



Politechnika
Wrocławska

BAZY DANYCH II
DOKUMENTACJA DO PROJEKTU

BAZA DANYCH DLA SKLEPU KOMPUTEROWEGO

Autorzy:

Kamil CIEŚLIK, 226138

Patryk ZDRAL, 220878

Prowadzący:

dr inż. Roman PTAK, W4/K9

Termin zajęć:

środa, 13:15 - 15:00

Wrocław, 2018 r.

Spis treści

1.	WSTĘP	5
1.1.	Cel projektu.....	5
1.2.	Zakres projektu	5
1.3.	Etapy i zakres projektu.....	5
1.4.	Opis działania i funkcje systemu.....	6
2.	Analiza wymagań.....	6
2.1.	Opis biznesowy „świata rzeczywistego”	6
2.1.1.	Opis zasobów ludzkich	6
2.1.2.	Przepis i strategia firmy	7
2.2.	Wymagania funkcjonalne	8
2.3.	Wymagania niefunkcjonalne.....	8
2.3.1.	Wykorzystywane technologie i narzędzia.....	8
2.3.2.	Wymagania dotyczące rozmiaru bazy danych	9
2.3.3.	Wymagania dotyczące bezpieczeństwa systemu	9
2.4.	Słownik danych przedstawiający opis atrybutów	10
3.	Projekt systemu	11
3.1.	Projekt bazy danych	11
3.1.1.	Analiza rzeczywistości i uproszczony model konceptualny	11
3.1.2.	Model logiczny	12
3.1.3.	Model fizyczny i ograniczenia integralności danych	13
3.1.4.	Projekt mechanizmów bezpieczeństwa na poziomie bazy danych	14
3.1.5.	Inne elementy schematu – mechanizmy przetwarzania danych.....	14
3.1.5.a.	Widoki	14
3.1.5.b.	Wyzwalacze	16
3.1.5.c.	Indeksy	17
3.1.5.d.	Ograniczenia na poprawność danych.....	18
3.2.	Projekt aplikacji użytkownika.....	18
3.2.1.	Architektura aplikacji i diagramy projektowe.....	18
3.2.1.a.	Architektura aplikacji wg wzorca Data Access Object	18
3.2.1.b.	Diagram przypadków użycia	19
3.2.2.	Interfejs graficzny i struktura menu	20
3.2.3.	Projekt wybranych funkcji systemu	23
3.2.3.a.	Interfejs metod CRUD dla obiektów bazy danych.....	23
3.2.3.b.	Przykład implementacji kilku metod interfejsu CRUD dla klasy Customer	23
3.2.3.c.	Metoda uwierzytelniania podczas logowania dla zarejestrowanego klienta	24
3.2.4.	Metoda podłączania do bazy danych - integracja z bazą danych	24
3.2.5.	Projekt zabezpieczeń na poziomie aplikacji.....	25
3.2.5.a.	Wyrażenie regularne opisujące łańcuch znaków hasła klienckiego	25
3.2.5.b.	Kontrola poprawności danych dla tworzonych oraz modyfikowanych rekordów	25
3.2.5.c.	Zapytania potwierdzające w przypadku usuwania rekordów.....	26
4.	Implementacja systemu baz danych.....	26

4.1.	Tworzenie tabel i definiowanie ograniczeń	26
4.1.1.	Tabela konta użytkownika - account.....	26
4.1.2.	Tabela adresu zamieszkania klienta oraz lokalizacji sklepu - address	26
4.1.3.	Tabela danych klienta sklepu - customer	27
4.1.4.	Tabela kodów rabatowych użytkowników - discount	27
4.1.5.	Tabela danych pracowników sklepu stacjonarnego - employee	27
4.1.6.	Tabela danych zamówień - order	28
4.1.7.	Tabela produktów - product.....	28
4.1.8.	Tabela kategorii produktów - product_category	28
4.1.9.	Tabela zdjęć produktów - product_photo.....	29
4.1.10.	Tabela zestawów produktów - product_set.....	29
4.1.11.	Tabela egzemplarzy produktów - used_specimen.....	29
4.1.12.	Tabela danych o sklepach stacjonarnych - stationary_shop.....	30
4.1.13.	Tabela pomocnicza dot. szczegółów zamówień - connector_order_details	30
4.1.14.	Tabela pomocnicza dot. szczegółów zestawów - connector_set_details	30
4.2.	Implementacja mechanizmów przetwarzania danych	31
4.2.1.	Wyzwalacze tabeli produktów - product.....	31
4.2.1.a.	BEFORE INSERT oraz BEFORE UPDATE.....	31
4.2.1.b.	BEFORE DELETE	31
4.2.2.	Wyzwalacze tabeli egzemplarzy produktów - used_specimen	32
4.2.2.a.	BEFORE INSERT	32
4.2.2.b.	AFTER INSERT oraz AFTER DELETE.....	33
4.2.3.	Wyzwalacze tabeli pomocniczej dot. szczegółów zamówień - connector_order_details	33
4.2.3.a.	AFTER INSERT oraz AFTER UPDATE.....	33
4.2.4.	Wyzwalacze tabeli zamówień - order	34
4.2.4.a.	BEFORE INSERT	34
4.2.5.	Widok wyświetlające dane ogólne o zestawach produktów	34
4.2.6.	Widok wyświetlający produkty wchodzące w skład zestawów produktów	34
4.2.7.	Widok wyświetlający dane ogólne o dostępnych w sklepie produktach.....	35
4.2.8.	Widok wyświetlający dane o klientach.....	35
4.2.9.	Widok wyświetlający informacje o pracownikach oraz ich miejscu pracy.....	36
4.2.10.	Widok wyświetlający najważniejsze szczegóły zamówienia.....	36
4.2.11.	Widok wyświetlający egzemplarze produktów zamówienia	37
4.3.	Implementacja uprawnień i innych ograniczeń.....	37
4.4.	Testowanie bazy danych dla przykładowych danych	38
4.4.1.	Wypełnienie bazy danych przykładowymi wartościami rekordów.....	38
4.4.2.	Przykłady użycia zaimplementowanych widoków	44
4.4.3.	Przykład działania mechanizmów ograniczeń wprowadzanych danych.....	44
5.	Implementacja i testowanie aplikacji	45
5.1.	Instalacja i konfiguracja systemu	45
5.2.	Instrukcja użytkowania aplikacji.....	45
5.3.	Testowanie wybranych funkcjonalności aplikacji	48

5.3.1.	Wprowadzenie kodu rabatowego przez niezalogowanego użytkownika	48
5.3.2.	Złożenie zamówienia	49
5.3.3.	Logowanie do panelu użytkownika	49
5.3.4.	Otrzymywanie kodów rabatowych po odpowiednio dokonanych zakupie	50
5.3.5.	Próba zakupu pojedynczych produktów danego zestawu z widoku zestawów	50
5.3.6.	Próba dodania do koszyka produktu, którego ilość wynosi 0	51
5.3.7.	Wyszukiwanie produktów	51
5.3.8.	Zaawansowane wyszukiwanie produktów	52
5.3.9.	Działanie kodu rabatowego	52
5.4.	Omówienie wybranych rozwiązań programistycznych	53
5.4.1.	Metoda połączenia z bazą danych	53
5.4.2.	Implementacja wybranych funkcjonalności sklepu	54
5.4.3.	Implementacja mechanizmów bezpieczeństwa	55
6.	Wnioski	57

1. WSTĘP

1.1. Cel projektu

Celem projektu jest stworzenie bazy danych oraz aplikacji desktopowej wraz z interfejsem graficznym dla tematu - „Baza danych dla sklepu komputerowego”. Aplikacja ma na celu dać klientom sklepów stacjonarnych możliwość dokonywania zakupów bez potrzeby kontaktu ze sprzedawcą, pozyskiwanie benefitów zachęcających do dalszych zakupów oraz intuicyjne przeglądanie historii wcześniejszych zamówień. Pracownikom aplikacja pozwoli zarządzać sklepem komputerowym.

1.2. Zakres projektu

Końcowym produktem przedsięwzięcia projektowego będzie aplikacja desktopowa zainstalowana na interaktywnych urządzeniach sprzedażowych w punktach stacjonarnych - sklepach firmy komputerowej. Celem jej istnienia jest odpowiedź na oczekiwania klientów, takich jak: skrócenie czasu oczekiwania w kolejkach do konsultantów, bardziej intuicyjna forma przeglądania produktów, możliwość uzyskiwania rabatów za pomocą członkostwa w klubie promocyjnym, które pozwoli również na wgląd w historię poprzednich zamówień, grupowanie produktów w zestawy np. komputer stacjonarny.

Produkty cząstkowe projektu:

- panel sprzedażowy (intuicyjne wyszukiwanie produktów, dodawanie/usuwanie z koszyka, kompletowanie zamówienia oraz jego realizacja, formularz danych osobowych - w przypadku nieposiadania konta),
- przejrzysty interfejs graficzny,
- system wysyłki podsumowania zamówienia w formie faktury w formie elektronicznej,
- panel rejestracji oraz logowania w klubie rabatowo-usługowym,
- system rabatowy dla zarejestrowanych klientów (możliwość uzyskiwania kodów rabatowych na konkretne kategorie produktów po spełnieniu określonych warunków, wykorzystywanie kodów rabatowych - obniżanie cen zamówień),
- uproszczony system pracowniczy:
 - możliwość grupowania produktów w tzw. zestawy,
 - nadawanie klientom kodów rabatowych w formie zadośćuczynienia,
 - wgląd w historię zamówień, podsumowań finansowych.

1.3. Etapy i zakres projektu

- 1) Sformułowanie założeń wstępnych,
- 2) Utworzenie diagramu przypadków użycia oraz diagramów związków encji,
- 3) Stworzenie lokalnej relacyjnej bazy danych,
- 4) Napisanie aplikacji desktopowej korzystającej z lokalnej bazy danych,
- 5) Wykonanie testów poprawności działania.

1.4. Opis działania i funkcje systemu

System ma na celu umożliwiać zarządzanie sklepem komputerowym w oparciu o relacyjną bazę danych zawierającą dane o klientach, pracownikach, stanie magazynu, zamówieniach, zestawach produktów, kodach rabatowych, sklepach, w których został wprowadzony.

System będzie przede wszystkim pozwalał na dokonywanie przez klienta zakupów stacjonarnych w tzw. interaktywnych kioskach bez konieczności bezpośredniego kontaktu z konsultantem. W panelu klienta dużą uwagę zwrócono na zaawansowany oraz intuicyjny interfejs graficzny.

Klientów podzielono na klientów niezarejestrowanych oraz posiadających konto klubowe. Zarówno niezarejestrowani jak i zarejestrowani będą mogli dokonywać zakupów, lecz Ci drudzy będą mogli czerpać korzyści wynikające z otrzymanych kodów rabatowych czy możliwości wglądu do historii zamówień.

Cały szereg możliwości klientów dokładnie został opisany w dalszej części dokumentacji.

Pracownik sklepu będzie zarządzać bazą danych poprzez specjalną dedykowaną aplikację, która pozwoli m.in. dodać nowy towar, usunąć go, zmodyfikować, pogrupować produkty w zestawy zachęcające klientów do większych zakupów. Ponadto będzie miał również dostęp do danych klientów, podsumowań ich zamówień oraz zestawień dochodów wg sortowanych wg różnych kryteriów. Pracownik chcący uzyskać dostęp do trybu zarządzania, będzie używał specjalnego panelu logowania. Powyższy panel będzie miał zaawansowane funkcje, lecz uproszczony interfejs graficzny.

2. Analiza wymagań

2.1. Opis biznesowy „świata rzeczywistego”

2.1.1. Opis zasobów ludzkich

Niezarejestrowany w systemie klient ma możliwość przeglądania asortymentu sklepu komputerowego za pomocą aplikacji desktopowej z poziomu kiosku w stacjonarnym punkcie firmy, tworzenia koszyka produktów, realizowania zamówień oraz utworzenia konta.

Każdy produkt przynależy do danej kategorii, posiada opis, cenę (brutto, netto), określoną ilość dostępnych sztuk oraz fotografie. Podczas tworzenia zamówienia (dodawanie produktów do koszyka) do zamówienia przypisany zostaje egzemplarz wybranego produktu (jeżeli jest dostępny) z wcześniej wygenerowanym unikalnym kodem kreskowym. Klient zatwierdza lub odrzuca aktualny stan zamówienia. W zależności od wyboru następuje jego realizacja (wybór metody płatności, dodanie danych klienta do bazy, płatność, wydanie towaru) bądź odrzucenie (wycofanie produktów z koszyka, usunięcie wcześniej utworzonych egzemplarzy, przywrócenie poprzedniej liczby stanu poszczególnych produktów).

Zarejestrowanemu klientowi przysługują benefity w postaci kodów rabatowych przyznawanych do obniżania sumy kosztów wszystkich produktów, przynależnych do kategorii, której rabat dotyczy, na wybranym zamówieniu.

Kod rabatowy generowany i przypisywany jest do wybranej kategorii produktów i do zarejestrowanego klienta w sytuacji realizacji zamówienia z przynajmniej trzema produktami jednej kategorii.

Rabat kodu rabatowego może ulec zmianie. W sytuacji spełnienia powyższego warunku ponownie rabat zostaje zwiększony o 10%, pod warunkiem, że kod nie został już wcześniej zużyty.

Po skompletowaniu produktów zamówienia klient posiadający konto w serwisie sklepu ma możliwość zmniejszenia jego kosztów za pomocą takiego kodu. Od całkowitej sumy zamówienia zostaną odjęte koszty równe sumie rabatu pomnożonej przez sumę wszystkich produktów kategorii, której rabat dotyczy.

Rejestracja daje klientowi również możliwość przeglądania historii wcześniejszych zamówień, wykorzystanych oraz niewykorzystanych kodów rabatowych oraz modyfikowania/aktualizacji danych osobowych.

Pracownik sklepu podając określone przez administratora bazy danych hasło może za pomocą powyższego systemu wejść do odpowiadającego jego osobie panelu pracownika i dodać do katalogu produktów nowe przedmioty, usuwać je lub zmieniać, poszerzać ofertę dostępnych kategorii. Ponadto może tworzyć zestawy produktów np. komputer, składające się z wybranych przez niego produktów w celu zachęcenia klienta do zakupu większej ilości sprzętu jednocześnie.

Pracownik może również usunąć klientów, zmodyfikować ich dane oraz w nadzwyczajnych przypadkach udzielić klientowi zadośćuczynienia w postaci rabatu.

2.1.2. Przepis i strategia firmy

Celem firmy, w odpowiedzi na duży ruch klientów w sklepach stacjonarnych, jest jak największe ułatwienie ważnego aspektu dla pracowników oraz odbiorców oferowanego sprzętu jakim jest marnotrawienie czasu czekając w długich kolejkach. Montaż kiosków oraz stworzenie odpowiedniego oprogramowania do obsługi klientów ma na celu zwiększyć sprzedaż w określonej jednostce czasu, odciążyć pracowników oraz zwiększyć zadowolenie z oferowanych usług. Wprowadzony system powinien być intuicyjny, nie generować żadnych błędów, działać stabilnie i posiadać szeroką gamę dostępnych opcji. Przepisy firmy opracowane są przy założeniu jak największej dbałości o klienta.

Pracownicy firmy odpowiadają materialnie za niezgodność danych ze stanem magazynowym sklepu komputerowego, za działanie na niekorzyść klienta w postaci modyfikowania jego danych bądź usuwania ich bez ważnego powodu oraz za niepodyktowane zadośćuczynieniem przydzielanie klientom rabatów.

2.2. Wymagania funkcjonalne

- Klient niezarejestrowany:
 - Przeglądanie oferty sklepu komputerowego,
 - Tworzenie własnego koszyka produktów oraz składanie zamówienia,
 - Możliwość rejestracji.
- Klient zarejestrowany:
 - Przeglądanie oferty sklepu komputerowego,
 - Modyfikowanie/aktualizacja danych osobowych,
 - Możliwość logowania się do sklepu za pomocą ustalonego wcześniej hasła,
 - Tworzenie własnego koszyka produktów oraz składanie zamówienia,
 - Zbieranie benefitów w postaci kodów rabatowych,
 - Wgląd w historię zamówień i zdobytych rabatów.
- Pracownik:
 - usuwanie istniejących klientów,
 - modyfikowanie danych istniejących klientów,
 - dodawanie nowych produktów do bazy,
 - dodawanie galerii zdjęć produktów do bazy i przypisywanie ich do produktów,
 - grupowanie produktów w zestawy,
 - usuwanie produktów z bazy,
 - modyfikowanie danych produktów w bazie,
 - wgląd w historię zamówień klientów, przeglądanie statystyk i podsumowań.

2.3. Wymagania нефункциональные

- czytelny i przejrzysty graficzny panel użytkownika w szczególności - panel klienta,
- obsługa w czasie rzeczywistym,
- całość stworzona w języku angielskim,
- modyfikowanie bazy danych może odbywać się tylko przez uprawnionych pracowników.

2.3.1. Wykorzystywane technologie i narzędzia

- Technologie:
 - SQL,
 - Hibernate (HQL),
 - Java,
 - JavaFX,
 - UML.
- Narzędzia:
 - IntelliJ IDEA 2016.3.2(64),
 - MySQL Workbench 6.3 CE
 - Visual Paradigm Community Edition 14.2,
 - Microsoft Word 2016.

2.3.2. Wymagania dotyczące rozmiaru bazy danych

- jedna baza sklepu danych komputerowego,
- około kilkuset członków sklepu komputerowego, wraz z danymi osobowymi i dostępem do konta,
- kilkunastu pracowników sklepu, wraz z niezbędnymi danymi,
- kilkaset oferowanych produktów w bazie,
- 100 – 200 transakcji dziennie.

2.3.3. Wymagania dotyczące bezpieczeństwa systemu

- dostęp do panelu pracowniczego zabezpieczony hasłem,
- logowaniem do panelu użytkownika zarejestrowanego zabezpieczone loginem i hasłem,
- sprawdzanie poprawności wprowadzanych danych na poziomie aplikacji,
- ograniczenia związane z dopuszczalnymi wartościami danych na poziomie bazy danych,
- szyfrowanie hasła użytkownik na poziomie bazy danych, niedostępne dla innych użytkowników.

2.4. Słownik danych przedstawiający opis atrybutów

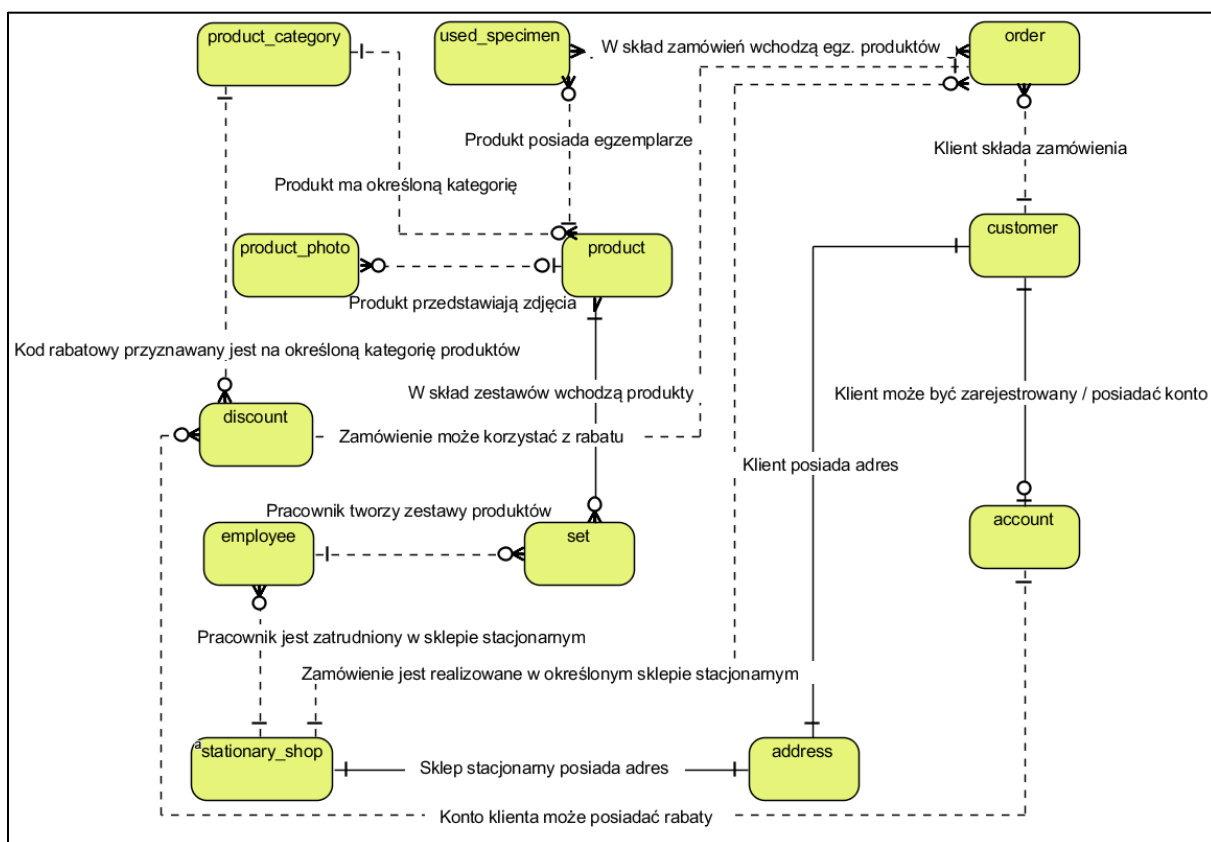
Nazwa zbioru encji	Opis zbioru encji	Atrybuty	Opis atrybutu
customer	Klient	id	Jednoznacznie identyfikuje klienta.
		address_id	Powiązanie z adresem klienta.
		first_name	Imię klienta.
		last_name	Nazwisko klienta.
		email	E-mail klienta.
		phone_number	Numer tel. klienta.
account	Konto uprawniające do korzystania z funkcji rabatowych systemu.	id	Jednoznacznie identyfikuje konto klienta.
		login	Login do konta klienta.
		password	Hasło do konta klienta.
		date_of_registration	Data rejestracji konta.
		customer_id	Powiązanie z klientem, do którego należy konto.
address	Adres klienta, sklepu stacjonarnego.	id	Jednoznacznie identyfikuje adres.
		street	Ulica.
		city	Miasto.
		postal_code	Kod pocztowy.
		country	Kraj.
order	Zamówienie zawierające wybrane egzemplarze produktów.	id	Jednoznacznie identyfikuje zamówienie.
		customer_id	Powiązanie z klientem tworzącym zamówienie.
		date	Data realizacji zamówienia.
		stationary_shop_id	Powiązanie ze sklepem, w którym zamówienie jest realizowane.
		promotion_price	Cena zamówienia po uwzględnieniu ew. rabatu.
		discount_id	Powiązanie z ew. rabatem.
product	Produkt oferty sklepu.	id	Jednoznacznie identyfikuje produkt.
		name	Nazwa produktu.
		product_category_id	Powiązanie z kategorią produktu.
		vat_rate	Cena sprzedaży netto.
		selling_price_netto	Cena sprzedaży brutto.
		selling_price_brutto	Wartość podatku.
		amount	Ilość dostępnych produktów.
product_category	Kategoria produktów, np. karty graficzne.	id	Jednoznacznie identyfikuje kategorie produktu.
		name	Nazwa kategorii.
product_photo	Zdjęcie produktu. Kilka zdjęć może być przypisanych do jednego produktu.	id	Jednoznacznie identyfikuje zdjęcie produktu.
		product_id	Powiązanie z produktem.
		photo_path	Lokalizacja zdjęcia.
product_set	Zestaw produktów, np. komputer stacjonarny. Zestawy kompletowane przez pracowników.	id	Jednoznacznie identyfikuje zestaw produktów.
		name	Nazwa zestawu.
		employee_id	Powiązanie z pracownikiem, który kompletuje zestaw.
used_specimen	Wykorzystywany egzemplarz produktów będący w koszyku kupującego.	id	Jednoznacznie identyfikuje egzemplarz produktu.
		product_id	Powiązanie z produktem.
		name	Nazwa produktu.
		barcode	Unikalny kod egzemplarza.
		purchase_price_brutto	Cena kupna brutto.
		purchase_price_netto	Cena kupna netto.
stationary_shop	Dane sklepu stacjonarnego.	vat_rate	Wartość podatku.
		id	Jednoznacznie identyfikuje sklep stacjonarny.
		name	Nazwa własna sklepu stacjonarnego.
employee	Pracownik.	address_id	Powiązanie z adresem sklepu stacjonarnego.
		id	Jednoznacznie identyfikuje pracownika.
		place_of_work_id	Powiązanie ze sklepem stacjonarnym - miejscem pracy pracownika.
		first_name	Imię pracownika.
		last_name	Nazwisko pracownika.
discount	Rabat o danej wartości przypisywany do konta klienta z możliwością wykorzystania do danej kategorii produktów.	phone_number	Numer tel. pracownika.
		id	Jednoznacznie identyfikuje klienta.
		discount_code	Kod rabatowy.
		account_id	Powiązanie z kontem, do którego rabat należy.
		product_category_id	Powiązanie z kategorią produktów na którą rabat został udzielony.
		discount_percentage	Procent udzielonego rabatu.
		is_used	Status rabatu - czy wykorzystano.

3. Projekt systemu

3.1. Projekt bazy danych

3.1.1. Analiza rzeczywistości i uproszczony model konceptualny

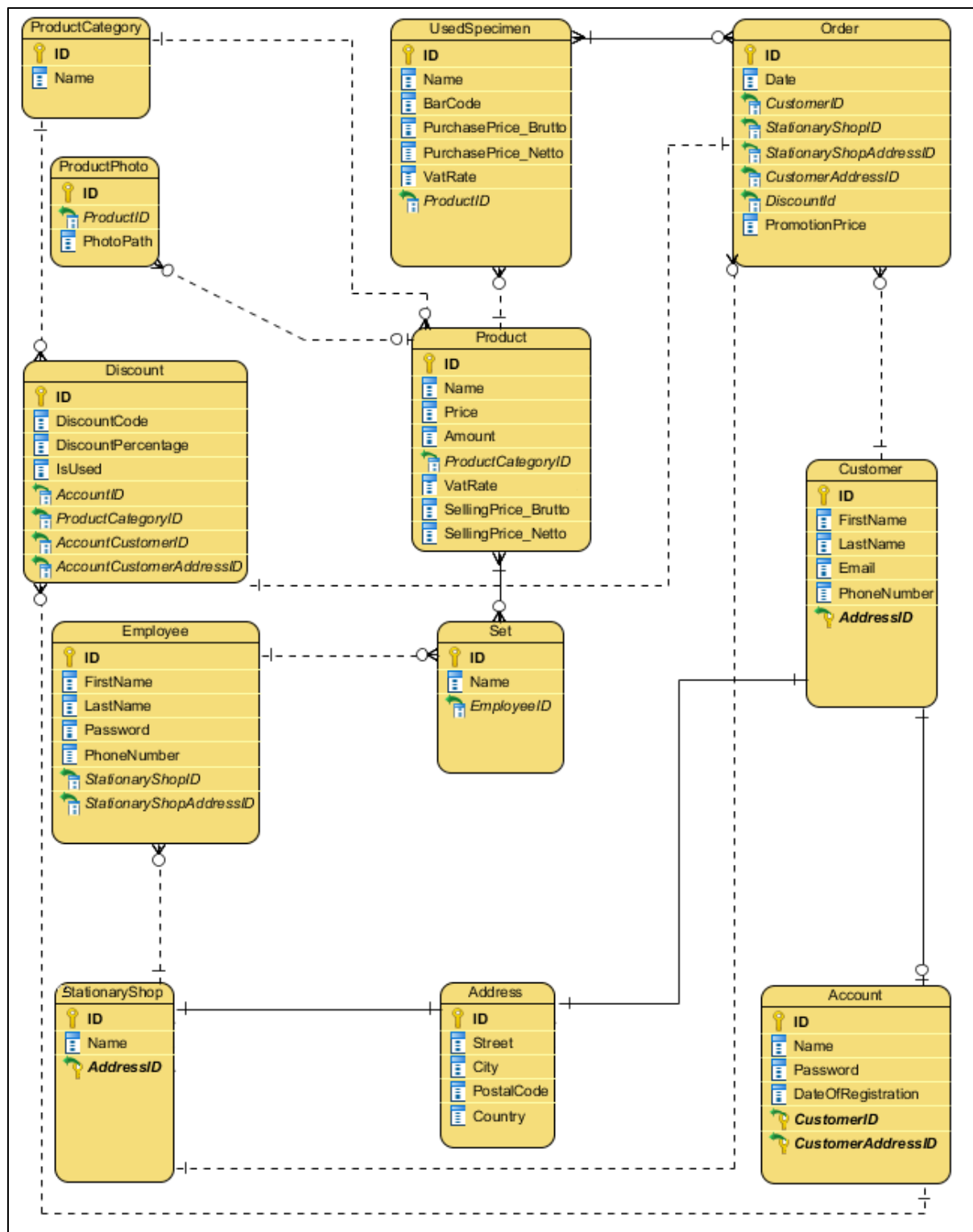
Konceptualne projektowanie bazy danych to proces konstrukcji modelu danych, który jest niezależny od wszelkich aspektów fizycznych (specyficzny model danych, docelowy SZBD, programy użytkowe, języki programowania, platforma sprzętowa).



Rysunek 1. Model konceptualny.

3.1.2. Model logiczny

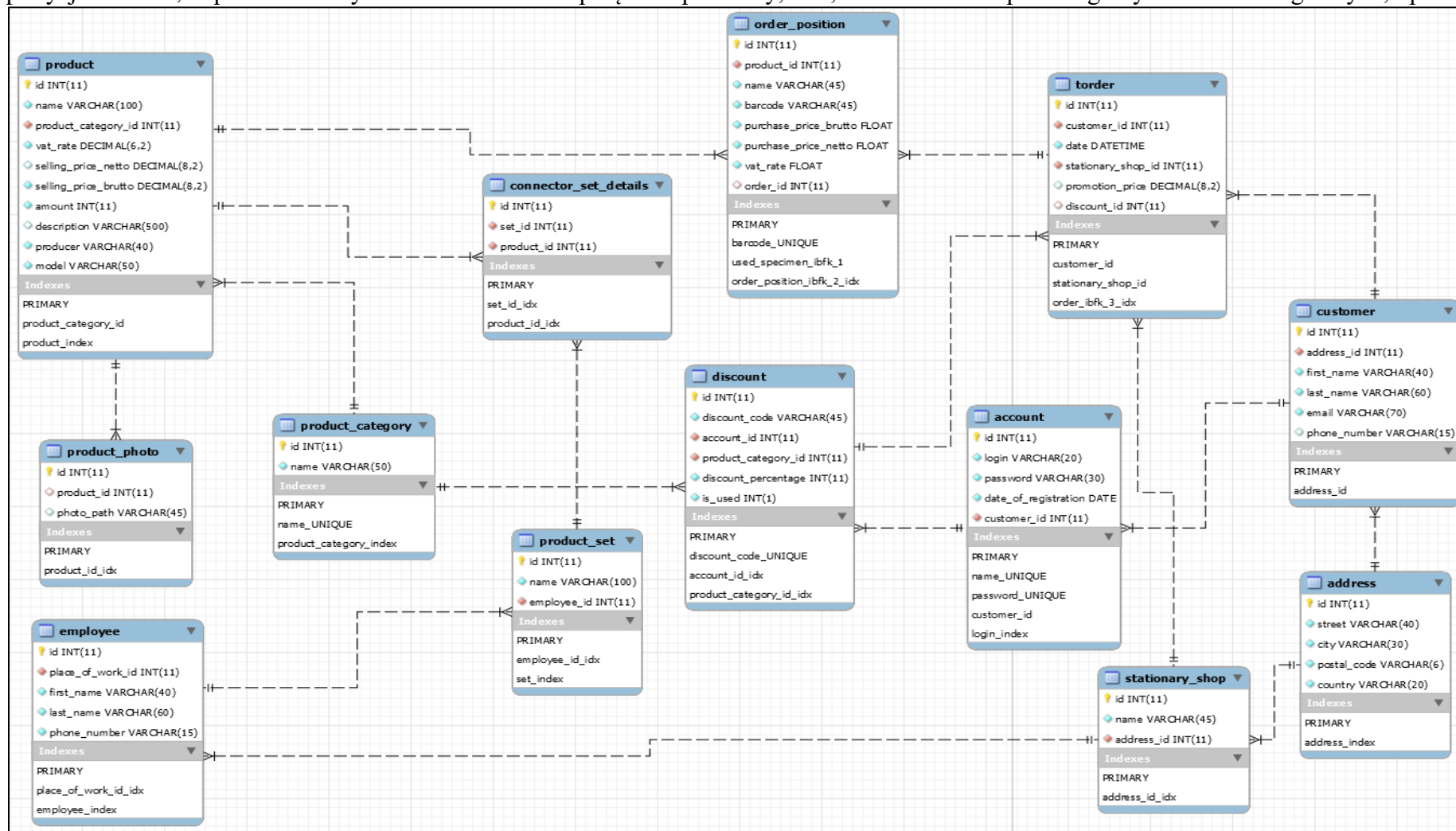
Logiczne projektowanie bazy danych to proces konstrukcji, który jest oparty na specyficznym modelu danych, np. relacyjnym lub obiektowym, ale niezależny od konkretnego SZBD i innych aspektów fizycznych. Opis na poziomie logicznym przypomina typowy projekt bazy danych wykonany np. w języku UML.



Rysunek 2. Model logiczny.

3.1.3. Model fizyczny i ograniczenia integralności danych

Model fizyczny to opis parametrów mających na celu optymalizację działania hurtowni danych, takich jak indeksowanie, partycjonowanie, kopiowanie danych oraz elementów: sprzęt komputerowy, sieć, rozmieszczenie poszczególnych zasobów logicznych, itp.



Rysunek 3. Model fizyczny.

3.1.4. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych





Dostęp do bazy danych jest możliwy bezpośrednio z niej samej lub z poziomu aplikacji. W pierwszym przypadku zabezpieczenie polega na podaniu hasła do bazy, a następnie loginu i hasła użytkownika. W drugiej sytuacji wymagane są tylko dane logowania użytkownika lub hasło panelu pracowniczego. Aby ochronić bazę przed podejmowaniem niepożądanych działań, takich jak dodawanie danych do tabeli, czy ich usuwanie bądź modyfikowanie, użyto triggerów, czyli tzw. wyzwalaczy. W celu dodatkowego zabezpieczenia regularnie wykonywana będzie kopia zapasowa przy użyciu MySQL Workbench.

3.1.5. Inne elementy schematu – mechanizmy przetwarzania danych






3.1.5.a. Widoki

Na potrzeby projektu tworzone będą widoki na etapie realizacji aplikacji. Niezbędne widoki to m.in.:







- widok wyświetlający ogólnikowe dane o zestawach produktów:

view_set_details	
	product_set - name
	product - SUM_OF_product_price_brutto
	employee - first_name
	employee - last_name

- widok wyświetlający produkty wchodzące w skład zestawów produktów:

view_products_of_set	
	product_set - name
	product_category - name
	product - name
	product - selling_price_brutto
	product - amount







- widok wyświetlający ogólnikowe dane o produktach:

view_info_about_products	
	product_category - name
	product - name
	product - selling_price_brutto
	product - selling_price_netto
	product - amount
	product_set - IFNULL_name









- widok wyświetlający dane o klientach:

view_info_about_customers	
	customer - first_name
	customer - last_name
	customer - email
	customer - IFNULL_phone_number
	account - IFNULL_login
	address - street
	address - city
	address - postal_code
	address - country







- widok wyświetlający dane o pracownikach i ich miejscu pracy:

view_info_about_employees_and_their_place_of_work	
	employee - first_name
	employee - last_name
	address - street
	address - postal_code
	address - city
	address - country

- widok wyświetlający najważniejsze szczegóły zamówienia:

view_order_details	
	order - date
	stationary_shop - name
	used_specimen - SUM_OF_purchase_price_brutto
	discount - IFNULL_discount_code
	order - promotion_price
	customer - id
	customer - first_name
	customer - last_name

- widok wyświetlający egzemplarze produktów zamówienia:

view_used_specimens_of_order	
	order - date
	used_specimen - name
	used_specimen - barcode
	used_specimen - purchase_price_netto
	used_specimen - purchase_price_brutto
	used_specimen - vat_rate

3.1.5.b. Wyzwalacze

Triggery będą podstawowym elementem zabezpieczającym bazę danych. Podobnie jak w przypadku widoków większość triggerów zostanie stworzona na etapie realizacji aplikacji.

Podstawowe wyzwalacze, które zostaną utworzone:

- Tabela **product**:
BEFORE INSERT: Wyliczenie kwoty netto produktu na podstawie wartości podatku oraz kwoty brutto. Ilość dostępnych sztuk dodawanego produktu musi być nie mniejsza od zera.

BEFORE UPDATE: Wyliczenie kwoty netto produktu na podstawie wartości podatku oraz kwoty brutto. Ilość dostępnych sztuk aktualizowanego produktu musi być nie mniejsza od zera.

BEFORE DELETE: Produkt może zostać usunięty tylko wtedy, gdy jego ilość wynosi 0.
- Tabela **used_specimen**:
BEFORE INSERT: Przepisanie wartości odpowiadającego produktu: ceny netto, ceny brutto, nazwy, wartości podatku oraz wygenerowanie 6 znakowego unikalnego znaku towarowego. Brak możliwości utworzenia egzemplarza produktu w przypadku ilości produktu równej 0. Dekrementacja wartości ilości produktów w odpowiadającym produkcie.

AFTER UPDATE: Inkrementacja wartości ilości produktów w odpowiadającym produkcie.
- Tabela **connector_order_details**:
AFTER INSERT: Sprawdzenie wartości rabatu (jeżeli podano kod rabatowy). Obliczenie wartości zniżki. Zmniejszenie ceny wybranych produktów (jeżeli podano kod rabatowy) zamówienia oraz aktualizacja ceny promocyjnej zamówienia.

AFTER UPDATE: Sprawdzenie wartości rabatu (jeżeli podano kod rabatowy). Obliczenie wartości zniżki. Zmniejszenie ceny wybranych produktów (jeżeli podano kod rabatowy) zamówienia oraz aktualizacja ceny promocyjnej zamówienia.
- Tabela **order**:
BEFORE INSERT: Brak możliwości wstawienia zamówienia z datą inną niż aktualna.

3.1.5.c. Indeksy

Na tabelach `order` (zamówienia) oraz `used_specimen` (wykorzystane egzemplarze produktów) nie będą tworzone indeksy, ze względu na często wykonywane operacje typu: `INSERT`, `UPDATE` oraz `DELETE`. W przypadku nałożenia indeksów na takie tabele czas wykonywania zapytań może wcale nie przyspieszyć, a znacznie się wydłużyć.

Indeksy zostaną utworzone na kolumnach tabel, na których są wykonywane częste zapytania filtrujące, dokonujące złączeń z innymi tabelami, a dane wynikowe są dodatkowo sortowane.

- Tabela **product_set**:

Indeks utworzony na kolumnie `name`. Tabela `product_set` będzie się łączyć z tabelą `product`, dane zestawów będą filtrowane (klauszula `WHERE`) i sortowane (klauszula `ORDER BY`).

- Tabela **account**:

Indeks utworzony na kolumnach `login` oraz `password`. Użytkownik podczas logowania będzie musiał podać login i hasło, a w celu weryfikacji poprawności danych logowania będzie wykonywane zapytanie wyszukiujące użytkownika o podanym hasle i loginie w bazie danych.

- Tabela **employees**:

Indeks na kolumnach `first_name`, `last_name`, `phone_number`. Wyświetlane dane o pracownikach ograniczają się do imienia, nazwiska i numeru telefonu komórkowego

- Tabela **product**:

Ze względu na to, że produkt jest najważniejszą tabelą w programie, a zapytania filtrujące, zbiór produktów będą wykonywane cały czas postawiono nałożyć indeks na kolumny `name`, `selling_price_brutto`, `product_category`.

- Tabela **product_category**:

Indeks nałożony na kolumnie `name`, ponieważ tabela będzie często łączona z innym tabelami i używana w zapytaniach filtrujących a zmiany w niej będą dokonywane bardzo rzadko.

- Tabela **address**:

Indeks utworzony na kolumnach `street`, `city`, `postal_code`, `country`. W przypadku pobierania danych z tej tabeli zawsze wyciągane są informacje ze wszystkich kolumn.

3.1.5.d. Ograniczenia na poprawność danych

W bazie danych zostanie wymuszone ograniczenie wartości dopuszczalnych dla tabeli Discount (Rabat). Rabat po utworzeniu przyjmie wartość domyślną równą 10. W przypadku zwiększenia rabatu wartości dopuszczalne to 20, 30, 40 oraz 50.

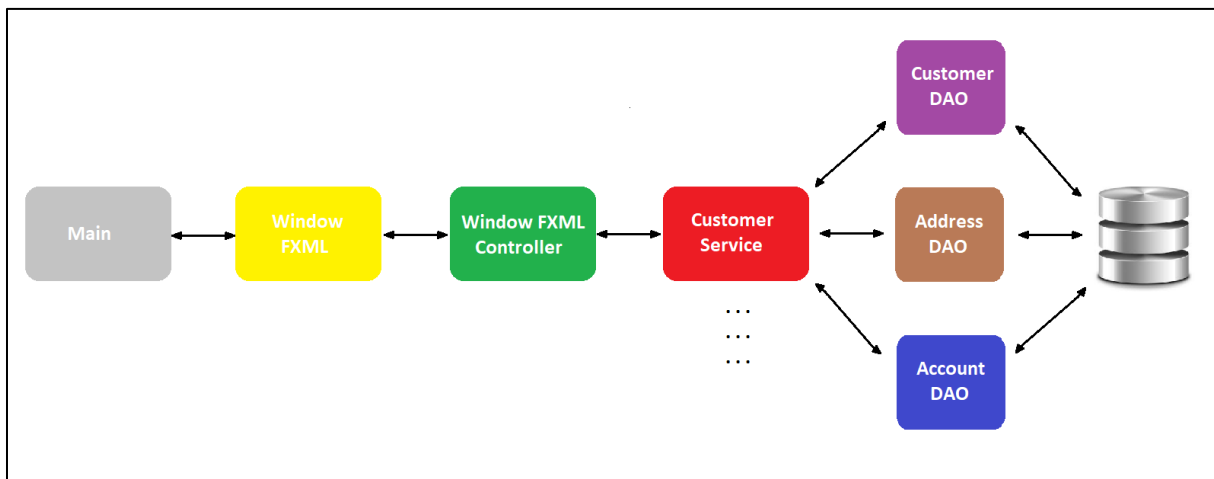
3.2. Projekt aplikacji użytkownika

3.2.1. Architektura aplikacji i diagramy projektowe

Zrezygnowano z tworzenia diagramów stanów, ponieważ klasy (encje) to odzwierciedlenie tabel bazy danych z instrukcjami mapowania. Metody łączenia z bazą danych oraz wykonywane na niej operacje zaimplementowane będą przy użyciu klas DAO. Jest to odseparowana warstwa aplikacji, w której przeważającą ilość stanowią metody typu CRUD.

3.2.1.a. Architektura aplikacji wg wzorca Data Access Object

Implementacja aplikacji odbędzie się zgodnie ze wzorcem DAO służącym do odseparowania API niskiego poziomu (dostęp oraz operacje na bazie danych) od usług biznesowych wysokiego poziomu.

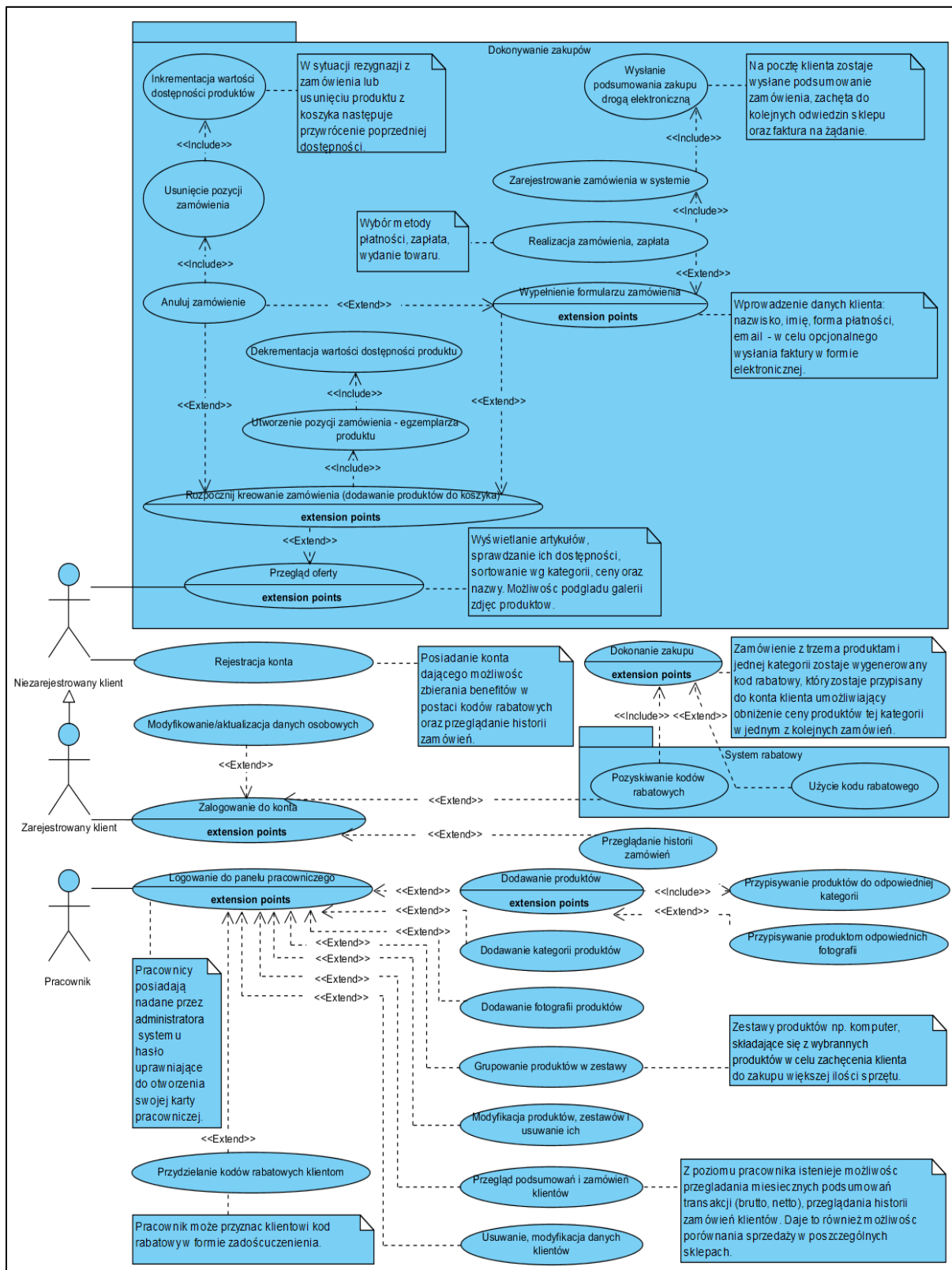


Rysunek 4. Model struktury plików wg wzorca DAO w zastosowaniu w aplikacji z interfejsem graficznym JavaFX.

Implementacja metod typu CRUD (tworzenie, odczytywanie, modyfikowanie, usuwanie elementów BD) będzie miała miejsce w klasach typu DAO. Kontrolery paneli interfejsu graficznego będą zawierały metody implementujące komponenty okien oraz wszelkich zdarzeń z nimi związanych.

3.2.1.b. Diagram przypadków użycia

Poniższy diagram jest graficzną reprezentacją wymagań funkcjonalnych. Definiuje zachowanie systemu bez informowania o wewnętrznej strukturze i narzucania sposobu implementacji. Pozwala na zdefiniowanie przyszłego, spodziewanego zachowania systemu.



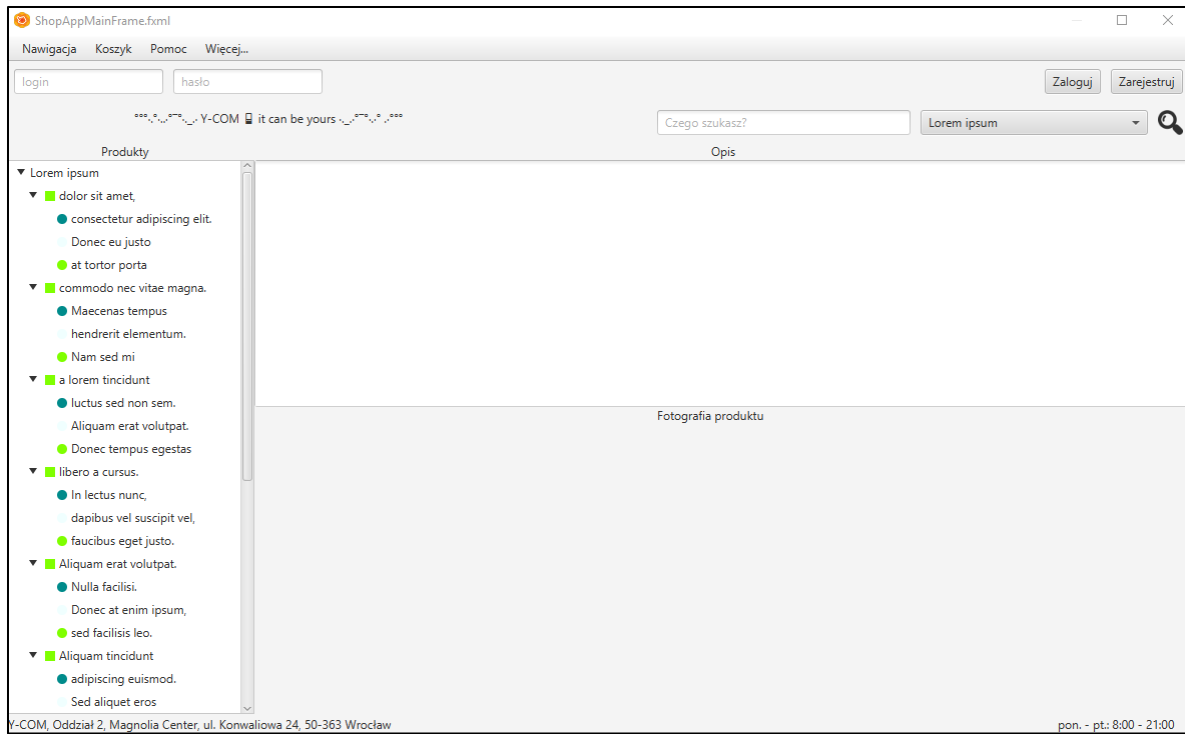
Rysunek 5. Diagram przypadków użycia.

3.2.2. Interfejs graficzny i struktura menu

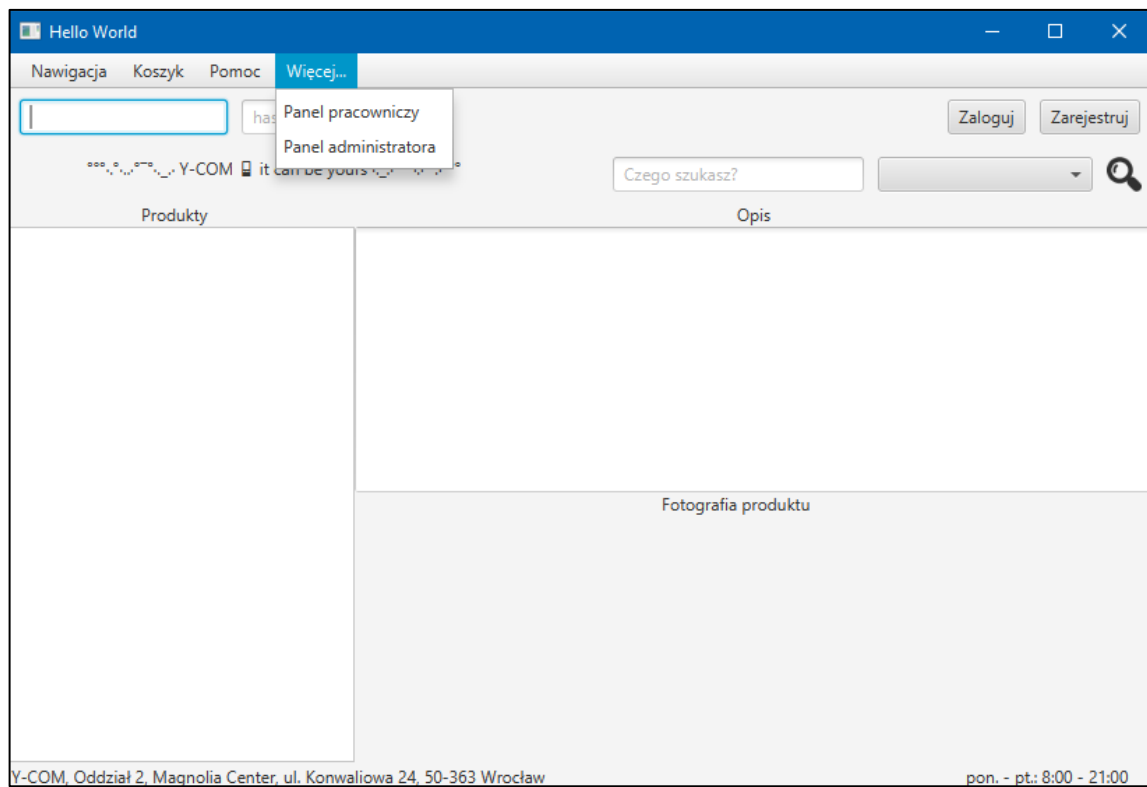
Interfejs graficzny zostanie utworzony przy użyciu biblioteki definiowania GUI - Java FX. Definiowanie widoków aplikacji będzie się odbywało na poziomie języka XML. Prosta edycja cech wspólnych komponentów danego widoku pozwala na łatwą edycję wyglądu komponentów przy pomocy stylów CSS.

Stworzono prototypy niektórych widoków w celu zaprezentowania idei wyglądu interfejsu graficznego oraz struktury aplikacji.

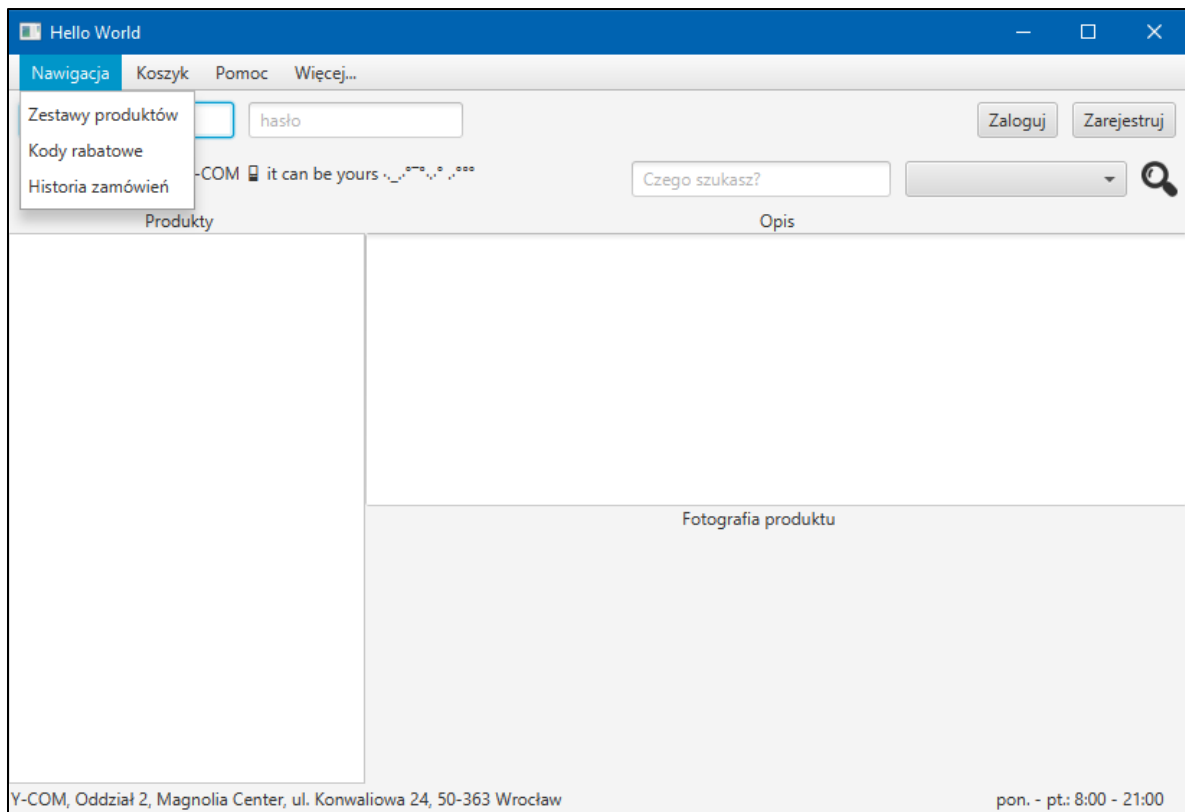
Rysunek 6. Widok aplikacji prezentujący panel tworzenia konta klienta.



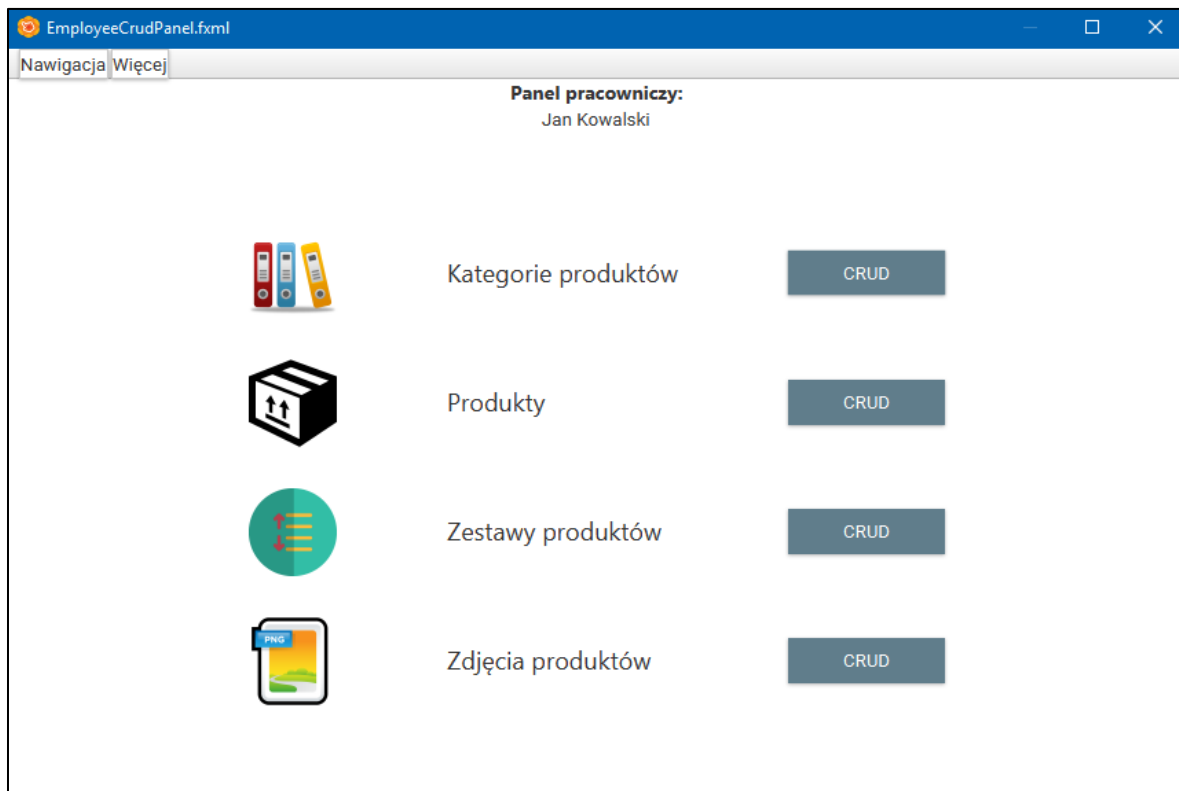
Rysunek 7. Widok aplikacji prezentujący główny panel sklepu - standardowe przeszukiwanie oferty sklepu.



Rysunek 8. Widok aplikacji prezentujący pasek menu (Więcej) głównego panelu sklepu.



Rysunek 9. Widok aplikacji prezentujący pasek menu (Nawigacja) głównego panelu sklepu.



Rysunek 10. Widok aplikacji prezentujący panel pracowniczy.

3.2.3. Projekt wybranych funkcji systemu

3.2.3.a. Interfejs metod CRUD dla obiektów bazy danych

```
public interface EntityCRUD<T extends Object> {  
  
    List<T> getEntities();  
  
    void saveEntity(T entity);  
  
    T getEntity(int id);  
  
    void deleteEntity(int id);  
  
}
```

Kod programu 1. CRUD encji bazy danych - podstawowe metody związane z operacjami na bazie danych.

3.2.3.b. Przykład implementacji kilku metod interfejsu CRUD dla klasy Customer

```
public class CustomerDAOImpl extends DbSessionFactory implements CustomerDAO {  
  
    @Override  
    public List<Customer> getEntities() {  
        List<Customer> customers = null;  
        try (Session currentSession = factory.getCurrentSession()) {  
            currentSession.beginTransaction();  
            Query<Customer> theQuery = currentSession.createQuery("from Customer order by  
                lastName", Customer.class);  
            customers = theQuery.getResultList();  
            currentSession.getTransaction().commit();  
        } catch (Exception exc) {  
            exc.printStackTrace();  
        }  
        return customers;  
    }  
  
    @Override  
    public void saveEntity(Customer entity) {  
        try (Session currentSession = factory.getCurrentSession()) {  
            currentSession.beginTransaction();  
            currentSession.save(entity);  
            currentSession.getTransaction().commit();  
        } catch (Exception exc) {  
            exc.printStackTrace();  
        }  
    }  
  
    @Override  
    public void deleteEntity(int id) {  
        try (Session currentSession = factory.getCurrentSession()) {  
            currentSession.beginTransaction();  
            currentSession.createQuery("delete from Customer where id=:customerId")  
                .setParameter("customerId", id).executeUpdate();  
            currentSession.getTransaction().commit();  
        } catch (Exception exc) {  
            exc.printStackTrace();  
        }  
    }  
  
}
```

Kod programu 2. Implementacja kilku podstawowych metod związanych z operacjami na bazie danych dla klasy Customer.

3.2.3.c. Metoda uwierzytelniania podczas logowania dla zarejestrowanego klienta

```
@Override
public Customer authenticate(String login, String password){
    Account userAccount = this.findByLogin(login);
    if(userAccount == null)
        return null;
    else
        if(password.equals(userAccount.getPassword()))
            return userAccount.getCustomer();
        else
            return null;
}
```

Kod programu 3. Implementacja metody uwierzytelniania podczas logowania dla zarejestrowanego klienta.

W przypadku pomyślnego zalogowania zmieniają się właściwości dostępu do bazy danych, ponieważ użytkownik BD dla zarejestrowanego klienta posiada więcej przywilejów.

3.2.4. Metoda podłączania do bazy danych - integracja z bazą danych

Połączenie z bazą danych odbędzie się za pomocą framework'a warstwy dostępu do bazy danych o nazwie Hibernate. Stanowi on warstwę abstrakcji pośredniczącą w komunikacji aplikacji z bazą danych obsługując trwałość obiektów aplikacji.

Parametry połączenia z bazą danych ustanawiane są w pliku konfiguracyjnym XML o nazwie hibernate.cfg.xml.

Jednym z najważniejszych elementów powstałych w wyniku połączenia aplikacji z bazą danych jest obiekt sessionFactory służący do tworzenia obiektów Session, poprzez które realizowana jest komunikacja z bazą danych. Najczęściej na całą aplikację występuje tylko jeden obiekt sessionFactory tworzony w momencie jej inicjalizacji. Odstępstwa od tych przypadków następują w momencie aplikacji, która łączy się z kilkoma bazami danych.


```

<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Ustawienia parametrów JDBC w celu połączenia z BD-->

        <!-- Wybór sterownika do obsługi bazy danych MySql -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
            <!-- Connection.url wskazujący na bazę danych -->
            name="connection.url">jdbc:mysql://localhost:3306/computer_shop?useSSL=false&
            useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&am
            p;</property>

        <!-- Login i hasło aktualnie uprzywilejowanego do obsługi BD konta -->
        <property name="connection.username">computer_shop_registered_client</property>
        <property name="connection.password">cli3nt</property>

        <!-- Pula połączeń - aktualna wartość jedynie w fazie implementacji -->
        <property name="connection.pool_size">1</property>

        <!-- Wybór obsługiwanego przez Hibernate dialektu. W tym przypadku - MySql -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Wyświetlanie działania Hibernate na ekran -->
        <property name="show_sql">true</property>

    </session-factory>

</hibernate-configuration>

```

Kod programu 4. Plik konfiguracyjny połączenia z bazą danych.

Wartości pliku konfiguracyjnego można bezproblemowo zmieniać z poziomu kodu Javy podczas wykonywania aplikacji. Zmiana typów użytkowników w zależności od rodzaju uprzywilejowania będzie się odbywać bez przeszkód.

3.2.5. Projekt zabezpieczeń na poziomie aplikacji

3.2.5.a. Wyrażenie regularne opisujące łańcuch znaków hasła klienckiego

System na poziomie aplikacji nie będzie dopuszczał tworzenia kont z hasłem nieposiadającym przynajmniej jednej dużej litery, jednej cyfry oraz jednego znaku specjalnego.

3.2.5.b. Kontrola poprawności danych dla tworzonych oraz modyfikowanych rekordów

Przed dodaniem nowego rekordu dane zostaną sprawdzone pod kątem zgodności typów, sprawdzania dużej litery na początku imienia/nazwiska, odpowiedniej konstrukcji adresu e-mail, odpowiedniej ilości liczb w numerze telefonu, odpowiedniej konstrukcji kodu pocztowego itp. Ponadto nastąpi sprawdzenie czy we wprowadzonym rekordzie nie występują puste znaki (na początku lub na końcu) i ewentualne usunięcie ich.

3.2.5.c. Zapytania potwierdzające w przypadku usuwania rekordów

Może nastąpić sytuacja, że użytkownik bazy danych usunie sporą część danych w wyniku niefortunnego zdarzenia. Aby temu zapobiec, w szczególności w przypadku usuwania kaskadowego, należy wprowadzić mechanizm potwierdzania operacji usuwania.

4. Implementacja systemu baz danych

4.1. Tworzenie tabel i definiowanie ograniczeń

4.1.1. Tabela konta użytkownika - account

```
CREATE TABLE `account` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `login` VARCHAR(40) NOT NULL,  
  `password` VARCHAR(20) NOT NULL,  
  `date_of_registration` DATETIME NOT NULL,  
  `is_confirmed` TINYINT(1) NOT NULL,  
  `customer_id` INT(11) NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `name_UNIQUE` (`login`),  
  UNIQUE KEY `password_UNIQUE` (`password`),  
  KEY `customer_id` (`customer_id`),  
  KEY `login_index` (`login`, `password`),  
  CONSTRAINT `account_ibfk_1` FOREIGN KEY (`customer_id`)  
    REFERENCES `customer` (`id`)  
) ENGINE=INNODB DEFAULT CHARSET=UTF8
```

Kod programu 5. Kod SQL odpowiadający za utworzenie tabeli account.

4.1.2. Tabela adresu zamieszkania klienta oraz lokalizacji sklepu - address

```
CREATE TABLE `address` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `street` VARCHAR(40) NOT NULL,  
  `city` VARCHAR(30) NOT NULL,  
  `postal_code` VARCHAR(6) NOT NULL,  
  `country` VARCHAR(20) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `address_index` (`street`, `city`, `postal_code`, `country`)  
) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=UTF8
```

Kod programu 2. Kod SQL odpowiadający za utworzenie tabeli address.

4.1.3. Tabela danych klienta sklepu - customer

```
CREATE TABLE `customer` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `address_id` INT(11) NOT NULL,  
  `first_name` VARCHAR(40) NOT NULL,  
  `last_name` VARCHAR(60) NOT NULL,  
  `email` VARCHAR(70) NOT NULL,  
  `phone_number` VARCHAR(15) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `phone_number_UNIQUE` (`phone_number`),  
  KEY `address_id` (`address_id`),  
  CONSTRAINT `customer_ibfk_1` FOREIGN KEY (`address_id`)  
    REFERENCES `address` (`id`)  
) ENGINE=INNODB DEFAULT CHARSET=UTF8
```

Kod programu 6. Kod SQL odpowiadający za utworzenie tabeli customer.

4.1.4. Tabela kodów rabatowych użytkowników - discount

```
CREATE TABLE `discount` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `discount_code` VARCHAR(45) NOT NULL,  
  `account_id` INT(11) NOT NULL,  
  `product_category_id` INT(11) NOT NULL,  
  `discount_percentage` INT(11) NOT NULL DEFAULT '10',  
  `is_used` INT(11) NOT NULL DEFAULT '0',  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `discount_code_UNIQUE` (`discount_code`),  
  KEY `account_id_idx` (`account_id`),  
  KEY `product_category_id_idx` (`product_category_id`),  
  CONSTRAINT `account_id` FOREIGN KEY (`account_id`)  
    REFERENCES `account` (`id`)  
    ON DELETE NO ACTION ON UPDATE NO ACTION,  
  CONSTRAINT `discount_ibfk_1` FOREIGN KEY (`product_category_id`)  
    REFERENCES `product_category` (`id`)  
) ENGINE=INNODB DEFAULT CHARSET=UTF8
```

Kod programu 7. Kod SQL odpowiadający za utworzenie tabeli discount.

4.1.5. Tabela danych pracowników sklepu stacjonarnego - employee

```
CREATE TABLE `employee` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `place_of_work_id` INT(11) NOT NULL,  
  `first_name` VARCHAR(40) NOT NULL,  
  `last_name` VARCHAR(60) NOT NULL,  
  `phone_number` VARCHAR(15) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `place_of_work_id_idx` (`place_of_work_id`),  
  KEY `employee_index` (`first_name`, `last_name`, `phone_number`),  
  CONSTRAINT `place_of_work_id` FOREIGN KEY (`place_of_work_id`)  
    REFERENCES `stationary_shop` (`id`)  
    ON DELETE NO ACTION ON UPDATE CASCADE  
) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=UTF8
```

Kod programu 8. Kod SQL odpowiadający za utworzenie tabeli employee.

4.1.6. Tabela danych zamówień - order

```
CREATE TABLE `order` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `customer_id` INT(11) NOT NULL,
  `date` DATETIME NOT NULL,
  `stationary_shop_id` INT(11) NOT NULL,
  `promotion_price` INT(11) NOT NULL DEFAULT '0',
  `discount_id` INT(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `customer_id` (`customer_id`),
  KEY `stationary_shop_id` (`stationary_shop_id`),
  KEY `order_ibfk_3_idx` (`discount_id`),
  CONSTRAINT `order_ibfk_1` FOREIGN KEY (`customer_id`)
    REFERENCES `customer` (`id`),
  CONSTRAINT `order_ibfk_2` FOREIGN KEY (`stationary_shop_id`)
    REFERENCES `stationary_shop` (`id`),
  CONSTRAINT `order_ibfk_3` FOREIGN KEY (`discount_id`)
    REFERENCES `discount` (`id`)
  ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=INNODB DEFAULT CHARSET=UTF8
```

Kod programu 9. Kod SQL odpowiadający za utworzenie tabeli order.

4.1.7. Tabela produktów - product

```
CREATE TABLE `product` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(30) NOT NULL,
  `product_category_id` INT(11) NOT NULL,
  `vat_rate` DECIMAL(6 , 2 ) NOT NULL,
  `selling_price_netto` DECIMAL(8 , 2 ) DEFAULT NULL,
  `selling_price_brutto` DECIMAL(8 , 2 ) NOT NULL,
  `amount` INT(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `product_category_id` (`product_category_id`),
  KEY `product_index` (`name` , `selling_price_brutto` , `product_category_id`),
  CONSTRAINT `product_category_id` FOREIGN KEY (`id`)
    REFERENCES `product_category` (`id`)
  ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `product_ibfk_1` FOREIGN KEY (`product_category_id`)
    REFERENCES `product_category` (`id`)
) ENGINE=INNODB AUTO_INCREMENT=8 DEFAULT CHARSET=UTF8
```

Kod programu 10. Kod SQL odpowiadający za utworzenie tabeli product.

4.1.8. Tabela kategorii produktów - product_category

```
CREATE TABLE `product_category` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`),
  KEY `product_category_index` (`name`)
) ENGINE=INNODB AUTO_INCREMENT=10 DEFAULT CHARSET=UTF8
```

Kod programu 11. Kod SQL odpowiadający za utworzenie tabeli product_category.

4.1.9. Tabela zdjęć produktów - product_photo

```
CREATE TABLE `product_photo` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `product_id` INT(11) DEFAULT NULL,  
  `photo_path` VARCHAR(45) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `product_id_idx` (`product_id`),  
  CONSTRAINT `product_photo_ibfk_1` FOREIGN KEY (`product_id`)  
    REFERENCES `product` (`id`)  
) ENGINE=INNODB DEFAULT CHARSET=UTF8
```

Kod programu 12. Kod SQL odpowiadający za utworzenie tabeli product_photo.

4.1.10. Tabela zestawów produktów - product_set

```
CREATE TABLE `product_set` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(20) NOT NULL,  
  `employee_id` INT(11) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `employee_id_idx` (`employee_id`),  
  KEY `set_index` (`name`),  
  CONSTRAINT `employee_id` FOREIGN KEY (`employee_id`)  
    REFERENCES `employee` (`id`)  
    ON DELETE NO ACTION ON UPDATE CASCADE  
) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=UTF8
```

Kod programu 13. Kod SQL odpowiadający za utworzenie tabeli product_set.

4.1.11. Tabela egzemplarzy produktów - used_specimen

```
CREATE TABLE `used_specimen` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `product_id` INT(11) DEFAULT NULL,  
  `name` VARCHAR(45) NOT NULL,  
  `barcode` VARCHAR(45) NOT NULL,  
  `purchase_price_brutto` FLOAT NOT NULL,  
  `purchase_price_netto` FLOAT NOT NULL,  
  `vat_rate` FLOAT NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `barcode_UNIQUE` (`barcode`),  
  KEY `used_specimen_ibfk_1` (`product_id`),  
  CONSTRAINT `used_specimen_ibfk_1` FOREIGN KEY (`product_id`)  
    REFERENCES `product` (`id`)  
) ENGINE=INNODB AUTO_INCREMENT=17 DEFAULT CHARSET=UTF8
```

Kod programu 14. Kod SQL odpowiadający za utworzenie tabeli used_specimen.

4.1.12. Tabela danych o sklepach stacjonarnych - stationary_shop

```
CREATE TABLE `stationary_shop` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(45) NOT NULL,  
  `address_id` INT(11) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `address_id_idx` (`address_id`),  
  CONSTRAINT `address_id` FOREIGN KEY (`address_id`)  
    REFERENCES `address` (`id`)  
    ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=UTF8
```

Kod programu 15. Kod SQL odpowiadający za utworzenie tabeli stationary_shop.

4.1.13. Tabela pomocnicza dot. szczegółów zamówień - connector_order_details

```
CREATE TABLE `connector_order_details` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `order_id` INT(11) NOT NULL,  
  `used_specimen_id` INT(11) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `order_id_idx` (`order_id`),  
  KEY `used_specimen_id_idx` (`used_specimen_id`),  
  CONSTRAINT `order_id` FOREIGN KEY (`order_id`)  
    REFERENCES `order` (`id`)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
  CONSTRAINT `used_specimen_id` FOREIGN KEY (`used_specimen_id`)  
    REFERENCES `used_specimen` (`id`)  
    ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=INNODB DEFAULT CHARSET=UTF8
```

Kod programu 16. Kod SQL odpowiadający za utworzenie tabeli connector_order_details.

4.1.14. Tabela pomocnicza dot. szczegółów zestawów - connector_set_details

```
CREATE TABLE `connector_set_details` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `set_id` INT(11) NOT NULL,  
  `product_id` INT(11) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `set_id_idx` (`set_id`),  
  KEY `product_id_idx` (`product_id`),  
  CONSTRAINT `product_id` FOREIGN KEY (`product_id`)  
    REFERENCES `product` (`id`)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
  CONSTRAINT `set_id` FOREIGN KEY (`set_id`)  
    REFERENCES `product_set` (`id`)  
    ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=INNODB DEFAULT CHARSET=UTF8
```

Kod programu 17. Kod SQL odpowiadający za utworzenie tabeli connector_set_details.

4.2. Implementacja mechanizmów przetwarzania danych

4.2.1. Wyzwalacze tabeli produktów - product

4.2.1.a. BEFORE INSERT oraz BEFORE UPDATE

Mechanizm wyliczenia kwoty netto produktu na podstawie wartości podatku oraz kwoty brutto. Sprawdzanie poprawności wartości odpowiadającej za ilość dostępnych sztuk dodawanego produktu. Przy wstawianiu oraz aktualizowaniu produktu wartość ta nie może być mniejsza od zera.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `computer_shop`.`product_BEFORE_INSERT` BEFORE INSERT
ON `product` FOR EACH ROW
BEGIN

DECLARE selling_price decimal(8,2);
SET @selling_price=(100-new.vat_rate)/100;

set new.selling_price_netto = @selling_price*new.selling_price_brutto;

IF(new.amount<0) then
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'ilość produktów nie może być mniejsza od zera';
end if;
```

Kod programu 18. Kod SQL odpowiadający za utworzenie wyzwalaczy BEFORE INSERT oraz BEFORE UPDATE dla tabeli product.

4.2.1.b. BEFORE DELETE

Sprawdzanie poprawności wartości odpowiadającej za ilość dostępnych sztuk dodawanego produktu. Przy usuwaniu produktu wartość ta nie może być mniejsza od zera.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `computer_shop`.`product_BEFORE_DELETE` BEFORE DELETE
ON `product` FOR EACH ROW
BEGIN

IF(old.amount>0) then
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Produkt istnieje jeszcze w bazie danych.';
end if;

END
```

Kod programu 19. Kod SQL odpowiadający za utworzenie wyzwalacza BEFORE DELETE dla tabeli product.

4.2.2. Wyzwalacze tabeli egzemplarzy produktów - used_specimen

4.2.2.a. BEFORE INSERT

Mechanizm przepisania wartości odpowiadającego produktu: ceny netto, ceny brutto, nazwy, wartości podatku oraz wygenerowanie 6 znakowego unikalnego znaku towarowego. Sprawdzanie warunków utworzenia egzemplarza produktu. W przypadku ilości produktu równej 0 utworzenie odpowiadającego mu egzemplarza jest niemożliwe. Mechanizm dekrementacji wartości ilości produktów w odpowiadającym produkcie w przypadku utworzenia egzemplarza.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `computer_shop`.`used_specimen_BEFORE_INSERT` BEFORE
INSERT ON `used_specimen` FOR EACH ROW
BEGIN

declare namee varchar(45);
declare purchase_price_netto decimal(8,2);
declare purchase_price_brutto decimal(8,2);
declare vat_rate integer;
declare amount integer;
declare ready int default 0;
declare rnd_str text;

SELECT
    @namee:=p.name,
    @purchase_price_netto:=p.selling_price_netto,
    @purchase_price_brutto:=p.selling_price_brutto,
    @vat_rate:=p.vat_rate
INTO namee , purchase_price_netto , purchase_price_brutto , vat_rate FROM
product p
WHERE
    new.product_id = p.id;

set new.name= @namee;
set new.purchase_price_netto= @purchase_price_netto;
set new.purchase_price_brutto= @purchase_price_brutto;
set new.vat_rate= @vat_rate;

SELECT
    @amount:=p.amount
INTO amount FROM
product p
WHERE
    p.id = new.id;

    IF(@amount<=0) then
        SIGNAL SQLSTATE '45001'
        SET MESSAGE_TEXT = 'Produktu nie ma na stanie';
    end if;

if new.barcode is null then
    while not ready do
        set rnd_str := lpad(conv(floor(rand()*pow(36,6)), 10, 36), 6, 0);
        if not exists (select * from used_specimen where barcode = rnd_str) then
            set new.barcode = rnd_str;
            set ready := 1;
        end if;
    end while;
end if;

END
```

Kod programu 20. Kod SQL odpowiadający za utworzenie wyzwalacza BEFORE INSERT dla tabeli used_specimen.

4.2.2.b. AFTER INSERT oraz AFTER DELETE

Mechanizm inkrementacji wartości ilości produktów w odpowiadającym produkcie w przypadku usunięcia egzemplarza oraz dekrementacji w przypadku utworzenia.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `computer_shop`.`used_specimen_AFTER_UPDATE` AFTER
UPDATE ON `used_specimen` FOR EACH ROW
BEGIN

update product p
set amount = amount - 1 -- amount + 1 dla AFTER DELETE
where p.id = new.product_id; -- old.product_id; dla AFTER DELETE

END
```

Kod programu 21. Kod SQL odpowiadający za utworzenie wyzwalacza AFTER INS/DEL E dla tabeli used_specimen.

4.2.3. Wyzwalacze tabeli pomocniczej dot. szczegółów zamówień - connector_order_details**4.2.3.a. AFTER INSERT oraz AFTER UPDATE**

Mechanizm obliczania wartości zniżki produktów zamówienia w sytuacji skorzystania z kodu rabatowego dla danej kategorii produktów.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `computer_shop`.`connector_order_details_AFTER_INSERT`
AFTER INSERT ON `connector_order_details` FOR EACH ROW
BEGIN
DECLARE selling_price decimal(8,2);      DECLARE selling_price2 decimal(8,2);
DECLARE selling_percent decimal(8,2);    DECLARE amount integer;
DECLARE percent integer;                 DECLARE discount_percent integer;
DECLARE used tinyint;                    DECLARE product_cat tinyint;
DECLARE discount_id integer;              DECLARE product_id integer;

SELECT @selling_price:=used.purchase_price_brutto , @amount:= count(*) , @discount_percent:=
d.discount_percentage, @used:= d.is_used, @discount_id:= d.product_category_id, @product_id:=
p.product_category_id
into selling_price, amount, discount_percent, used, discount_id, product_id
FROM computer_shop.order o
LEFT JOIN connector_order_details con ON (new.order_id = o.id)
LEFT JOIN used_specimen used ON (used.id = new.used_specimen_id)
left join product p on (used.product_id =p.id)
LEFT JOIN discount d on (d.id = o.discount_id)
where new.id = con.id;

SET @selling_percent=(100-@discount_percent)/100;
if (@used = 0) then
    if(@product_id = @discount_id) then
        update computer_shop.order o
        set o.promotion_price = o.promotion_price + @selling_percent * @selling_price
        where o.id = new.order_id;
    else
        update computer_shop.order o
        set o.promotion_price = o.promotion_price + @selling_price
        where o.id = new.order_id;
    end if; else
    update computer_shop.order o
    set o.promotion_price = o.promotion_price + @selling_price
    where o.id = new.order_id;
end if;
END
```

Kod programu 22. Kod SQL odp. za utworzenie wyzw. AFTER INSERT/UPDATE dla tabeli connector_order_details.

4.2.4. Wyzwalacze tabeli zamówień - order

4.2.4.a. BEFORE INSERT

Sprawdzanie możliwości wstawienia zamówienia. Nie ma możliwości wstawienia nowej pozycji zamówienia z datą inną niż aktualna.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `computer_shop`.`order_BEFORE_INSERT` BEFORE INSERT ON
`order` FOR EACH ROW
BEGIN

IF(new.date!=curdate()) then
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Data zamówienia jest błędna.';
end if;
```

Kod programu 23. Kod SQL odpowiadający za utworzenie wyzwalacza BEFORE INSERT dla tabeli order.

4.2.5. Widok wyświetlające dane ogólne o zestawach produktów

Widok tworzy zbiór danych złożony z nazwy zestawu, łącznej sumy kosztu produktów zestawu oraz imienia i nazwiska opiekuna zestawu.

```
CREATE VIEW `view_set_details` AS
SELECT
    `s`.`name` set_name,
    SUM(`p`.`selling_price_brutto`) total_price,
    CONCAT(CONCAT(`e`.`first_name`, ' '),
            `e`.`last_name`) set_patron
FROM
    (((`product_set` `s`
    LEFT JOIN `connector_set_details` `d_s` ON ((`s`.`id` = `d_s`.`set_id`)))
    LEFT JOIN `product` `p` ON ((`d_s`.`product_id` = `p`.`id`)))
    LEFT JOIN `employee` `e` ON ((`e`.`id` = `s`.`employee_id`)))
GROUP BY `s`.`name`, `e`.`first_name`, `e`.`last_name`
```

Kod programu 24. Kod SQL odpowiadający za utworzenie widoku view_set_details.

4.2.6. Widok wyświetlający produkty wchodzące w skład zestawów produktów

Widok tworzy zbiór danych złożony z nazwy zestawu, kategorii produktu, nazwy produktu, ceny brutto produktu oraz ilości dostępnych produktów tego samego typu.

```
CREATE VIEW `view_products_of_set` AS
SELECT
    `s`.`name` set_name,
    `c`.`name` product_category,
    `p`.`name` product_name,
    `p`.`selling_price_brutto` price,
    `p`.`amount` amount
FROM
    (((`product` `p`
    LEFT JOIN `product_category` `c` ON ((`c`.`id` = `p`.`product_category_id`)))
    LEFT JOIN `connector_set_details` `d_s` ON ((`d_s`.`product_id` = `p`.`id`)))
    LEFT JOIN `product_set` `s` ON ((`s`.`id` = `d_s`.`set_id`)))
WHERE
    (`s`.`name` IS NOT NULL)
```

Kod programu 25. Kod SQL odpowiadający za utworzenie widoku view_products_of_set.

4.2.7. Widok wyświetlający dane ogólne o dostępnych w sklepie produktach

Widok tworzy zbiór danych złożony z kategorii produktu, nazwy produktu, ceny netto, ceny brutto, ilości dostępnych produktów tego samego typu oraz nazwy zestawu, do którego przynależy (wyświetlenie „-”, jeżeli nie należy do żadnego zestawu).

```
CREATE VIEW `view_info_about_products` AS
SELECT
  `c`.`name` product_category,
  `p`.`name` product_name,
  `p`.`selling_price_netto` price_netto,
  `p`.`selling_price_brutto` price_brutto,
  `p`.`amount` amount,
  IFNULL(`s`.`name`, '-') belongs_to_set
FROM
  (((`product` `p`
  LEFT JOIN `product_category` `c` ON ((`c`.`id` = `p`.`product_category_id`)))
  LEFT JOIN `connector_set_details` `d_s` ON ((`d_s`.`product_id` = `p`.`id`)))
  LEFT JOIN `product_set` `s` ON ((`s`.`id` = `d_s`.`set_id`)))
```

Kod programu 26. Kod SQL odpowiadający za utworzenie widoku view_info_about_products.

4.2.8. Widok wyświetlający dane o klientach

Widok tworzy zbiór danych złożony z imienia klienta, nazwiska klienta, emaila klienta, numeru telefonu (znak „-”, jeżeli nie podano), loginu konta (znak „-”, jeżeli nie posiada), adresu w postaci ulicy, miasta, kodu pocztowego oraz kraju.

```
CREATE VIEW `view_info_about_customers` AS
SELECT
  `c`.`first_name` first_name,
  `c`.`last_name` last_name,
  `c`.`email` email,
  IFNULL(`c`.`phone_number`, '-') phone_number,
  IFNULL(`ac`.`login`, '-') account_login,
  `a`.`street` street,
  `a`.`city` city,
  `a`.`postal_code` postal_code,
  `a`.`country` country
FROM
  ((`customer` `c`
  LEFT JOIN `address` `a` ON ((`a`.`id` = `c`.`address_id`)))
  LEFT JOIN `account` `ac` ON ((`ac`.`customer_id` = `c`.`id`)))
```

Kod programu 27. Kod SQL odpowiadający za utworzenie widoku view_info_about_customers.

4.2.9. Widok wyświetlający informacje o pracownikach oraz ich miejscu pracy

Widok tworzy zbiór danych złożony z imienia pracownika, jego nazwiska, adresu miejsca pracy w postaci ulicy, kodu pocztowego, miasta oraz kraju.

```
CREATE VIEW `view_info_about_employees_and_their_place_of_work` AS
SELECT
  `e`.`first_name` first_name,
  `e`.`last_name` last_name,
  CONCAT(CONCAT(CONCAT(CONCAT(CONCAT(CONCAT(CONCAT(`a`.`street`, ' '),
                                                    `a`.`postal_code`),
                                                    ' '),
                                                    `a`.`city`),
                                                    ' '),
                                                    `a`.`country`) workplace_address
FROM
  ((`employee` `e`
  LEFT JOIN `stationary_shop` `s` ON ((`s`.`id` = `e`.`place_of_work_id`)))
  LEFT JOIN `address` `a` ON ((`a`.`id` = `s`.`address_id`)))
```

*Kod programu 28. Kod SQL odpowiadający za utworzenie widoku
view_info_about_employees_and_their_place_of_work.*

4.2.10. Widok wyświetlający najważniejsze szczegóły zamówienia

Widok tworzy zbiór danych złożony z daty zamówienia, nazwy sklepu stacjonarnego, standardowej kwoty będącej sumą cen egzemplarzy produktów wchodzących w skład zamówienia, kodu rabatowego (jeżeli został wprowadzony, znak „-”, jeżeli nie podano), ceny promocyjnej, numeru id klienta, jego imienia oraz nazwiska.

```
CREATE VIEW `view_order_details` AS
SELECT
  `o`.`date` AS `date`,
  `sh`.`name` AS `stationary_shop_name`,
  SUM(`op`.`purchase_price_brutto`) AS `standard_order_price`,
  IFNULL(`d`.`discount_code`, '-') AS `discount_code`,
  `o`.`promotion_price` AS `promotion_order_price`,
  `c`.`id` AS `customer_id`,
  `c`.`first_name` AS `customer_first_name`,
  `c`.`last_name` AS `customer_last_name`
FROM
  (((`torder` `o`
  LEFT JOIN `order_position` `op` ON ((`op`.`order_id` = `o`.`id`)))
  LEFT JOIN `stationary_shop` `sh` ON ((`sh`.`id` = `o`.`stationary_shop_id`)))
  LEFT JOIN `customer` `c` ON ((`c`.`id` = `o`.`customer_id`)))
  LEFT JOIN `discount` `d` ON ((`d`.`id` = `o`.`discount_id`)))
GROUP BY `o`.`date`, `sh`.`name`, `c`.`first_name`, `c`.`last_name`, `c`.`id`,
  `d`.`discount_code`, `o`.`promotion_price`
```

Kod programu 29. Kod SQL odpowiadający za utworzenie widoku view_order_details.

4.2.11. Widok wyświetlający egzemplarze produktów zamówienia

Widok tworzący zbiór danych złożony z daty zamówienia, nazwy egzemplarza produktu, ceny netto, ceny brutto oraz wartości podatku VAT.

```
VIEW `view_order_positions_of_order` AS
SELECT
  `o`.`date` AS `order_date`,
  `op`.`name` AS `name`,
  `op`.`barcode` AS `barcode`,
  `op`.`purchase_price_netto` AS `purchase_price_netto`,
  `op`.`purchase_price_brutto` AS `purchase_price_brutto`,
  `op`.`vat_rate` AS `vat_rate`
FROM
  (`order_position` `op`
  LEFT JOIN `torder` `o` ON ((`o`.`id` = `op`.`order_id`)))
WHERE
  (`o`.`date` IS NOT NULL)
```

Kod programu 30. Kod SQL odpowiadający za utworzenie widoku view_order_positions_of_order.

4.3. Implementacja uprawnień i innych ograniczeń

W celu ograniczenia dostępu do danych do minimum dla każdego z typów osób korzystających ze sklepu komputerowego oraz systemu zarządzania nim, stworzono 4 typy użytkowników oraz nadano im odpowiednie uprawnieniami na poziomie bazy danych:

- niezalogowany klient - możliwość wyszukiwania rekordów (oferta sklepu), wstawiania nowych wierszy w tabelach (pozycje zamówienia, dane osobowe), usuwania wierszy z tabel (rezygnacja z pozycji zamówienia)

```
GRANT SELECT, INSERT, DELETE ON `computer_shop`.* TO 'computer_shop_customer'@'%',
```

- zalogowany klient - możliwość wyszukiwania rekordów (oferta sklepu, kody rabatowe, historia zamówień), wstawiania nowych wierszy w tabelach (pozycje zamówienia, dane osobowe), usuwania wierszy z tabel (rezygnacja z pozycji zamówienia), zmiany wartości zapisanych w tabelach (modyfikowanie danych dostępu do konta):

```
GRANT SELECT, INSERT, UPDATE, DELETE ON `computer_shop`.* TO 'computer_shop_registered_client'@'%',
```

- pracownik - możliwość wyszukiwania rekordów (oferta sklepu, kody rabatowe klientów, historia zamówień, dane o klientach), wstawiania nowych wierszy w tabelach (pozycje zamówienia, dane osobowe, produkty, zestawy produktów), usuwania wierszy z tabel (usuwanie produktów, kodów rabatowych klientów, danych klientów, zestawów produktów), zmiany wartości zapisanych w tabelach (modyfikowanie informacji o ofercie sklepu):

```
GRANT SELECT, INSERT, UPDATE, DELETE, EXECUTE, SHOW VIEW ON `computer_shop`.* TO 'computer_shop_employee'@'%',
```

- administrator - wszystkie uprawnienia powyższych użytkowników oraz uprawnienia administratorów związane z zarządzaniem bazą danych, tworzeniem nowych tabel, usuwaniem ich itp.:

```
GRANT ALL PRIVILEGES ON `computer_shop`.* TO 'computer_shop_admin'@'%';
```

4.4. Testowanie bazy danych dla przykładowych danych

4.4.1. Wypełnienie bazy danych przykładowymi wartościami rekordów

W celu przetestowania poprawności tworzenia nowych obiektów danych oraz działania wyzwalaczy, tabelę bazy danych wypełniono przykładowymi losowymi rekordami.

```
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Habrowa 20/7', 'Wrocław', '52-300', 'Polska');
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Zaporoska 64/60', 'Wrocław', '53-416', 'Polska');
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Stalowa 17/23', 'Wrocław', '51-420', 'Polska');
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Hirszfelda 23/32', 'Wrocław', '51-300', 'Polska');
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Librania 43/100', 'Zamysłowo', '10-100', 'Polska');
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Mirosławowo 1/1', 'Walkowo', '11-200', 'Polska');
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Innosław 32/100', 'Kraków', '12-230', 'Polska');
INSERT INTO `computer_shop`.`address` (`street`, `city`, `postal_code`, `country`) VALUES ('Zmysłowska 20/30', 'Lubań', '32-230', 'Polska');
```

Kod programu 31. Kod SQL odpowiadający za uzupełnienie tabeli adresów przykładowymi rekordami danych.

	id	street	city	postal_code	country
	2	Habrowa 20/7	Wrocław	52-300	Polska
	5	Hirszfelda 23/32	Wrocław	51-300	Polska
	8	Innosław 32/100	Kraków	12-230	Polska
	6	Librania 43/100	Zamysłowo	10-100	Polska
	7	Mirosławowo 1/1	Walkowo	11-200	Polska
	4	Stalowa 17/23	Wrocław	51-420	Polska
	3	Zaporoska 64/60	Wrocław	53-416	Polska
	9	Zmysłowska 20/30	Lubań	32-230	Polska
	NULL	NULL	NULL	NULL	NULL

Rysunek 11. Stan tabeli adresów po wykonaniu operacji wstawiania.

```
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('2', 'Igor', 'Bagrowski', 'bagrowski.ig@gmail.com', '662342322');
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('3', 'Michał', 'Właszczenko', 'lodo@op.pl', '232323232');
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('4', 'Roman', 'Polański', 'polan@gmail.com', '656234843');
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('5', 'Michał', 'Wolan', 'wolan.michal@onet.pl', '534234232');
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('6', 'Radosław', 'Dzik', 'dziku@gmail.com', '298329023');
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('7', 'Marek', 'Zelent', 'kochamcpp@gmail.com', '232654542');
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('8', 'Kamil', 'Cieślak', 'ciesliczek@gmail.com', '989832214');
INSERT INTO `computer_shop`.`customer` (`address_id`, `first_name`, `last_name`, `email`, `phone_number`) VALUES ('9', 'Piotr', 'Szepiatowski', 'szepiat@onet.pl', '235454332');
```

Kod programu 32. Kod SQL odpowiadający za uzupełnienie tabeli klientów przykładowymi rekordami danych.

	id	address_id	first_name	last_name	email	phone_number
	2	2	Ioor	Bacrowski	baacrowski.ia@gmail.com	662342322
	3	3	Michał	Właszczenko	lodo@op.pl	232323232
	4	4	Roman	Polański	oolan@gmail.com	656234843
	5	5	Michał	Wolan	wolan.michal@onet.pl	534234232
	6	6	Radosław	Dzik	dziku@gmail.com	298329023
	7	7	Marek	Zelent	kochamcpp@gmail.com	232654542
	8	8	Kamil	Cieślik	ciesliczek@gmail.com	989832214
	9	9	Piotr	Szepiatowski	szepiat@onet.pl	235454332
	NULL	NULL	NULL	NULL	NULL	NULL

Rysunek 12. Stan tabeli klientów po wykonaniu operacji wstawiania.

```

INSERT INTO `product_category` (`name`) VALUES ('Akcesoria');
INSERT INTO `product_category` (`name`) VALUES ('Foto, TV i kamery');
INSERT INTO `product_category` (`name`) VALUES ('Gaming');
INSERT INTO `product_category` (`name`) VALUES ('Komputery stacjonarne');
INSERT INTO `product_category` (`name`) VALUES ('Laptopy i tablety');
INSERT INTO `product_category` (`name`) VALUES ('Oprogramowanie');
INSERT INTO `product_category` (`name`) VALUES ('Podzespoły komputerowe');
INSERT INTO `product_category` (`name`) VALUES ('Telefony');
INSERT INTO `product_category` (`name`) VALUES ('Urządzenia peryferyjne');

```

Kod programu 33. Kod SQL odpowiadający za uzupełnienie tabeli kategorii produktów przykładowymi rekordami danych.

	id	name
	10	Akcesoria
	11	Foto, TV i kamery
	12	Gaming
	13	Komputery stacjonarne
	14	Laptopy i tablety
	15	Oprogramowanie
	16	Podzespoły komputerowe
	17	Telefony
	18	Urządzenia peryferyjne
	NULL	NULL

Rysunek 13. Stan tabeli kategorii produktów po wykonaniu operacji wstawiania.

```

INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('misiak', '2123', '2017-11-29', '2');
INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('sole', '3534523', '2017-11-29 00:00:00', '3');
INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('kucharzyk', '574523', '2017-11-29 00:00:00', '4');
INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('santon', 'asdaskd23', '2017-11-29 00:00:00', '5');
INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('patrykz13', 'sda34wd', '2017-11-29 00:00:00', '6');
INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('pekiel', 'sads34', '2017-11-29 00:00:00', '7');
INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('hahah', 'hsdsk34', '2017-11-29 00:00:00', '8');
INSERT INTO `computer_shop`.`account` (`login`, `password`, `date_of_registration`, `customer_id`) VALUES ('radar', 'rader34', '2017-11-29 00:00:00', '9');

```

Kod programu 34. Kod SQL odpowiadający za uzupełnienie tabeli kont użytkowników przykładowymi rekordami danych.

	id	login	password	date_of_registration	customer_id
	1	misiak	2123	2017-11-29 00:00:00	2
	2	sole	3534523	2017-11-29 00:00:00	3
	3	kucharzvk	574523	2017-11-29 00:00:00	4
	4	santon	asdaskd23	2017-11-29 00:00:00	5
	5	patrvkz13	sda34wd	2017-11-29 00:00:00	6
	6	pekiel	sads34	2017-11-29 00:00:00	7
	7	hahah	hsdsk34	2017-11-29 00:00:00	8
	8	radar	rader34	2017-11-29 00:00:00	9
	NULL	NULL	NULL	NULL	NULL

Rysunek 14. Stan tabeli kont użytkowników po wykonaniu operacji wstawiania.

```

INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('2312jknjjnj',1,10,10);
INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('123123jnj',2,11,5);
INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('24124ijufg',3,12,23);
INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('89789789',4,13,11);
INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('asda87978',5,14,5);
INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('87897asdsa',6,15,10);
INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('897sdasdas',7,16,15);
INSERT INTO `discount` (`discount_code`,`account_id`,`product_category_id`,`discount_percentage`) VALUES
('3123opkoo',8,17,2);

```

Kod programu 35. Kod SQL odpowiadający za uzupełnienie tabeli rabatów użytkowników przykładowymi rekordami danych.

	id	discount_code	account_id	product_category_id	discount_percentage	is_used
	10	2312ikniini	1	10	10	0
	11	123123inii	2	11	5	0
	12	24124iufu	3	12	23	0
	13	89789789	4	13	11	0
	14	asda87978	5	14	5	0
	15	87897asdsa	6	15	10	0
	16	897sdasdas	7	16	15	0
	17	3123opkoo	8	17	2	0
	NULL	NULL	NULL	NULL	NULL	NULL

Rysunek 15. Stan tabeli rabatów użytkowników po wykonaniu operacji wstawiania.

```

INSERT INTO `computer_shop`.`employee` (`place_of_work_id`,`first_name`,`last_name`,`phone_number`) VALUES
('2','Kamil','Cieřlik','231234252');
INSERT INTO `computer_shop`.`employee` (`place_of_work_id`,`first_name`,`last_name`,`phone_number`) VALUES
('2','Michał','Zuber','232543543');
INSERT INTO `computer_shop`.`employee` (`place_of_work_id`,`first_name`,`last_name`,`phone_number`) VALUES
('3','Radosław','Markowski','530253422');
INSERT INTO `computer_shop`.`employee` (`place_of_work_id`,`first_name`,`last_name`,`phone_number`) VALUES
('3','Piotr','Cieszek','354346423');
INSERT INTO `computer_shop`.`employee` (`place_of_work_id`,`first_name`,`last_name`,`phone_number`) VALUES
('2','Zenon','Marton','565653278');

```

Kod programu 36. Kod SQL odpowiadający za uzupełnienie tabeli pracowników przykładowymi rekordami danych.

	id	place_of_work_id	first_name	last_name	phone_number
	2	2	Kamil	Cieřlik	231234252
	3	2	Michał	Zuber	232543543
	5	2	Kamil	Cieřlik	231234252
	6	2	Michał	Zuber	232543543
	7	3	Radosław	Markowski	530253422
	8	3	Piotr	Cieszek	354346423
	9	2	Zenon	Marton	565653278
	NULL	NULL	NULL	NULL	NULL

Rysunek 16. Stan tabeli pracowników po wykonaniu operacji wstawiania.

```

INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (2,'2017-11-29 00:00:00',2,10);
INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (2,'2017-11-29 00:00:00',3,11);
INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (2,'2017-11-29 00:00:00',3,12);
INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (3,'2017-11-29 00:00:00',3,13);
INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (4,'2017-11-29 00:00:00',2,14);
INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (5,'2017-11-29 00:00:00',2);
INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (5,'2017-11-29 00:00:00',2,15);
INSERT INTO `order` (`customer_id`,`date`,`stationary_shop_id`,`discount_id`) VALUES (6,'2017-11-29 00:00:00',2,16);

```

Kod programu 37. Kod SQL odpowiadający za uzupełnienie tabeli zamówień przykładowymi rekordami danych.

	id	customer_id	date	stationary_shop_id	promotion_price	discount_id
	3	2	2017-11-29 00:00:00	2	0	10
	4	2	2017-11-29 00:00:00	3	0	11
	5	2	2017-11-29 00:00:00	3	0	12
	6	3	2017-11-29 00:00:00	3	0	13
	7	4	2017-11-29 00:00:00	2	0	14
	8	5	2017-11-29 00:00:00	2	0	NULL
	9	5	2017-11-29 00:00:00	2	0	15
	10	6	2017-11-29 00:00:00	2	0	16
	NULL	NULL	NULL	NULL	NULL	NULL

Rysunek 17. Stan tabeli zamówień po wykonaniu operacji wstawiania.

```

INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Lenovo y50-70','10','23','3400','2');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Huawei p9','11','23','1900','3');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Asus computer pack','12','23','5000','10');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Windows 10','13','23','500','15');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Fotel Gracza Hykker','14','23','900','3');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Sony a350 alpha','15','20','1999','20');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Drukarka Canon mp3550','16','21','300','10');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Smycz do kluczy','17','23','10','5');
INSERT INTO `computer_shop`.`product` (`name`,`product_category_id`,`vat_rate`,`selling_price_brutto`,`amount`) VALUES ('Karta graficzna Geforce gtx 1060','18','23','1500','10');

```

Kod programu 38. Kod SQL odpowiadający za uzupełnienie tabeli produktów przykładowymi rekordami danych.

DOKUMENTACJA DO PROJEKTU

id	name	product_category_id	vat_rate	selling_price_netto	selling_price_brutto	amount
10	Lenovo v50-70	10	23.00	2618.00	3400.00	2
11	Huawei p9	11	23.00	1463.00	1900.00	3
12	Asus computer pack	12	23.00	3850.00	5000.00	10
13	Windows 10	13	23.00	385.00	500.00	15
14	Fotel Gracza Hvkker	14	23.00	693.00	900.00	3
15	Sonv a350 alpha	15	20.00	1599.20	1999.00	20
16	Drukarka Canon mp3550	16	21.00	237.00	300.00	10
17	Smvcz do kluczy	17	23.00	7.70	10.00	5
18	Karta graficzna Geforce gtx 1060	18	23.00	1155.00	1500.00	10
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Rysunek 18. Stan tabeli produktów po wykonaniu operacji wstawiania.

```

INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('10');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('11');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('12');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('13');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('14');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('15');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('16');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('17');
INSERT INTO `computer_shop`.`used_specimen` (`product_id`) VALUES ('18');

```

Kod programu 39. Kod SQL odpowiadający za uzupełnienie tabeli egzemplarzy produktów przykładowymi rekordami danych.

id	product_id	name	barcode	purchase_price_brutto	purchase_price_netto	vat_rate
17	10	Lenovo v50-70	KTY790	3400	2618	23
18	11	Huawei p9	DF0V7S	1900	1463	23
19	12	Asus computer pack	4L9ODP	5000	3850	23
20	13	Windows 10	IPA2B0	500	385	23
21	14	Fotel Gracza Hvkker	7OL4FS	900	693	23
22	15	Sonv a350 alpha	IL3FBT	1999	1599.2	20
23	16	Drukarka Canon mp3550	XPPRDK	300	237	21
24	17	Smvcz do kluczy	4TAIY9	10	7.7	23
25	18	Karta graficzna Geforce gtx 1060	UXBCOH	1500	1155	23
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Rysunek 19. Stan tabeli egzemplarzy produktów po wykonaniu operacji wstawiania.

```

INSERT INTO `computer_shop`.`product_set` (`name`, `employee_id`) VALUES ('Mega zestaw', '2');
INSERT INTO `computer_shop`.`product_set` (`name`, `employee_id`) VALUES ('New Zestaw', '3');
INSERT INTO `computer_shop`.`product_set` (`name`, `employee_id`) VALUES ('Ultra komputer', '8');
INSERT INTO `computer_shop`.`product_set` (`name`, `employee_id`) VALUES ('Komputer', '5');
INSERT INTO `computer_shop`.`product_set` (`name`, `employee_id`) VALUES ('Rador', '6');
INSERT INTO `computer_shop`.`product_set` (`name`, `employee_id`) VALUES ('Extra zestaw', '7');

```

Kod programu 40. Kod SQL odpowiadający za uzupełnienie tabeli zestawów produktów przykładowymi rekordami danych.

id	name	employee_id
3	Mega zestaw	2
4	New Zestaw	3
6	Mega zestaw	2
7	New Zestaw	3
8	Ultra komputer	8
9	Komputer	5
10	Rador	6
11	Extra zestaw	7
NULL	NULL	NULL

Rysunek 20. Stan tabeli zestawów produktów po wykonaniu operacji wstawiania.

```
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('3', '10');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('3', '11');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('3', '11');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('4', '12');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('4', '13');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('4', '14');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('7', '15');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('7', '16');
INSERT INTO `computer_shop`.`connector_set_details` (`set_id`, `product_id`) VALUES ('7', '17');
```

Kod programu 41. Kod SQL odpowiadający za uzupełnienie tabeli pomocniczej dotyczącej szczegółów zestawów produktów przykładowymi rekordami danych.

	id	set_id	product_id
	17	3	10
	18	3	11
	19	3	11
	20	4	12
	21	4	13
	22	4	14
	23	7	15
	24	7	16
	25	7	17
	NULL	NULL	NULL

Rysunek 21. Stan tabeli pomocniczej dotyczącej szczegółów zestawów produktów po wykonaniu operacji wstawiania.

```
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('3', '17');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('3', '18');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('4', '18');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('5', '18');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('5', '19');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('6', '20');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('6', '21');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('6', '22');
INSERT INTO `computer_shop`.`connector_order_details` (`order_id`, `used_specimen_id`) VALUES ('7', '23');
```

Kod programu 42. Kod SQL odpowiadający za uzupełnienie tabeli pomocniczej dotyczącej szczegółów zamówień przykładowymi rekordami danych.

	id	order_id	used_specimen_id
	1	3	17
	2	3	18
	3	4	18
	4	5	18
	5	5	19
	6	6	20
	7	6	21
	8	6	22
	9	7	23
	NULL	NULL	NULL

Rysunek 22. Stan tabeli pomocniczej dotyczącej szczegółów zamówień po wykonaniu operacji wstawiania.

4.4.2. Przykłady użycia zaimplementowanych widoków

first_name	last_name	email	phone_number	account_login	street	city	postal_code	country
Igor	Bacowski	bacowski.ig@gmail.com	662342322	miśiak	Habrowa 20/7	Wrocław	52-300	Polska
Michał	Właszczenko	lodo@oo.pl	232323232	sole	Zaporoska 64/60	Wrocław	53-416	Polska
Roman	Polański	polan@gmail.com	656234843	kucharzvk	Stalowa 17/23	Wrocław	51-420	Polska
Michał	Wolan	wolan.michal@onet.pl	534234232	santon	Hirszfelda 23/32	Wrocław	51-300	Polska
Radosław	Dzik	dziku@gmail.com	298329023	patrvkz13	Librania 43/100	Zamvstowo	10-100	Polska
Marek	Zelent	kocharmcp@gmail.com	232654542	pekiel	Mirosławowo 1/1	Walkowo	11-200	Polska
Kamil	Cieślik	ciesliczek@gmail.com	989832214	hahah	Innosław 32/100	Kraków	12-230	Polska
Piotr	Szepiatowski	szepiat@onet.pl	235454332	radar	Zmvsłowska 20/30	Lubań	32-230	Polska

Rysunek 23. Wynik wywołania obiektów widoku `select * from view_info_about_customers;`

set_name	total_price	set_patron
Extra zestaw	NULL	Radosław Markowski
Komputer	NULL	Kamil Cieślik
Mega zestaw	7200.00	Kamil Cieślik
New Zestaw	8709.00	Michał Zuber
Rador	NULL	Michał Zuber
Ultra komputer	NULL	Piotr Cieszek

Rysunek 24. Wynik wywołania obiektów widoku `select * from view_set_details;`

first_name	last_name	workplace_address
Kamil	Cieślik	Galowao 30/20 342-10 Wrocław Polska
Michał	Zuber	Galowao 30/20 342-10 Wrocław Polska
Kamil	Cieślik	Galowao 30/20 342-10 Wrocław Polska
Michał	Zuber	Galowao 30/20 342-10 Wrocław Polska
Radosław	Markowski	Marakuji 15/23 32-343 Wrocław Polska
Piotr	Cieszek	Marakuji 15/23 32-343 Wrocław Polska
Zenon	Marton	Galowao 30/20 342-10 Wrocław Polska

Rysunek 25. Wynik wywołania obiektów widoku `select * from view_info_about_employees_and_their_place_of_work;`

4.4.3. Przykład działania mechanizmów ograniczeń wprowadzanych danych

Po próbie wprowadzenia rekordu tabeli - products z wartością ilości produktów mniejszą od zero następuje reakcja systemu i wyświetlenie błędu (działanie triggera).

Kod wstawienia nowego rekordu produktu z niepoprawnymi danymi:

```
INSERT INTO `computer_shop`.`product` (`name`, `product_category_id`, `vat_rate`,
`selling_price_brutto`, `amount`) VALUES ('Lenovo y700', '10', '23', '3400', '-1');
```

Odpowiedź systemu:

Error Code: 1644. Ilość produktów nie może być mniejsza od zera.0.000 sec

5. Implementacja i testowanie aplikacji

5.1. Instalacja i konfiguracja systemu

Użytkownik nie będzie samodzielnie instalować aplikacji. System zakupowy będzie instalowany i konfigurowany na komputerach pracowników sklepów stacjonarnych, a następnie udostępniony w postaci uruchomionej aplikacji.

Najprostszym sposobem uruchomienia aplikacji z postaci kodu źródłowego jest utworzenie pliku .jar zawierającego zewnętrzne biblioteki. Proces ten może zostać zautomatyzowany po zainstalowaniu dodatkowej biblioteki o nazwie „Fat Jar Eclipse Plug-In”.

5.2. Instrukcja użytkowania aplikacji

Za inicjację aplikacji odpowiada pracownik. Wybiera on sklep, w którym znajduje się kiosk, dla którego rejestrowane będą zakupy.



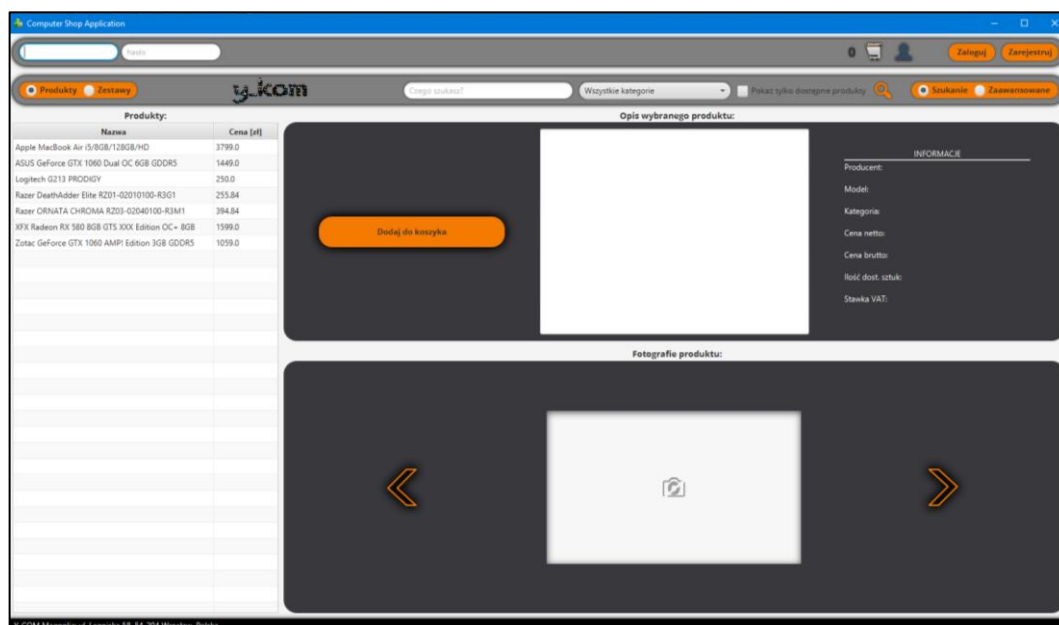
Nazwa	Ulica	Miasto	Kod pocztowy	Kraj
Y-COM Bielany	Czekoladowa 7-9	Bielany Wrocławskie	55-040	Polska
Y-COM Magnolia	Legnicka 58	Wrocław	54-204	Polska

Przeładuj tabelę

Dalej

Rysunek 26. Ekran startowy aplikacji.

Po uruchomieniu aplikacji i wybraniu sklepu ukazuje się główny panel użytkowy interaktywnego kiosku.



Rysunek 27. Panel główny aplikacji.

Poprzez panel główny użytkownik ma dostęp do wszystkich podstawowych funkcji sklepu. Niezarejestrowany klient ma możliwość przeglądania oferty sklepu, tworzenia własnego koszyka produktów, składania zamówień oraz rejestracji. Osoba zarejestrowana ma możliwość zalogowania się do swojego konta.

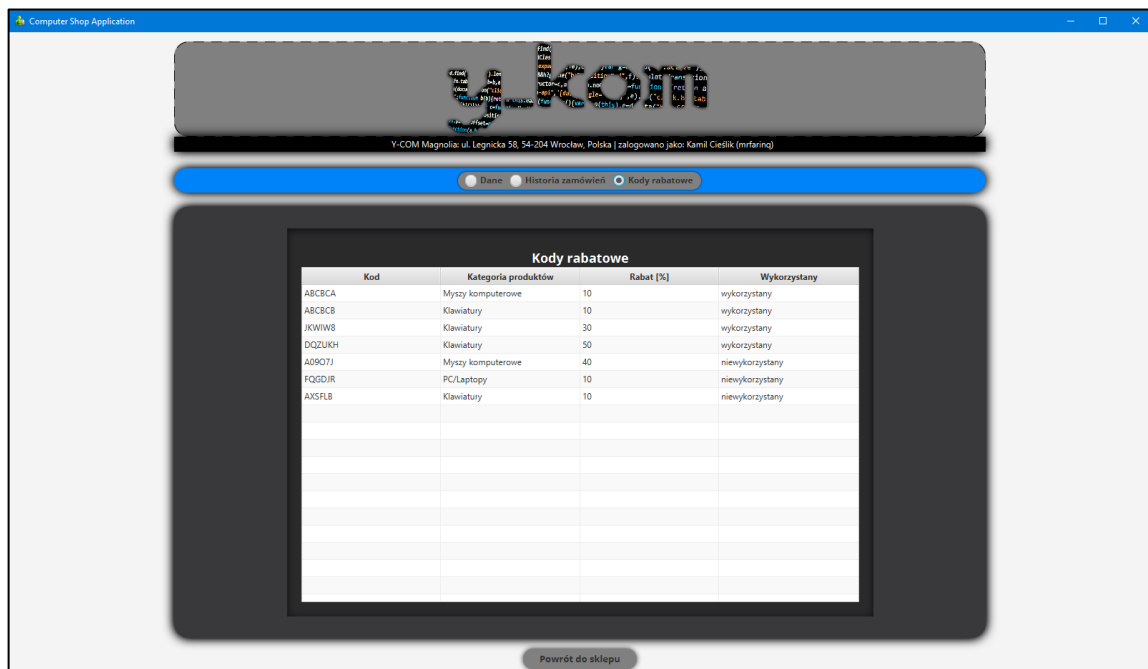
Klienci zalogowani mają dostęp do funkcjonalności dających szereg następujących benefitów:

- modyfikowanie/aktualizacja danych osobowych



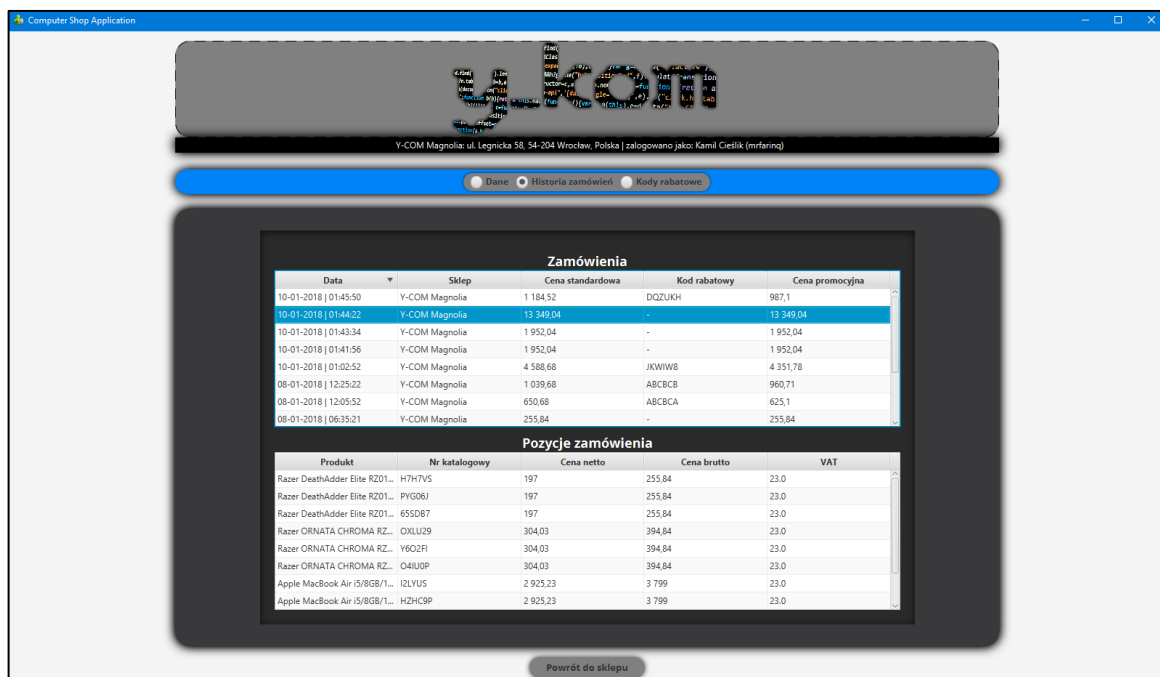
Rysunek 28. Panel modyfikacji danych zalogowanego użytkownika.

- otrzymywanie kodów rabatowych za robienie większej ilości zakupów



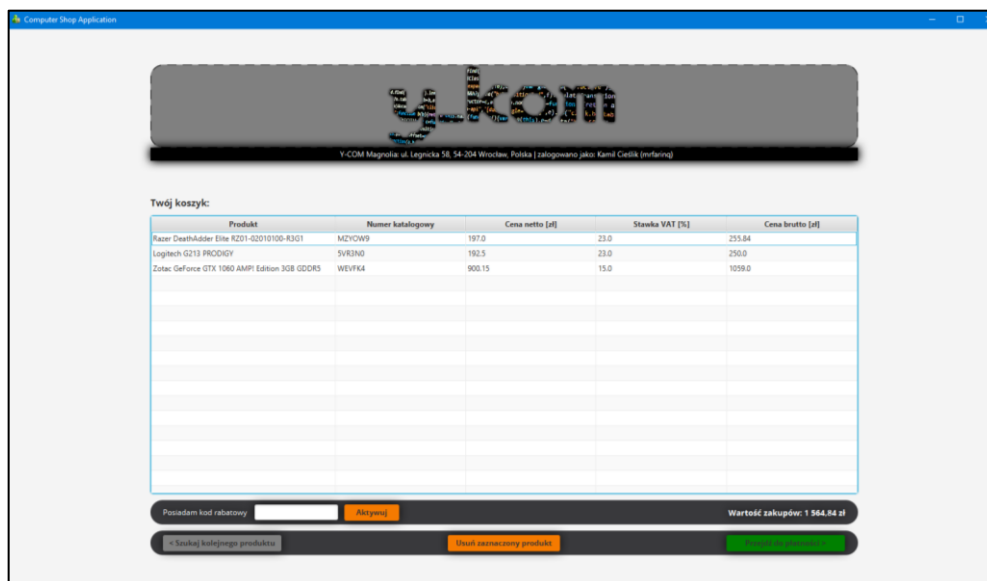
Rysunek 29. Panel przeglądania kodów rabatowych zalogowanego użytkownika.

- wgląd w historię zamówień wraz z produktami zamówień:



Rysunek 30. Panel historii zamówień zalogowanego użytkownika.

- tworzenie własnego koszyka zamówień, w którym można wykorzystać kod rabatowy



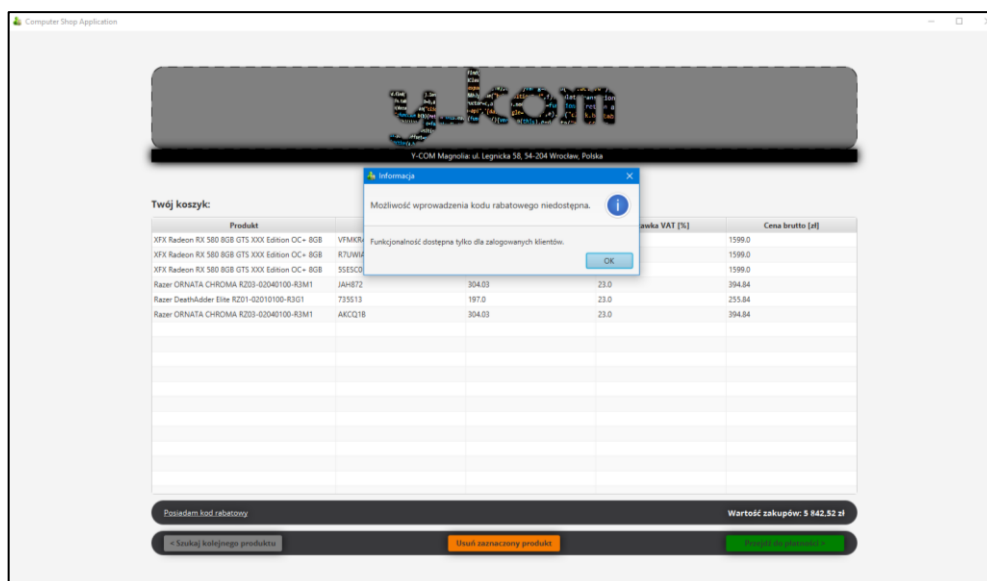
Rysunek 31. Koszyk zakupowy z widoku użytkownika zalogowanego.

5.3. Testowanie wybranych funkcjonalności aplikacji

W aplikacji zostały przeprowadzone testy funkcjonalne działania sklepu. Wszystkie przeprowadzone testy zakończyły się pomyślnie.

5.3.1. Wprowadzenie kodu rabatowego przez niezalogowanego użytkownika

Niezalogowany użytkownik nie ma możliwości korzystania z dodatkowych benefitów systemu zakupowego.



Rysunek 32. Komunikat wywołany po próbie wprowadzenia kodu rabatowego przez niezalogowanego użytkownika.

5.3.2. Złożenie zamówienia

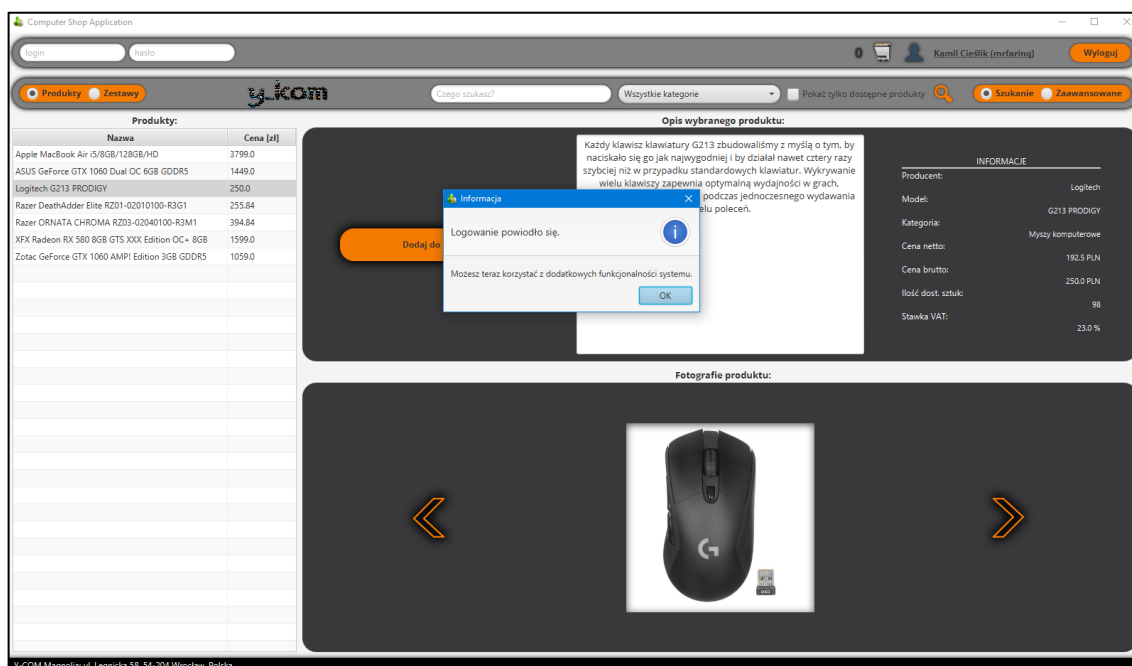
Jeśli wszystkie dane personalne zostały poprawnie wprowadzone po kliknięciu przycisku „Potwierdź zakup” użytkownik może dokonać zakupu w oknie zakupowym.



Rysunek 33. Komunikat finalizacji zamówienia.

5.3.3. Logowanie do panelu użytkownika

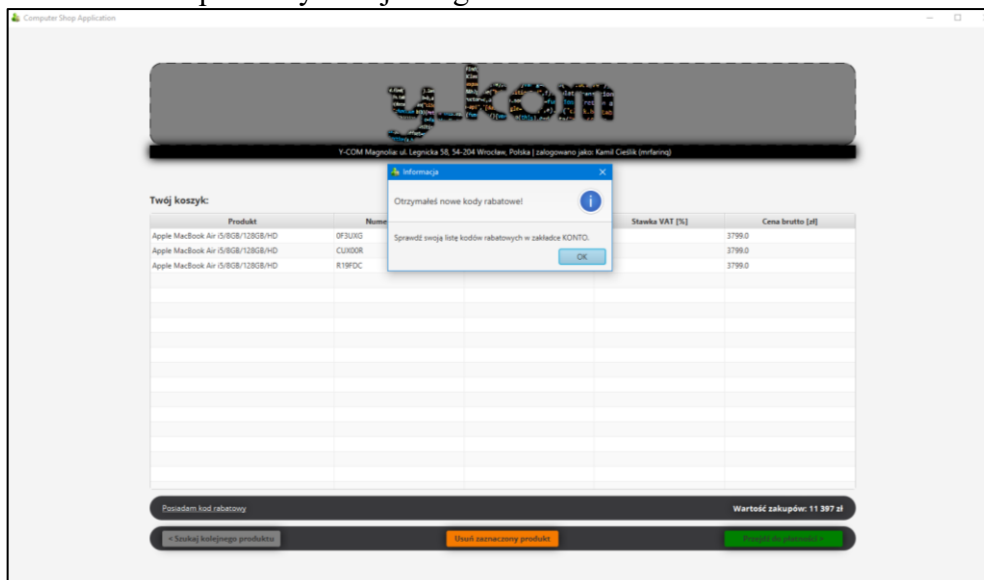
W momencie poprawnego wprowadzenia danych logowania i kliknięciu przycisku „Zaloguj” użytkownik zostaje zalogowany i ma możliwość korzystania z dodatkowych funkcjonalności sklepu – o czym informuje stosowny komunikat.



Rysunek 34. Komunikat pomyślnego zalogowania.

5.3.4. Otrzymywanie kodów rabatowych po odpowiednio dokonanych zakupie

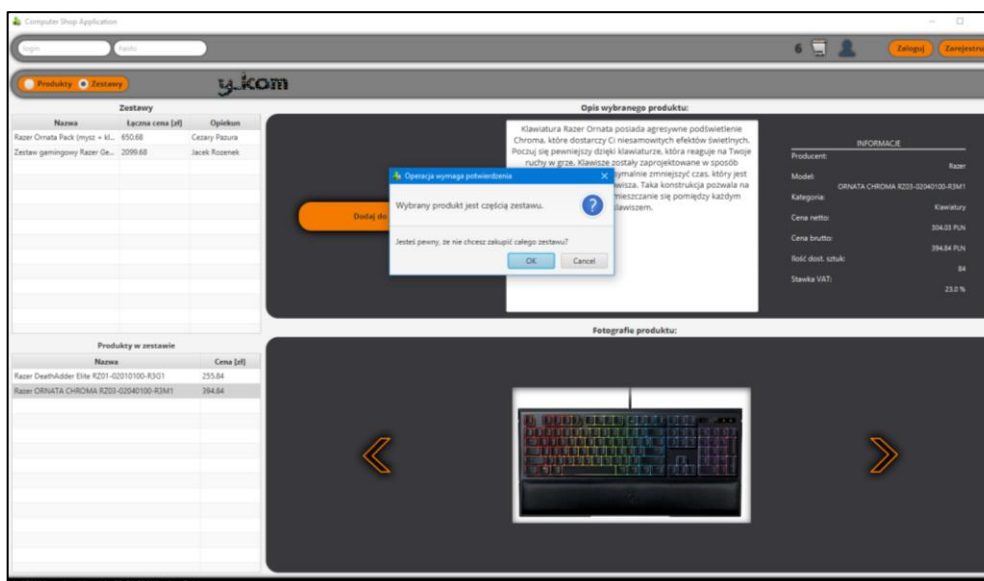
Jeżeli użytkownik w jednym zamówieniu zakupi przynajmniej 3 produkty jednej kategorii to następuje sprawdzanie czy posiada on już niewykorzystany kod rabatowy na tę kategorię produktów. Jeżeli tak to wartość posiadanego już rabatu zwiększa się o 10% (do 50% maksymalnie). Natomiast w sytuacji, kiedy użytkownik nie posiada niewykorzystanego kodu tej kategorii lub posiada kod wykorzystany to dostaje nowy kod rabatowy o wartości równej 10% zniżki na produkty danej kategorii.



Rysunek 35. Komunikat informujący o przyznaniu nowych kodów rabatowych.

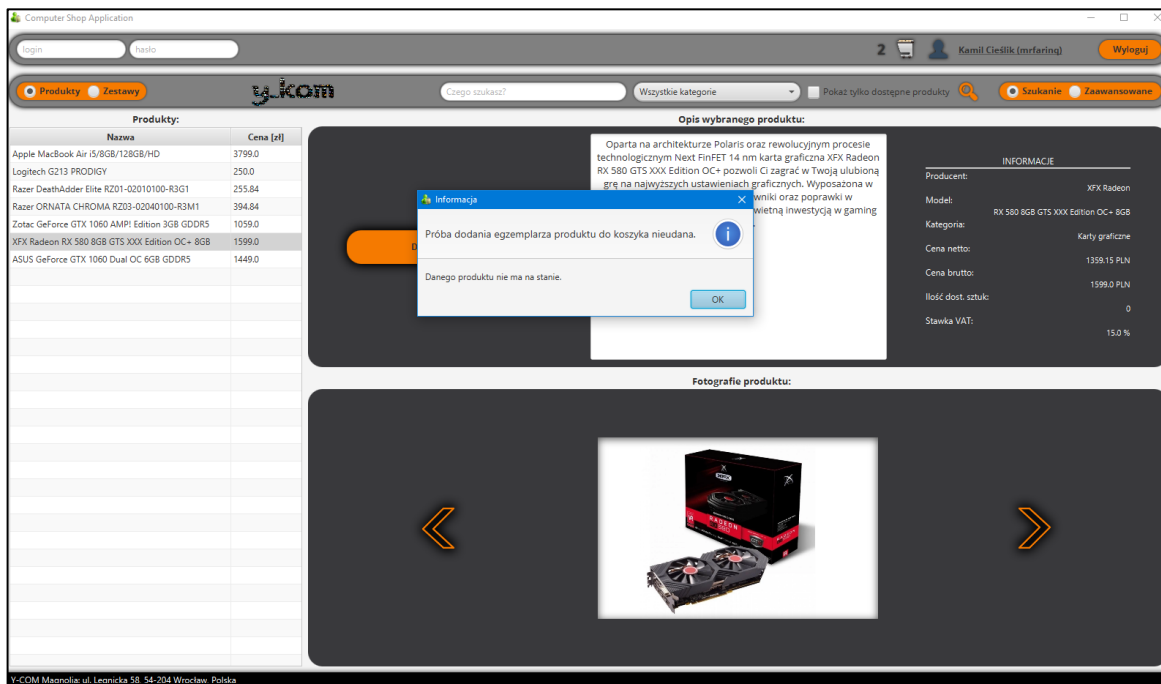
5.3.5. Próba zakupu pojedynczych produktów danego zestawu z widoku zestawów

Użytkownik po wybraniu odszukanego zestawu produktów ma możliwość kupna jego poszczególnych składowych osobno. W tej sytuacji jest informowany o możliwości skorzystania z oferty kupna wszystkich produktów zestawu. Operacja zakupu produktu pojedynczego z widoku zestawów produktów wymaga potwierdzenia.



Rysunek 36. Komunikat informujący o zakupie jednej składowej zestawu.

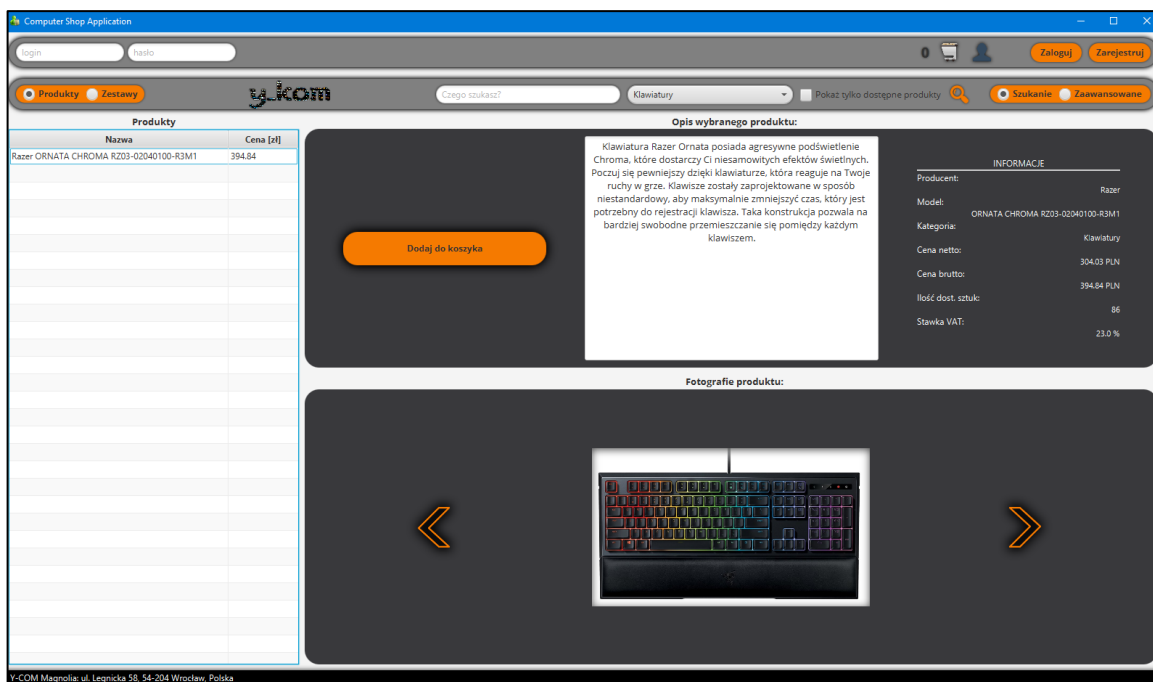
5.3.6. Próba dodania do koszyka produktu, którego ilość wynosi 0



Rysunek 37. Komunikat informujący o nieudanej próbie dodania produktu do koszyka z powodu jego braku na stanie.

5.3.7. Wyszukiwanie produktów

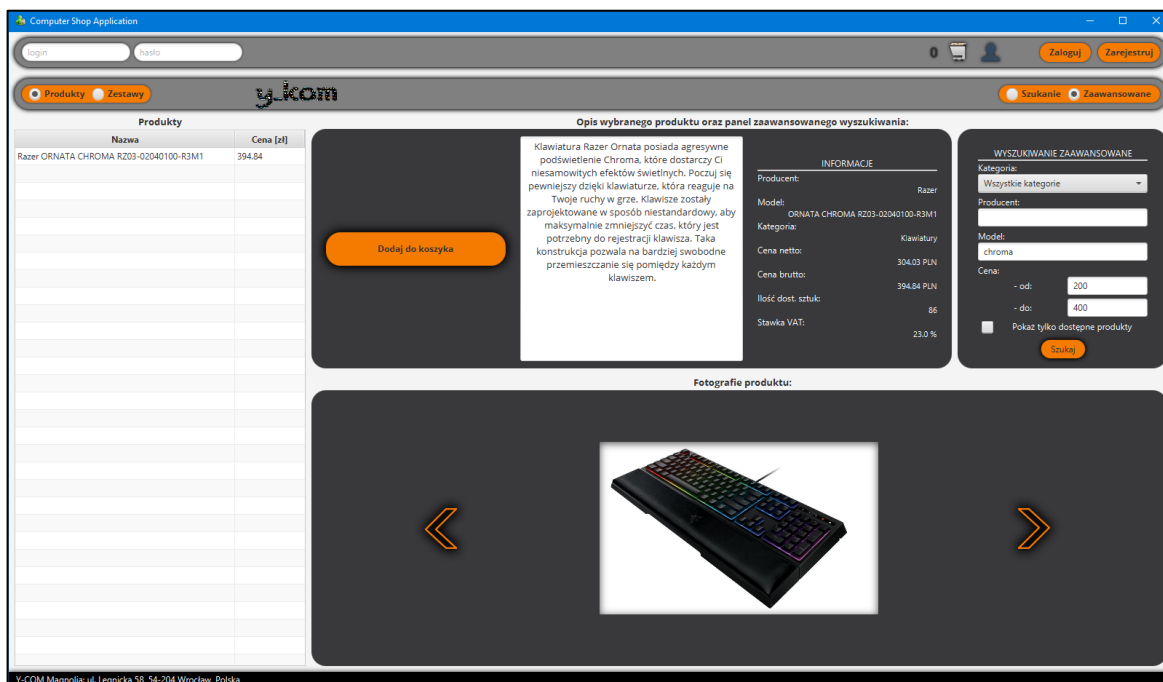
Wyszukiwanie produktów zostało ograniczone do kategorii „Klawiatury”.



Rysunek 38. Wynik wyszukiwania produktów wg kategorii „Klawiatury”.

5.3.8. Zaawansowane wyszukiwanie produktów

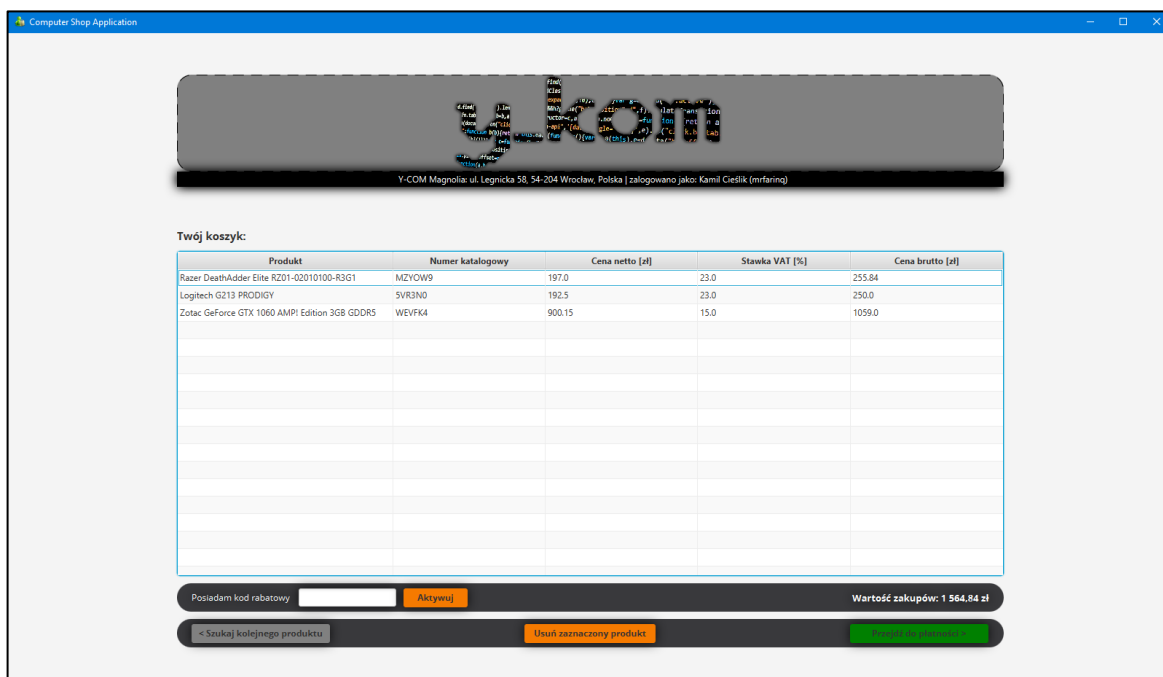
Wyszukiwanie odbyło się poprzez filtrowanie produktów wg słowa kluczowego wyszukiwanego w modelach produktów – „chroma”.



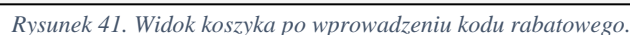
Rysunek 39. Wynik zaawansowanego wyszukiwania produktów po słowie kluczowym „chroma” wyszukiwanym w modelach produktów.

5.3.9. Działanie kodu rabatowego

- koszyk przed wprowadzeniem kodu rabatowego



Rysunek 40. Widok koszyka przed wprowadzeniem kodu rabatowego.



5.4.1. Metoda połączenia z bazą danych

Session (sesja) jest to jeden z głównych interfejsów wykorzystywanych w aplikacji Hibernate. Jest nieodporny na wielowątkowość, niedrogi w tworzeniu i przy niszczeniu. Można sobie wyobrażać obiekt Session jako coś pomiędzy bazodanowym połączeniem a transakcją. Obiekt Session otrzymywany jest z SessionFactory. SessionFactory jest bardzo drogi przy otwieraniu i przy niszczeniu, a ponieważ jest odporny na wielowątkowość powinien być dzielony pomiędzy wiele wątków. Zazwyczaj aplikacja będzie wykorzystywała jeden obiekt SessionFactory, chyba że będzie korzystać z więcej niż jednej bazy danych, wtedy występuje potrzeba oddzielnych obiektów SessionFactory dla każdej bazy danych.

- SessionFactory dla aplikacji systemu zakupowego:

```
public abstract class DbSessionFactory {
    SessionFactory factory;
    DbSessionFactory() {
        System.out.println("Stworzenie fabryki.");
        factory = new Configuration()
            .configure()
            .addAnnotatedClass(Customer.class)
            .addAnnotatedClass(Address.class)
            .addAnnotatedClass(Account.class)
            .addAnnotatedClass(Order.class)
            .addAnnotatedClass(StationaryShop.class)
            .addAnnotatedClass(Discount.class)
            .addAnnotatedClass(Product.class)
            .addAnnotatedClass(ProductCategory.class)
            .addAnnotatedClass(ProductPhoto.class)
            .addAnnotatedClass(Employee.class)
            .addAnnotatedClass(OrderPosition.class)
            .addAnnotatedClass(ProductSet.class)
            .addAnnotatedClass(InfoAboutStationaryShop.class)
            .addAnnotatedClass(InfoAboutSet.class)
            .addAnnotatedClass(InfoAboutDiscounts.class)
            .addAnnotatedClass(InfoAboutOrder.class)
            .addAnnotatedClass(InfoAboutOrderPositions.class)
            .buildSessionFactory();
    }

    @Override
    public void finalize(){
        System.out.println("Zamknięcie fabryki.");
        factory.close();
    }
}
```

Kod programu 43. Implementacja otwarcia oraz zamknięcia fabryki obiektów BD.

5.4.2. Implementacja wybranych funkcjonalności sklepu

Ciekawą funkcją zaimplementowanego systemu zakupowego jest sposób przyznawania kodów rabatowych. Funkcjonalność tą opisuje poniższy listing.

```
List<Discount> loggedCustomerDiscounts = loggedCustomerAccount.getDiscounts();
Discount alreadyOwnedDiscount;
for (Map.Entry<Integer, Integer> entry : numberOfProductsOfEachCategory.entrySet()) {
    Integer keyCategoryId = entry.getKey();
    Integer valueAmountOfProducts = entry.getValue();
    alreadyOwnedDiscount = null;

    if (valueAmountOfProducts >= 3) {
        if (loggedCustomerDiscounts != null) {
            for (Discount dis : loggedCustomerDiscounts) {
                if ((dis.getProductCategory().getId() == keyCategoryId) && dis.getUsed() ==
                    false) {
                    alreadyOwnedDiscount = dis;
                    break;
                }
            }
        }
        if (alreadyOwnedDiscount != null) {
            if (alreadyOwnedDiscount.getDiscountPercentage() < 50) {
                alreadyOwnedDiscount.setDiscountPercentage(alreadyOwnedDiscount.getDiscountPercentage()
                    + 10);
                discountService.saveDiscount(alreadyOwnedDiscount);
                newDiscountCode = true;
            }
        } else {
            Discount discount = new Discount();
            discount.setAccount(this.loggedCustomerAccount);
            discount.setProductCategory(productCategoryService.getProductCategory(keyCategoryId));
            discount.setDiscountPercentage(10);
            discount.setUsed(false);
            discountService.saveDiscount(discount);
            newDiscountCode = true;
        }
    }
}
```

Kod programu 44. Implementacja przyznawania kodów rabatowych.

Sposób nadawania kodów rabatowych wg powyższej implementacji został dokładnie omówiony w punkcie 5.3.4.

Dzięki frameworkowi Hibernate język HQL umożliwia łatwe w implementacji tworzenie zaawansowanych funkcji wyszukiwujących. Wyszukiwanie produktów opisuje poniższy listing.

```
public List<Product> searchEntities(String category, String name, String producer, String model, Float priceFrom, Float priceTo, Boolean amount) {
    List<Product> products = null;
    try (Session currentSession = factory.getCurrentSession()) {
        currentSession.beginTransaction();
        Query<Product> theQuery = currentSession.createQuery("select p from Product p " +
            "where p.name like:searchedName and (:searchedCategory like 'Wszystkie kategorie' or " +
            "p.productCategory.name=:searchedCategory) and p.producer like:searchedProducer " +
            "and p.model like:searchedModel and (:searchedAmount!=false or p.amount > 0) " +
            "and p.sellingPriceBrutto >=:searchedPriceFrom " +
            "and p.sellingPriceBrutto <=:searchedPriceTo"
            , Product.class).setParameter("searchedName", "%" + name + "%")
            .setParameter("searchedAmount", amount)
            .setParameter("searchedCategory", category)
            .setParameter("searchedProducer", "%" + producer + "%")
            .setParameter("searchedModel", "%" + model + "%")
            .setParameter("searchedPriceFrom", priceFrom)
            .setParameter("searchedPriceTo", priceTo);
        products = theQuery.getResultList();
        currentSession.getTransaction().commit();
    } catch (Exception exc) {
        exc.printStackTrace();
    }
    return products;
}
```

Kod programu 45. Implementacja zaawansowanego wyszukiwania produktów w języku HQL.

5.4.3. Implementacja mechanizmów bezpieczeństwa

W aplikacji wprowadzono szereg zabezpieczeń kontrolujących poprawne działanie sklepu. Bardzo pomocne okazały się wyrażenia regularne, które kontrolują poprawność wprowadzania adresu e-mail, loginu, hasła użytkownika. Wprowadzanie danych osobowych takich jak: kod pocztowy, adres zamieszkania, czy numer telefonu również jest kontrolowane przez wyrażenia regularne.

Przykładowa metoda sprawdzania poprawności nazwy ulicy wraz z numerem domu/mieszkania:

```
@FXML
void textFieldStreet_onAction() {
    String streetPattern = "^[A-ZŹĆĄŚŁŃ] [a-zżćńółęś]+\\s([A-ZŹĆĄŚŁŃ] " +
        "[a-zżćńółęś]+ \\s)?[1-9][0-9]*[A-Z]?(/[1-9][0-9]*[A-Z]?)?$";
    if (textFieldStreet.getText().isEmpty())
        labelStreet.setText("Podaj ulicę i nr domu.");
    else if (!textFieldStreet.getText().matches(streetPattern)) {
        labelStreet.setText("Niepoprawny format.");
    } else if (textFieldStreet.getText().length() > 40)
        labelStreet.setText("Przekroczono limit znaków.");
    else
        labelStreet.setText("");
}
```

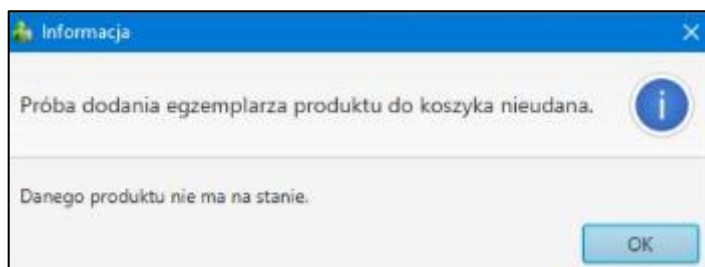
Kod programu 46. Metoda sprawdzająca poprawność wpisanej nazwy ulicy wraz z numerem domu/mieszkania i ustawiająca wartość stosownego komunikatu etykiety.

Akceptowalne nazwy ulic to ulice maksymalnie dwuwyrazowe, z jednym znakiem białym pomiędzy każdym wyrazem/cyfrą. Po spacji następującej po nazwie ulicy musi wystąpić oznaczenie budynku/mieszkania. Akceptowalne oznaczenia to np. 39/41, 39A/41, 39/41A, 39A/41B, 3.

Wiele testów potwierdzających efektywne działanie zabezpieczeń aplikacji zostało udokumentowane w punkcie dot. testowania – 5.3.

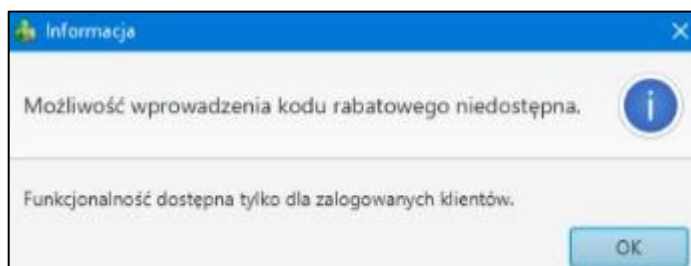
Kolejne zabezpieczenia to m.in.:

- użytkownik nie ma możliwości zakupu towaru, którego ilość w bazie danych wynosi zero



Rysunek 42. Komunikat zwrotny systemu w sytuacji próby dodania do koszyka produktu, którego ilość wynosi 0.

- zbieranie oraz używanie kodów rabatowych tylko z poziomu zalogowanego użytkownika



Rysunek 43. Komunikat zwrotny systemu w sytuacji próby nałożenia kodu rabatowego na koszyk z pozycji niezalogowanego użytkownika.

W przypadku niepoprawnego podania jakichkolwiek danych, np. e-mail, klient również także zostanie poinformowany odpowiednim komunikatem. Aplikacja zapewnia pełne zabezpieczenie przed wykonywaniem niedozwolonych operacji po stronie klienta.

6. Wnioski

Do realizacji projektu koniecznym było zapoznanie się z nowymi środowiskami programistycznymi i technologiami, które umożliwiły końcowy efekt działania aplikacji. Ponadto należało zapoznać się z implementacją bazy danych, sposobem poruszania się po niej, korzystania z jej zasobów oraz łączenia się z nią. W implementacji użyto nowoczesnych technologii takich jak JavaFX, Hibernate. Początkowo projekt celował w aplikację obsługującą zarówno panel administratora jak i klienta, jednak podjęto decyzję, aby aplikacja została w pełni stworzona z myślą o kliencie. Panel administratora może zostać stworzony w osobnej aplikacji. Aplikacja staje się tym samym dużo bardziej bezpieczna, występuje mniejsza szansa, że do bazy dostaną się niepowołani użytkownicy.

Reszta początkowych założeń została wykonana zgodnie z planami. Cały panel kliencki jest w pełni rozwinięty i gotowy do użytkowania. W przyszłości planowane jest rozwinięcie aplikacji oraz wprowadzenie wersji internetowej.