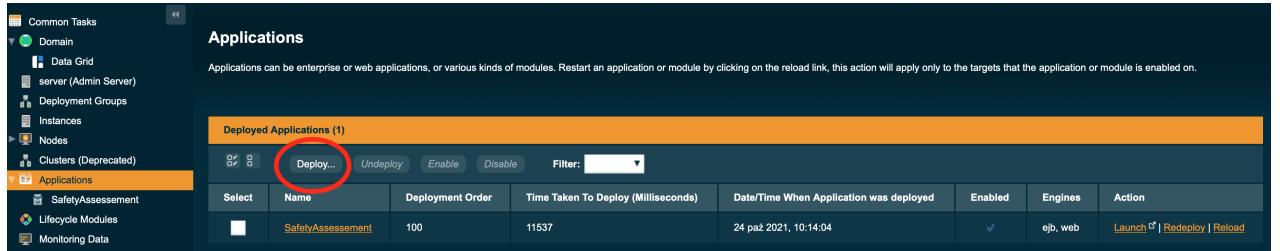
Należy otworzyć przeglądarkę internetową i w pasku adresu wpisać adres URL lokalnego serwera <http://localhost:48/common/index.jsf>. Z menu po lewej stronie należy wybrać kolejno Configurations → server-config → Security → Realms. Następnie wybrać przycisk New po prawej stronie. Otwarty formularz uzupełniamy danymi podanymi w listku 33 i naciskamy OK.

Ream Name SafetyAssessementJDBCRealm  
 Class Name: com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm  
 JAAS Context: jdbcRealm  
 JNDI: jdbc/SafetyAssessementDS  
 User Table: ACCOUNTS  
 User Name Column: LOGIN  
 Password Column: PASSWORD  
 Group Table: ACCOUNTS  
 Group Name Column: TYPE

*Listing 33: Dane do konfiguracji obszaru bezpieczeństwa w serwerze Payara*

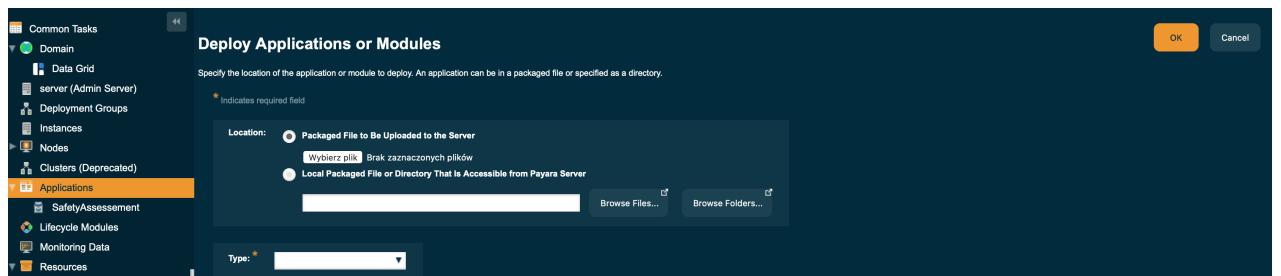
### 3.5.3 Wgranie aplikacji na serwer aplikacyjny i uruchomienie aplikacji

Pozostając na stronie <http://localhost:48/common/index.jsf> z menu po lewej stronie wybieramy przycisk Applications, a następnie przycisk Deploy wskazany na rysunku 22.



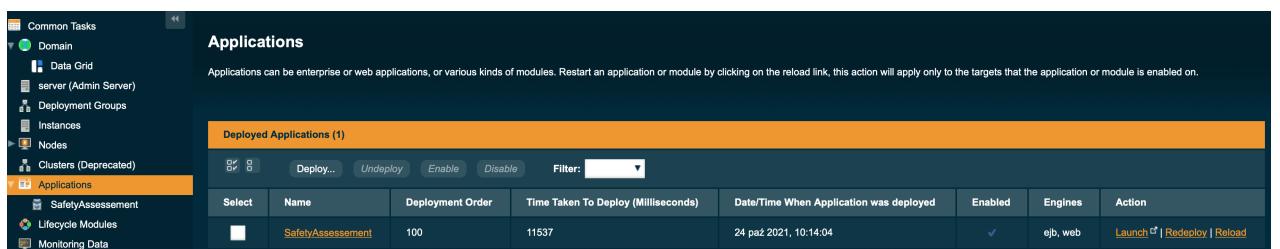
Rysunek 22: Wgranie aplikacji na serwer aplikacyjny Payara (etap pierwszy)

Poprzez użycie przycisku Wybierz plik (ang. Browse) wybieramy plik SafetyAssessement-1.0.war, który znajduje się na załączonym do raportu nośniku (rysunek 23).



Rysunek 23: Wgranie aplikacji na serwer aplikacyjny Payara (etap drugi)

Do pliku prowadzi ścieżka SafetyAssessement/target/SafetyAssessement-1.0.war. Po załączeniu pliku zatwierdzamy wybór przyciskiem OK. Aplikacja powinna być widoczna na serwerze. Aby ją uruchomić należy z kolumny Action wybrać przycisk Launch (rysunek 24).



Rysunek 24: Uruchomienie aplikacji znajdującej się na serwerze aplikacyjnym Payara

### **3.5.4 Umieszczenie w bazie struktur baz danych i danych inicjujących**

Aby użytkownik mógł korzystać z funkcjonalności systemu wymagane jest zainicjowanie wstępnych danych zawierających m.in. konta użytkowników. W tym celu należy wczytać plik *dbStartingData.sql* zgodnie z instrukcją przedstawioną w listingu .

```
ij>run 'nazwa_katalogu/SafetyAssessemment/src/main/resources/dbStartingData.sql'
```

*Listing 34: Inicjalizacja danych*

Wdrożona aplikacja dostępna jest pod adresem <http://localhost:8080/SafetyAssessemment>. Jeśli wszystkie powyższe kroki zostały wykonane poprawnie możliwe będzie zalogowanie się do aplikacji z wykorzystaniem jednego z predefiniowanych kont, znajdujących się w pliku z danymi inicjującymi. Loginy oraz hasła dla tych kont zawarte są w tabeli 2.

Tabela 2: Predefiniowane konta i dane do uwierzytelnienia

Typ konta	Login	hasło
Administrator	administrator1	Administrator_1
Administrator	administrator2	Administrator_2
Handlowiec	handlowiec1	Handlowiec_1
Assessor	assessor1	Assessor_1
Assessor	assessor2	Assessor_2
Laborant	laborant1	Laborant_1

## **3.6 Podsumowanie**

Celem pracy było stworzenie wielodostępnego systemu informatycznego wspomagającego tworzenie raportów bezpieczeństwa produktów kosmetycznych. System ma za zadanie umożliwić uproszczenie przygotowania oceny bezpieczeństwa kosmetyków. Zadanie to jest być realizowane poprzez automatyczne przypisanie wymaganych analiz oraz generowanie opisu toksykologicznego. Analizy przypisywane są na podstawie wybranej dla kosmetyku kategorii, natomiast opis generowany jest w oparciu o skład podany przy tworzeniu kosmetyku do oceny . Przygotowany system realizuje to zadanie.

Aplikacja została wykonana na platformie Java Enterprise Edition oraz osadzona na serwerze aplikacyjnym Payara. Zgodnie z założeniami funkcjonalnymi stworzony system zapewnia cztery poziomy dostępu dla zalogowanych użytkowników oraz funkcje użytkownika nieuwierzytelnionego. Administrator posiada możliwość zarządzania kontami wszystkich użytkowników, m. in. ich edycji, weryfikacji wraz z przydzieleniem poziomu dostępu oraz aktywowania i dezaktywowania kont użytkowników. Handlowiec zarządza zleceniami na oceny bezpieczeństwa kosmetyków: ma możliwość wyświetlenia listy kosmetyków, dodać kosmetyk do oceny, edytować go lub usunąć, a także wyświetlić jego szczegóły. Laborant ma dostęp do listy kosmetyków oraz szczegółów każdego kosmetyku. W szczegółach znajduje się lista analiz dla danego kosmetyku oraz edytowalne pola, do których pracownik z tym poziomem dostępu może wprowadzić wyniki uzyskane w badaniach laboratoryjnych. Dodatkowo ma dostęp do listy analiz, które może dodawać i edytować. Użytkownik z poziomem dostępu Assessor ma dostęp do listy kosmetyków, na której może przyjąć kosmetyk do oceny lub z oceny zrezygnować, oraz do szczegółów kosmetyku, gdzie ma

możliwość wygenerować opis toksykologiczny. Assessor jest również odpowiedzialny za zarządzanie kategoriami produktu oraz surowcami oraz, podobnie jak Laborant, analizami, co oznacza, że może wyświetlać ich listy, edytować poszczególne pozycje na liście oraz dodawać nowe elementy.

W stworzonym wielodostępnym systemie informatycznym interfejs użytkownika został zdefiniowany w oparciu o technologię Java Server Faces oraz wykorzystuje komponenty biblioteki BootsFaces. Biblioteka ta umożliwiając wykorzystanie gotowych komponentów GUI wspomaga tworzenie zaawansowanego i funkcjonalnego interfejsu użytkownika w sposób stosunkowo prosty, przy czym dający estetyczny efekt końcowy. Tabele tworzone w oparciu o tą bibliotekę są stronicowane, natomiast zawarte w nich elementy można segregować w kolumnach, przeszukiwać oraz pobierać w formie plików PDF oraz Excel. Zastosowanie mechanizmu internacjonalizacji umożliwia wyświetlanie interfejsu użytkownika stworzonego systemu w języku polskim i angielskim. Jednocześnie stworzone oprogramowanie posiada możliwość poszerzenia o dalsze języki.

W aplikacji zaimplementowana została obsługa błędów. Aby zapobiec uszkodzeniu danych każdy wyjątek aplikacyjny, który zostaje zgłoszony wymusza wycofanie transakcji aplikacyjnej. Zastosowanie propagacji wyjątku do warstwy widoku umożliwia jego dalszą obsługę przez wyświetlenie odpowiedniego komunikatu, mającego na celu poinformowanie użytkownika o wystąpieniu błędu i możliwej reakcji. Metoda fabrykująca obiekt wyjątku ma zdefiniowane klucze komunikatów błędów, które przekazane zostają użytkownikowi. Dzięki temu w stosunku do komunikatów również możliwe jest zastosowanie internacjonalizacji i wyświetlanie komunikatów w preferowanym przez użytkownika języku.

W aplikacji zastosowana została strategia CMT zarządzania transakcjami aplikacyjnymi przez kontener EJB. Obsługa uprawnień oraz kontroli dostępu została oddelegowana do kontenera EJB oraz serwera aplikacji. Podstawą przyznawania dostępu do metod biznesowych w punktach dostępowych oraz fasadach jest stosowanie odpowiednich adnotacji. W aplikacji zaimplementowano mechanizm umożliwiający zgłaszanie zdarzeń wynikających z podejmowanych przez użytkowników systemu czynności biznesowych w dziennikach zdarzeń. Dzienniki zdarzeń zawierają również informacje o granicach transakcji bazodanowych wraz ze statusem ich zakończenia. Wykorzystanie mechanizmu blokad optymistycznych umożliwiło ochronę przed niespójnością danych przy jednoczesnej pracy wielu użytkowników.

Podsumowując, cel pracy został zrealizowany, a stworzony system informatyczny spełnia wszystkie wymagania postawione w założeniach na początku pracy, zarówno funkcjonalne jak i nie funkcjonalne.

## 4 Źródła

### Bibliografia

1. Burke B., Monson-Haefel R.: Enterprise JavaBeans 3.0, Helion, 2007
2. Heffelfinger D.R.: Java EE 6. Tworzenie aplikacji w NetBeans 7, Helion, 2014
3. Bauer C., King G.: Java Persistence. Programowanie aplikacji bazodanowych w Hibernate., Helion, 2016
4. Horstman C.S.: Java. Podstawy., Helion, 2019
5. Layka V.: Java. Projektowanie aplikacji WWW, Wydanie I, Helion, 2016
6. Darwin I.F.: Java. Receptury., Wydanie III, Helion, 2015
7. Leonard A.: JavaServerFaces 2.2 Mistrzowskie programowanie., Wydanie II, Helion, 2016
8. URL: GlassFish Server 5 Open Source Edition Documentation, [Online, data dostępu 27.06.2021], <https://javaee.github.io/glassfish/documentation>
9. URL: The World's Largest Web Developer Site, (dostęp: listopad 2021)
10. URL: Witryna projektu lombok., (dostęp: wrzesień 2021)
11. URL: Witryna zawierająca pokazowe rozwiązania dla Frameworka Bootsfaces., (dostęp: październik 2021)

### Indeks tabel

Tabela 1: Przypadki użycia w zależności od poziomu dostępu użytkownika systemu.....	6
Tabela 2: Predefiniowane konta i dane do uwierzytelnienia.....	51

### Indeks listingów

Listing 1: Fragment kodu klasy encyjnej Cosmetic.....	28
Listing 2: Fragment kodu strony listCosmetics.xhtml.....	30
Listing 3: Fragment kodu klasy ListCosmeticBean.....	30
Listing 4: Fragment kodu klasy CosmeticEndpoint.....	30
Listing 5: Fragment kodu klasy CosmeticController.....	30
Listing 6: Metoda findAll z interfejsu AbstractFacade.....	30
Listing 7: Fragment strony createNewCosmetic.xhtml - formularz tworzenia kosmetyku.....	31
Listing 8: Fragment kodu klasy CreateCosmeticBean.....	32
Listing 9: Metoda createCosmetic w klasie CosmeticController.....	32
Listing 10: Metoda <i>createCosmetic</i> klasy CosmeticEndpoint.....	33
Listing 11: Fragment klasy CosmeticManager.....	33
Listing 12: Metoda create w klasie CosmeticFacade.....	34
Listing 13: Fragment kodu klasy CosmeticController – edycja kosmetyku.....	35
Listing 14: Metoda saveCosmeticAfterEdition z klasy CosmeticEndpoint.....	35
Listing 15: Metoda edit w klasie CosmeticFacade.....	36
Listing 16: Fragment klasy CosmeticController - usuwanie kosmetyku.....	37
Listing 17: Fragment klasy CosmeticEndpoint – usuwanie kosmetyku.....	37
Listing 18: Fragment klasy CosmeticManager – usuwanie kosmetyku.....	38
Listing 19: Fragment kodu klasy CosmeticFacade z metodą remove.....	38
Listing 20: Fragment kodu klasy AbstractFacade.....	39
Listing 21: Klasa AbstractManager.....	40
Listing 22: Definicja klasy LoggingInterceptor.....	41
Listing 23: Fragment pliku web.xml - deklaracja stron błędów java.lang.RuntimeException.....	42

Listing 24: Fragment kodu klasy CosmeticException.....	43
Listing 25: Fragment kodu klasy CosmeticDTO.....	44
Listing 26: Fragment pliku faces-config.xml zawierający definicję zasobów potrzebnych do internacjonalizacji.....	46
Listing 27: Fragment klasy ContextUtils z metodami internacjonalizacji.....	46
Listing 28: Fragment kodu pliku <i>menu.xhtml</i> przedstawiający podział w widoku.....	47
Listing 29: Fragment kodu deskryptora wdrożenia web.xml - uwierzytelnianie i przypisanie poziomów dostępu.....	48
Listing 30: Fragment kodu pliku konfiguracyjnego glassfish-resources.xml.....	49
Listing 31: Uruchomienie serwera bazy danych.....	49
Listing 32: Tworzenie bazy danych w programie ij.....	49
Listing 33: Dane do konfiguracji obszaru bezpieczeństwa w serwerze Payara.....	50
Listing 34: Inicjalizacja danych.....	51

## Indeks rysunków

Rysunek 1: Diagram przypadków użycia dla poziomu dostępu: Gość.....	6
Rysunek 2: Diagram przypadków użycia dla poziomu dostępu: Handlowiec.....	7
Rysunek 3: Diagram przypadków użycia dla poziomu dostępu: Laborant.....	7
Rysunek 4: Diagram przypadków użycia dla poziomu dostępu: Assessor.....	7
Rysunek 5: Diagram przypadków użycia dla poziomu dostępu: Administrator.....	8
Rysunek 6: Lista kont użytkowników.....	12
Rysunek 7: Lista produktów kosmetycznych do oceny.....	15
Rysunek 8: Formularz dodawania nowego kosmetyku.....	16
Rysunek 9: Formularz edycji kosmetyku.....	17
Rysunek 10: Strona potwierdzenia usunięcia kosmetyku.....	18
Rysunek 11: <i>Strona szczegółów kosmetyku</i> .....	19
Rysunek 12: Lista kategorii kosmetyku.....	20
Rysunek 13: Diagram komponentów aplikacji dla wybranych przypadków użycia.....	25
Rysunek 14: Model relacyjnej bazy danych.....	26
Rysunek 15: Diagram klas encyjnych.....	27
Rysunek 16: Wygląd strony z formularzem logowania.....	45
Rysunek 17: Wygląd paska menu dla poziomu dostępu Administrator.....	45
Rysunek 18: Wygląd paska menu dla poziomu dostępu Handlowiec.....	45
Rysunek 19: Wygląd paska menu dla poziomu dostępu Assessor.....	45
Rysunek 20: Wygląd paska menu dla poziomu dostępu Laborant.....	45
Rysunek 21: Fragment tabeli zawierającej klucze do internacjonalizacji.....	46
Rysunek 22: Wgranie aplikacji na serwer aplikacyjny Payara (etap pierwszy).....	50
Rysunek 23: Wgranie aplikacji na serwer aplikacyjny Payara (etap drugi).....	50
Rysunek 24: Uruchomienie aplikacji znajdującej się na serwerze aplikacyjnym Payara.....	50