**Temat: Sklep z odzieżą**

**1 etap - Specyfikacja wymagań**

**Autorzy:**   
**Mateusz Błach 264010, Michał Lewandowski 264458, Jakub Jakubowicz 263925**

**Wstęp:**

Nasz projekt zakłada stworzenie aplikacji bazodanowej, która usprawni zarządzanie sklepem z odzieżą online. Aplikacja ta ma na celu ułatwić klientom przeglądanie i zakup produktów, a jednocześnie umożliwić administratorom i pracownikom sprawną obsługę sklepu. Aplikacja zostanie podzielona na trzy główne grupy użytkowników: administratorów, pracowników i klientów, z każdą grupą posiadającą dostęp do określonych funkcjonalności. Dla administratorów, głównym celem będzie efektywne zarządzanie produktami, zamówieniami oraz kontami klientów. Pracownicy będą odpowiedzialni za akceptację i anulację zamówień, a klienci będą mogli wygodnie przeglądać i zamawiać produkty.

# Wymagania funkcjonalne:

## Dla administratora:

1. **Logowanie:** Administrator ma możliwość zalogowania się do systemu za pomocą unikalnego identyfikatora (login) i hasła
2. **Zarządzanie produktami:** Administrator ma dostęp do funkcji dodawania, edytowania i usuwania produktów w bazie danych.
3. **Zarządzanie zamówieniami:** Administrator powinien być w stanie przeglądać, edytować i usuwać.
4. **Zarządzanie kontami:** Administrator ma prawo tworzyć, edytować i usuwać konta klientów sklepu.
5. **Backup i odzyskiwanie danych:** Administrator ma możliwość stworzenia backupu stanu produktów, użytkowników oraz zamówień.

## Dla pracownika:

1. **Przeglądanie produktów:** Pracownik może przeglądać produkty w bazie, sprawdzać dostępność, filtrowanie produktów po różnych parametrach.
2. **Logowanie:** Pracownik ma możliwość zalogowania się do systemu za pomocą unikalnego identyfikatora (login) i hasła.
3. **Akceptacja zamówienia:** Każde zamówienie złożone przez klienta musi zostać zaakceptowane przez pracownika.
4. **Anulowanie zamówienia:** Każde zamówienie złożone przez klienta może zostać anulowane przez pracownika.

## Dla klienta:

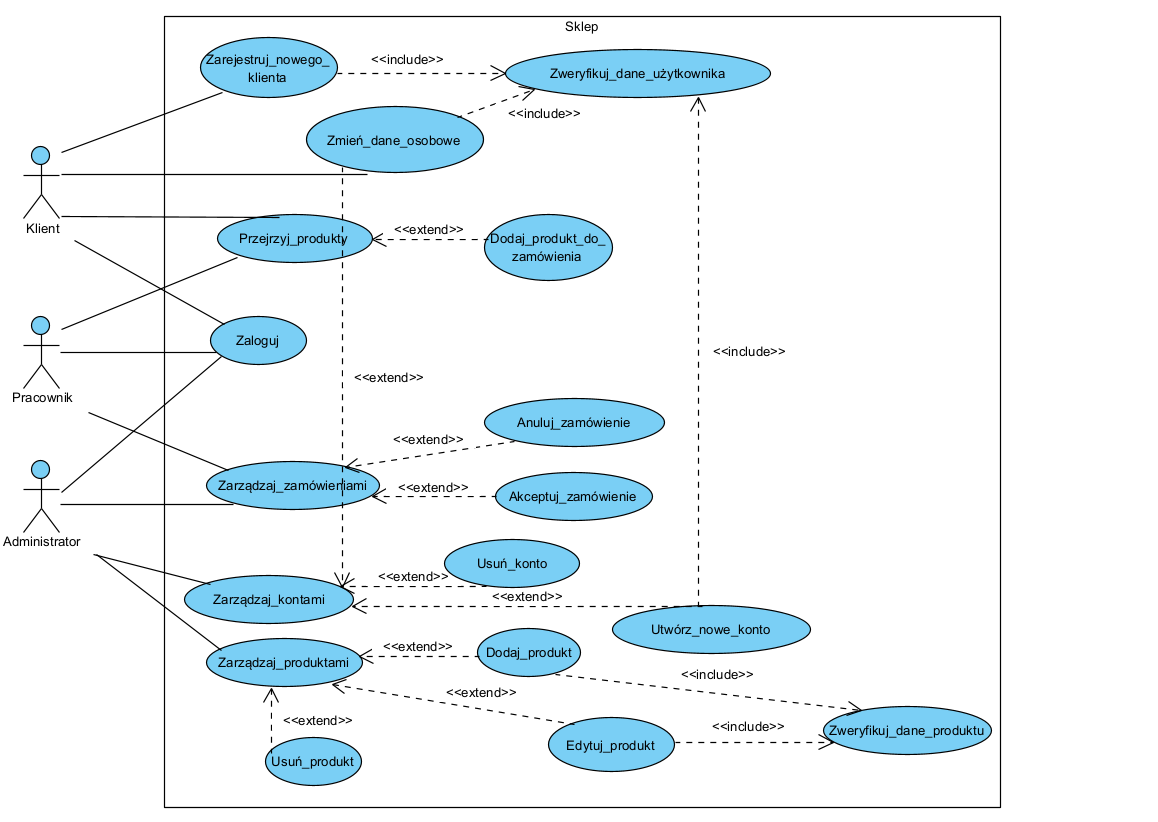
1. **Zmiana danych osobowych:** Klienci mogą usunąć swoje konto z bazy oraz edytować swoje dane.
2. **Logowanie:** Klienci mogą logować się za pomocą loginu i hasła.
3. **Rejestracja:** Klienci mogą stworzyć konto.
4. **Przeglądanie produktów:** Klienci mogą przeglądać produkty według kategorii, cen i marki.
5. **Dodawanie do zamówienia:** Klienci mogą dodawać produkty do zamówienia.
6. **Składanie zamówień:** Klienci mogą składać zamówienia podając dane dostawy i płatności.
7. **Śledzenie zamówień:** Klienci mogą sprawdzać status swoich zamówień i historię zakupów.

## Dla wszystkich użytkowników:

1. **Weryfikacja danych logowania:** Weryfikacja czy konto istnieje w bazie danych dla podanych danych logowania.
2. **Weryfikacja wprowadzonych danych:** Weryfikacja wprowadzonych danych.

# Wymagania niefunkcjonalne:

1. **Autoryzacja operacji:** Każda operacja na bazie danych jest dostępna w zależności od typu konta użytkownika.
2. **Walidacja wprowadzonych danych:** Weryfikacja czy podane dane przez użytkownika są prawidłowe.
3. **Wykorzystanie procedur**: Wszystkie zapytania kierowane z aplikacji do bazy danych będą wykonywane poprzez procedury.
4. **Hashowanie haseł:** Zastosowanie funkcji haszujących do bezpiecznego przechowywania haseł w bazie danych.
5. **Jedno konto administratora:** Istnienie jednego wbudowanego konta administratora w bazie danych
6. **Wymaganie logowania:** Konieczność logowania się dla klientów korzystających z aplikacji bazodanowej
7. **Akceptacja zamówień**: Każde zamówienie musi zostać zaakceptowane przez odpowiedniego pracownika przed dalszym przetwarzaniem.

Diagram przypadków użycia:

**Scenariusze przypadków użycia:**  
**Nazwa PU: Zarejestruj\_nowego\_klienta**

**Cel: Zarejestrowanie nowego klienta**

**Warunki początkowe: Uruchomienie programu aplikacji bazodanowej**

**Warunki końcowe: Wprowadzenie unikalnej nazwy użytkownika**

**Scenariusz:**

1. **Należy podać imię, nazwisko, adres, nazwę użytkownika i hasło.**
2. **Należy sprawdzić czy dany użytkownik już nie istnieje i czy hasło jest odpowiednio skonstruowane poprzez PU Zweryfikuj\_wprowadzone\_dane**
3. **Jeżeli wprowadzone dane nie są prawidłowe należy zakończyć PU, w przeciwnym wypadku należy zapisać dane w bazie danych.**

**Nazwa PU: Zweryfikuj\_dane\_użytkownika**

**Cel: Zweryfikowanie poprawności wprowadzonych danych**

**Warunki początkowe: Jest uruchamiany z następujących PU: Zarejestruj\_nowego\_klienta, Utwórz\_nowe\_konto, Zmień\_dane\_osobowe**

**Warunki końcowe: Zwraca wynik określający czy wprowadzone dane są prawidłowe**

**Scenariusz:**

1. **Porównuje nazwę użytkownika z nazwami użytkownika istniejącymi już w bazie danych.**
2. **W przypadku znalezienia użytkownika o podanej nazwie zwraca te nazwę użytkownika i kończy przeszukiwanie bazy danych.**
3. **W przypadku nieznalezienia podanej nazwy użytkownika zwraca wynik negatywny**

**Nazwa PU: Zmień\_dane\_osobowe**

**Cel: Zmiana danych osobowych klienta**

**Warunki początkowe: Uruchomienie programu aplikacji bazodanowej**

**Warunki końcowe: Wprowadzona nazwa użytkownika musi być unikalna**

**Scenariusz:**

1. **Należy podać dane, które chcemy zmienić**
2. **Jeżeli wśród zmienionych danych występuje nazwa użytkownika to uruchamia PU Zwerfikuj\_dane\_użytkownika.**
3. **Jeżeli wprowadzone dane nie są prawidłowe należy zakończyć PU, w przeciwnym wypadku należy zapisać dane w bazie danych.**

**Nazwa PU: Przejrzyj\_produkty**

**Cel: Przegląd dostępnych produktów**

**Warunki początkowe: Uruchomienie programu aplikacji bazodanowej**

**Warunki końcowe: Kryteria przeglądania muszą być poprawne**

**Scenariusz:**

1. **Użytkownik aplikacji podaje określone kryteria wedle których aplikacja filtruje produkty.**
2. **Zgodnie z określonymi kryteriami zwracana jest lista produktów.**

**Nazwa PU: Dodaj\_produkt\_do\_zamówienia**

**Cel: Dodanie wybranego produktu do zamówienia**

**Warunki początkowe: Jest uruchamiany z PU Przejrzyj\_produkty**

**Warunki końcowe: Wybrany produkt musi być dostępny**

**Scenariusz:**

1. **Użytkownik wybiera produkt**
2. **Jeżeli produkt jest dostępny to dodaje go do zamówienia, w przeciwnym wypadku kończy PU**

**Nazwa PU: Zaloguj**

**Cel: Zalogowanie użytkownika**

**Warunki początkowe: Uruchomienie programu aplikacji bazodanowej**

**Warunki końcowe: Użytkownik musi podać prawidłowe dane logowania**

**Scenariusz:**

1. **Należy podać nazwę użytkownika oraz hasło**
2. **Sprawdzana jest poprawność wprowadzonych danych**
3. **Jeżeli podano prawidłowe dane logowania to użytkownik ma dostęp do aplikacji, w przeciwnym wypadku kończymy PU**

**Nazwa PU: Zarządzaj\_zamówieniami**

**Cel: Zarządzanie zamówieniami**

**Warunki początkowe: Uruchomienie programu aplikacji bazodanowej**

**Warunki końcowe: Użytkownik musi być administratorem lub pracownikiem**

**Scenariusz:**

1. **Użytkownik wybiera zamówienie, które może zaakceptować korzystając z PU Akcpetuj\_zamówienie lub anulować korzystając z PU Anuluj\_zamówienie**
2. **W przypadku edycji zamówienia należy zapisać zmiany w bazie danych, w innym przypadku należy zakończyć PU**

**Nazwa PU: Anuluj\_zamówienie**

**Cel: Anulowanie zamówienia**

**Warunki początkowe: Jest uruchamiany z PU Zarządzaj\_zamówieniami**

**Warunki końcowe: Zamówienie musi znajdować się w bazie**

**Scenariusz:**

1. **W przypadku anulowania zamówienia jest ono usuwane z bazy danych**
2. **W innym wypadku należy zakończyć PU**

**Nazwa PU: Akceptuj\_zamówienie**

**Cel: Akceptacja zamówienia**

**Warunki początkowe: Jest uruchamiany z PU Zarządzaj\_zamówieniami**

**Warunki końcowe: Zamówienie musi istnieć**

**Scenariusz:**

1. **W przypadku akceptacji zamówienia jest ono dodawane do bazy danych**
2. **W innym wypadku należy zakończyć PU**

**Nazwa PU: Zarządzaj\_kontami**

**Cel: Zarządzanie kontami użytkowników**

**Warunki początkowe: Uruchomienie programu aplikacji bazodanowej**

**Warunki końcowe: Użytkownik musi być administratorem**

**Scenariusz:**

1. **Użytkownik wybiera konto i może edytować dane osobowe korzystając z PU Zmień\_dane\_osobowe lub usunać konto korzystając z PU Usuń\_konto**
2. **W przypadku, kiedy konto nie istnieje użytkownik może je utworzyć korzystając z PU Utwórz\_nowe\_konto**
3. **W wypadku edycji/utworzenia/usunięcia konta należy zapisać zmiany w bazie danych w innym przypadku należy zakończyć PU**

**Nazwa PU: Usuń\_konto**

**Cel: Usunięcie konta użytkownika**

**Warunki początkowe: Jest uruchamiany z PU Zarządzaj\_kontami**

**Warunki końcowe: Wybrane konto musi znajdować się w bazie**

**Scenariusz:**

1. **W przypadku usunięcia konta jest ono usuwane z bazy danych**
2. **W przeciwnym wypadku należy zakończyć PU**

**Nazwa PU: Utwórz\_nowe\_konto**

**Cel: Utworzenia nowego konta użytkownika**

**Warunki początkowe: Jest uruchamiany z PU Zarządzaj\_kontami**

**Warunki końcowe: Wprowadzone dane do konta muszą być unikalne**

**Scenariusz:**

1. **Należy zweryfikować czy podane dane do konta nie znajdują się już w bazie danych za pomocą PU Zweryfikuj\_dane\_użytkownika**
2. **Jeżeli wprowadzone dane są unikalne należy zapisać nowe konto w bazie danych, w przeciwnym wypadku należy zakończyć PU**

**Nazwa PU: Zarządzaj\_produktami**

**Cel: Zarządzanie produktami**

**Warunki początkowe: Uruchomienie programu aplikacji bazodanowej**

**Warunki końcowe: Użytkownik musi być administratorem**

**Scenariusz:**

1. **Użytkownik wybiera produkt, który może edytować korzystając z PU Edytuj\_produkt lub usunąć korzystając z PU Usuń\_produkt**
2. **Jeżeli produkt nie istnieje użytkownik może go utworzyć korzystająć z PU Dodaj\_produkt**
3. **W przypadku edycji/utworzenia/usunięcia produktu należy zapisać zmiany w bazie danych, w przeciwnym wypadku należy zakończyć PU**

**Nazwa PU: Dodaj\_produkt**

**Cel: Dodawanie produktu**

**Warunki początkowe: Jest uruchamiany z PU Zarządzaj produktami**

**Warunki końcowe: Wprowadzone dane produktu muszą być unikalne**

**Scenariusz:**

1. **Należy zweryfikować czy wprowadzone dane produktu są unikalne za pomocą PU Zweryfikuj\_dane\_produktu**
2. **Jeżeli wprowadzone dane produktu są unikalne i poprawne następuje dodanie produktu do bazy danych**
3. **W innym przypadku należy zakończyć PU**

**Nazwa PU: Zweryfikuj\_dane\_produktu**

**Cel: Weryfikacja poprawności danych produktu**

**Warunki początkowe: Jest uruchamiany z PU Dodaj\_produkt lub PU Edytuj\_produkt**

**Warunki końcowe: Zwraca wynik określający czy wprowadzone dane są poprawne lub nie**

**Scenariusz:**

1. **Porównuje dane produktu z danymi produktów znajdujących się już w bazie danych**
2. **W przypadku znalezienia produktu o podanych danych zwracany jest wynik pozytywny i następuje koniec przeszukiwania bazy danych**
3. **W innym przypadku zwracany jest wynik negatywny**

**Nazwa PU: Usuń\_produkt**

**Cel: Usunięcie produktu**

**Warunki początkowe: Jest uruchamiany z PU Zarządzaj\_produktami**

**Warunki końcowe: Wybrany produkt musi znajdować się w bazie**

**Scenariusz:**

1. **W przypadku usunięcia produkt jest usuwany z bazy danych**
2. **W innym wypadku należy zakończyć PU**

**Nazwa PU: Edytuj\_produkt**

**Cel: Edycja produktu**

**Warunki początkowe: Jest uruchamiany z PU Zarządzaj\_produktami**

**Warunki końcowe: Wprowadzone dane produktu muszą być unikalne**

**Scenariusz:**

1. **Należy zweryfikować czy wprowadzone dane produktu nie znajdują się już w bazie danych za pomocą PU Zweryfikuj\_dane\_produktu**
2. **Jeżeli dane produktu nie pokrywają się z danymi z bazy danych należy zaktualizować produkt w bazie danych, w innym wypadku należy zakończyć PU**

**2 etap - Projekt, implementacja i testy bazy danych**

1. Identyfikacja encji.

Tabela Users:

* user\_id
* login
* password
* email
* phone
* name
* surname
* role

Tabela Orders:

* order\_id
* user\_id
* clothes\_id
* date
* amount
* delivery\_id
* payment\_id

Tabela Clothes:

* clothes\_id
* material
* size
* sex
* price
* collection\_id

Tabela Collections:

* collection\_id
* name
* start\_date
* end\_date

Tabela Delivery:

* delivery\_id
* city
* street
* number
* postal\_\_code
* country

Tabela Payments:

* payment\_id
* status
* payment\_form
* date

1. Sformułowanie wymagań dotyczących dostępu do bazy i jej zawartości.

Użytkownik z bazy danych może:

* Pobierać informację znajdując się w bazie danych o swoim koncie, poza jego user\_id

SELECT login, name, surname, role, email, phone FROM Users WHERE user\_id = :user\_id;

* Modyfikować niektóre informacje o swoim koncie (użytkownik nie może modyfikować user\_id, login, name, surname, role):

UPDATE Users SET email = :email, phone = :phone WHERE user\_id = :user\_id;

* Pobierać dostępne w tabeli Clothes ubrania i filtrować wyszukiwania po wszystkich jej atrybutach oraz filtrować po atrybutach z tabeli Collections

SELECT \* FROM Clothes WHERE collection\_id IN (

SELECT collection\_id FROM Collections WHERE start\_date <= CURRENT\_DATE AND end\_date >= CURRENT\_DATE

);

SELECT \* FROM Clothes

WHERE (:material IS NULL OR material = :material)

AND (:size IS NULL OR size = :size)

AND (:sex IS NULL OR sex = :sex)

AND (:price IS NULL OR price <= :max\_price);

* Tworzyć zamówienie:

INSERT INTO Orders (user\_id, clothes\_id, date, amount, delivery\_id, payment\_id)

VALUES (:user\_id, :clothes\_id, :date, :amount, :delivery\_id, :payment\_id);

* Tworzyć (dokonywać) płatności za stworzone przez siebie zamówienie:

INSERT INTO Payments (status, payment\_form, date, user\_id)

VALUES (:status, :payment\_form, CURRENT\_DATE, :user\_id);

* Pobierać informacje o płatności dokonanej przez swoje konto:

SELECT status, payment\_form, date FROM Payments WHERE payment\_id = :payment\_id;

* Tworzyć adres dostawy dla zamówienia:

INSERT INTO Delivery (city, street, number, postal\_code, country)

VALUES (:city, :street, :number, :postal\_code, :country);

* Modyfikować adres dostawy swoich zamówień:

UPDATE Delivery SET city = :city, street = :street, number = :number, postal\_code = :postal\_code, country = :country

WHERE delivery\_id = :delivery\_id;

1. Opracowanie diagramu konceptualnego modelu bazy.

A screenshot of a computer

Description automatically generated

1. Opracowanie diagramu fizycznego modelu bazy

A diagram of a computer

Description automatically generated

3 etap – Projekt, implementacja i testy aplikacji bazodanowej

1. Makieta interfejsu graficznego aplikacji
   1. Główne Menu

Obraz zawierający tekst, wizytówka, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* 1. Zarządzanie kontem

Obraz zawierający tekst, zrzut ekranu, linia, Prostokąt

Opis wygenerowany automatycznie

* 1. Zarządzanie zamówieniami

Obraz zawierający tekst, zrzut ekranu, wizytówka, Czcionka

Opis wygenerowany automatycznie

* 1. Zarządzanie koszykiem

Obraz zawierający tekst, zrzut ekranu, Równolegle, linia

Opis wygenerowany automatycznie

* 1. Przeglądanie artykułów

Obraz zawierający tekst, zrzut ekranu, Równolegle, linia

Opis wygenerowany automatycznie

1. Diagramy czynności

2.1 Rejestracja nowego klienta

Obraz zawierający zrzut ekranu, tekst, diagram, linia

Opis wygenerowany automatycznie

2.2 Weryfikacja wprowadzonych danych

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznie

2.3 Zmiana danych osobowych

Obraz zawierający tekst, diagram, linia, Wykres

Opis wygenerowany automatycznie

2.4 Przeglądanie produktów według podanych kryteriów

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznie

2.5 Dodawanie produktu do zamówienia

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

2.6 Logowanie do aplikacji

Obraz zawierający tekst, linia, diagram, Wykres

Opis wygenerowany automatycznie

2.7 Zarządzanie zamówieniami

Obraz zawierający tekst, diagram, linia, Wykres

Opis wygenerowany automatycznie

2.8 Anulowanie zamówienia

Obraz zawierający tekst, linia, diagram, Wykres

Opis wygenerowany automatycznie

2.9 Akceptacja zamówienia

Obraz zawierający tekst, linia, diagram, Wykres

Opis wygenerowany automatycznie

2.10 Zarządzanie kontami

Obraz zawierający tekst, diagram, zrzut ekranu, design

Opis wygenerowany automatycznie

2.11 Usuwanie konta

Obraz zawierający tekst, linia, diagram, Wykres

Opis wygenerowany automatycznie

2.12 Tworzenie nowego konta

Obraz zawierający tekst, linia, zrzut ekranu, Wykres

Opis wygenerowany automatycznie

2.13 Zarządzanie produktami

Obraz zawierający tekst, diagram, linia, Wykres

Opis wygenerowany automatycznie

2.14 Dodawanie nowego produktu

Obraz zawierający tekst, linia, diagram, Wykres

Opis wygenerowany automatycznie

2.15 Weryfikacja danych produktu

Obraz zawierający zrzut ekranu, tekst, diagram, linia

Opis wygenerowany automatycznie

2.16 Usuwanie produktu

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznie

2.17 Edycja danych produktu

Obraz zawierający tekst, diagram, linia, Wykres

Opis wygenerowany automatycznie

1. Diagram klas

Obraz zawierający tekst, zrzut ekranu, diagram, Prostokąt

Opis wygenerowany automatycznie

1. Wykonanie i wdrożenie aplikacji

Aplikacja została wykonana i wdrożona w języku Pyhon. Poniżej zostały przedstawione i omówione kody klas, z których się składa:

* **Basket.py** – klasa reprezentuje koszyk zakupowy;
* **Clothes.py** – klasa reprezentuje ubrania;
* **Collection.py** – klasa reprezentuje kolekcje ubrań;
* **Db\_communication.py** - klasa odpowiada za komunikację z bazą danych MySQL;
* **Delivery.py** – klasa obsługuje informacje dotyczące dostaw zamówień;
* **Order.py** – Klasa odpowiada za zarządzanie zamówieniami użytkowników;
* **Payment.py** – Klasa odpowiada za zarządzanie płatnościami;
* **Security\_utility.py** – Klasa odpowiedzialna za zabezpieczenie hasła;
* **Ui\_app.py** – Klasa implementuje interfejs odpowiedzialny za komunikację z użytkownikiem;
* **User.py** – Klasa reprezentuje użytkownika w systemie sklepu;
* **Validation\_utility.py** – Klasa zawiera zestaw narzędzi do walidacji danych wejściowych.
* **Basket.py**: *Obraz zawierający tekst, dokument, menu, zrzut ekranu

  Opis wygenerowany automatycznie*

*Funkcja* ***\_\_init\_\_*** *:*

*Inicjalizuje nową instancję koszyka. Ustala połączenie z bazą danych, tworzy nowe zamówienie dla użytkownika, pobiera jego identyfikator i inicjalizuje zmienną `calculated\_price` jako `None`.*

*Funkcja* ***assign\_order\_id*** *:*

*Zwraca najnowszy identyfikator zamówienia z bazy danych, inkrementując go o 1. Wykorzystywane jest to, aby przypisać unikalny identyfikator zamówienia do koszyka.*

*Funkcja* ***add\_clothes*** *:*

*Dodaje ubranie o określonym identyfikatorze do koszyka przypisanego bieżącemu zamówieniu.*

*Funkcja* ***calculate\_price*** *:*

*Oblicza łączną cenę ubrań w koszyku na podstawie cen ubrań z bazy danych.*

*Funkcja* ***display\_clothes*** *:*

*Wyświetla szczegóły dotyczące materiału, rozmiaru, płci i ceny ubrań znajdujących się w koszyku.*

*Funkcja* ***remove\_clothes*** *:*

*Usuwa określone ubranie z koszyka przypisanego bieżącemu zamówieniu.*

*Funkcja* ***view\_edit\_basket*** *:*

*Wyświetla zawartość koszyka w pętli, umożliwia dodawanie lub usuwanie ubrań oraz wyjście z interaktywnego widoku koszyka.*

* **Clothes.py**:  
    
  Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

  Opis wygenerowany automatycznie

Funkcja **\_\_init\_\_** :

Inicjalizuje nowy obiekt klasy **Clothes**. Przypisuje wartości atrybutom, takim jak identyfikator ubrania, materiał, rozmiar, płeć, cena, identyfikator kolekcji oraz daty rozpoczęcia i zakończenia kolekcji. Dodaje utworzone ubranie do listy wszystkich ubrań **\_all\_clothes**.

Funkcja **print\_all\_clothes** :

Wyświetla szczegóły wszystkich ubrań przechowywanych w liście **\_all\_clothes**, takie jak identyfikator, materiał, rozmiar, płeć, cena, nazwa kolekcji oraz daty rozpoczęcia i zakończenia kolekcji.

Funkcja **get\_clothes\_by\_id** :

Znajduje i zwraca ubranie o określonym identyfikatorze.

Funkcja **fetch\_clothes\_from\_db** :

Pobiera informacje o ubraniach z bazy danych i aktualizuje listę **\_all\_clothes**.

Funkcja **create\_clothes\_with\_collection** :

Tworzy obiekt ubrania z przypisanymi informacjami o kolekcji.

Funkcja **refresh\_clothes** :

Aktualizuje listę **\_all\_clothes**, pobierając najnowsze informacje z bazy danych.

Funkcja **create\_clothes** :

Tworzy i zwraca ubranie z podanymi danymi.

Funkcja **print\_sorted\_by\_price** :

Wyświetla posortowane ubrania według ceny, z możliwością ustalenia rosnącego lub malejącego porządku.

Funkcja **print\_sorted\_by\_collection\_date** :

Wyświetla posortowane ubrania według daty rozpoczęcia kolekcji, z możliwością ustalenia kolejności od najnowszych do najstarszych.

Funkcja **print\_filtered\_clothes** :

Pozwala na filtrowanie ubrań na podstawie różnych kryteriów, takich jak materiał, rozmiar, cena czy płeć. Wyświetla ubrania spełniające kryteria użytkownika.

* **Collection.py** :  
    
  Obraz zawierający tekst, zrzut ekranu, dokument, menu

  Opis wygenerowany automatycznie

Funkcja **\_\_init\_\_** :

Inicjalizuje nowy obiekt klasy `Collection`. Przypisuje wartości atrybutom, takim jak identyfikator kolekcji, nazwa, data rozpoczęcia i zakończenia. Dodaje utworzoną kolekcję do listy wszystkich kolekcji **\_all\_collections**.

Funkcja **print\_all\_collections** :

Wyświetla szczegóły wszystkich kolekcji przechowywanych w liście **\_all\_collections**, takie jak identyfikator, nazwa, data rozpoczęcia i zakończenia.

Funkcja **get\_collection\_by\_id** :

Znajduje i zwraca kolekcję o określonym identyfikatorze.

Funkcja **fetch\_collections\_from\_db** :

Pobiera informacje o kolekcjach z bazy danych i aktualizuje listę `\_all\_collections`.

Funkcja **refresh\_collections** :

Aktualizuje listę **\_all\_collections**, pobierając najnowsze informacje z bazy danych.

Funkcja create\_collection :

Tworzy i zwraca nową kolekcję z podanymi danymi, dodając ją jednocześnie do listy **\_all\_collections**.

* **Db\_communication** :  
    
  Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

  Opis wygenerowany automatycznie

Funkcja **\_\_init\_\_** :

Inicjalizuje nowy obiekt klasy **DB\_Communication**. Inicjalizuje atrybuty **conn** i **cursor** na wartość **None** oraz wykonuje połączenie z bazą danych poprzez wywołanie metody **connect\_to\_db**.

Funkcja **connect\_to\_db** :

Próbuje nawiązać połączenie z bazą danych MySQL, korzystając z podanych danych dostępowych (host, użytkownik, hasło, port, nazwa bazy danych). Jeśli połączenie się powiedzie, ustawia atrybuty **conn** i **cursor** na odpowiednie wartości. W przypadku błędu wypisuje komunikat o nieudanej próbie połączenia.

* **Delivery.py** :  
    
  Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

  Opis wygenerowany automatycznie

Atrybut klasowy **\_delivery\_details** :

Lista przechowująca wszystkie dostępne informacje o dostawach.

Funkcja **\_\_init\_\_** :

Inicjalizuje nowy obiekt klasy **Delivery**. Przypisuje wartości atrybutom takim jak identyfikator dostawy (**delivery\_id**), miasto (**city**), ulica (**street**), numer (**number**), kod pocztowy (**postal\_code**), kraj (**country**). Dodaje utworzoną dostawę do listy **\_delivery\_details**.

Funkcja **fetch\_deliveries\_by\_user** :

Pobiera informacje o dostawach związanych z danym użytkownikiem na podstawie identyfikatora użytkownika. Czyści wcześniejsze informacje o dostawach i aktualizuje listę **\_delivery\_details**.

Funkcja **print\_delivery\_details** :

Wyświetla szczegóły dostaw przechowywanych w liście **\_delivery\_details**, takie jak identyfikator, miasto, ulica, numer, kod pocztowy i kraj.

Funkcja **insert\_new\_delivery** :

Pozwala użytkownikowi wprowadzić nowe dane dostawy. Dodaje te dane do bazy danych, a następnie aktualizuje atrybut **delivery\_id** utworzonego obiektu. Komunikuje użytkownikowi o pomyślnym dodaniu danych dostawy do bazy danych.

Funkcja **select\_delivery** :

Wyświetla dostępne dostawy, a następnie pozwala użytkownikowi wybrać dostawę na podstawie identyfikatora. Zwraca identyfikator wybranej dostawy lub **None**, jeśli dostawa o podanym identyfikatorze nie istnieje.

Funkcja **menu** :

Zapewnia interaktywne menu zarządzania dostawami, umożliwiając użytkownikowi wybór opcji takich jak wyświetlanie, wybieranie lub dodawanie dostaw. Loop kończy się, gdy użytkownik wybierze opcję "**Exit**".

* **Order.py** :  
    
  Obraz zawierający tekst, dokument, menu, zrzut ekranu

  Opis wygenerowany automatycznie

Obraz zawierający tekst, menu, papier, dokument

Opis wygenerowany automatycznie  
Funkcja **fetch\_orders\_by\_user** :

Pobiera zamówienia użytkownika z bazy danych na podstawie identyfikatora użytkownika. Aktualizuje słownik **orders** przechowujący szczegóły zamówień, uwzględniając informacje o płatnościach, dostawach, ubraniach i kolekcjach.

Funkcja **print\_all\_orders\_by\_user** :

Wyświetla szczegóły wszystkich zamówień użytkownika, takie jak identyfikator zamówienia, kwota, status płatności, forma płatności, data płatności, adres dostawy oraz szczegóły ubrań i kolekcji w zamówieniu.

Funkcja **cancel\_order** :

Anuluje zamówienie o podanym identyfikatorze. Wywołuje procedurę składowaną "**CancelOrder**" na podstawie identyfikatora zamówienia.

Funkcja **print\_order\_details** :

Wyświetla szczegóły zamówienia o podanym identyfikatorze, takie jak identyfikator zamówienia, kwota, status płatności, forma płatności, data płatności, adres dostawy oraz szczegóły ubrań i kolekcji w zamówieniu.

Funkcja **add\_new\_order** :

Dodaje nowe zamówienie do bazy danych na podstawie przekazanych informacji, takich jak kwota, identyfikator użytkownika, identyfikator płatności i identyfikator dostawy.

Funkcja **update\_order\_attributes** :

Aktualizuje atrybuty zamówienia o podanym identyfikatorze, takie jak kwota, identyfikator płatności i identyfikator dostawy.

* **Payment.py** :  
    
  Obraz zawierający tekst, dokument, zrzut ekranu, Czcionka

  Opis wygenerowany automatycznie

Funkcja **\_\_init\_\_** :

Inicjalizuje nowy obiekt klasy **Payment**, ustawiając atrybuty na wartości początkowe.

Funkcja **fetch\_payments\_by\_user** :

Pobiera informacje o płatnościach użytkownika z bazy danych na podstawie identyfikatora użytkownika. Aktualizuje listę **payment\_details**, przechowującą szczegóły płatności.

Funkcja **print\_payment\_details** :

Wyświetla szczegóły wszystkich dostępnych płatności przechowywanych w liście **payment\_details**.

Funkcja **insert\_new\_payment** :

Pozwala użytkownikowi wprowadzić nowe dane płatności, takie jak forma płatności (karta, blik, przelew). Dodaje nową płatność do bazy danych i zwraca jej identyfikator.

Funkcja **select\_saved\_payment\_details** :

Pozwala na wybór zapisanych szczegółów płatności na podstawie identyfikatora płatności. Ustawia atrybuty obiektu na wybrane szczegóły płatności.

Funkcja **menu** :

Obsługuje menu opcji płatności, umożliwiając użytkownikowi wybór działań takich jak wyświetlanie, dodawanie nowej płatności oraz wybór zapisanych szczegółów płatności.

* **Security\_utility.py** :

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Funkcja **hash\_password** :

Przyjmuje hasło jako argument, a następnie używa algorytmu SHA-256 do wygenerowania skrótu (hashu) tego hasła. Do hashowania dodawany jest tajny sól (SECRET\_SALT), co zwiększa bezpieczeństwo operacji. Ostateczny zhashowany wynik jest reprezentowany jako szesnastkowy ciąg znaków i zwracany jako wynik funkcji. Wprowadzenie tajnej soli jest praktyką stosowaną w celu utrudnienia ataków typu "**rainbow table**" oraz zwiększenia odporności na ataki bruteforce.

* **Ui\_app.py** :

Obraz zawierający tekst, dokument, zrzut ekranu, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, dokument, menu, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, dokument, zrzut ekranu, menu

Opis wygenerowany automatycznie

Funkcja **login\_menu** :

Obsługuje proces logowania i tworzenia nowego konta. W zależności od wyboru użytkownika może prowadzić do utworzenia nowego konta, zalogowania się lub wyjścia z programu.

Funkcja **main\_menu** :

Główna pętla interfejsu, gdzie użytkownik ma dostęp do różnych funkcji, takich jak zarządzanie kontem, zamówieniami, koszykiem, ubraniami. Umożliwia również wylogowanie.

Funkcja **order\_menu** :

Zarządza opcjami związanymi z zamówieniami, takimi jak przeglądanie wszystkich zamówień, anulowanie wybranego zamówienia czy wyświetlanie szczegółów zamówienia.

Funkcja **basket\_menu** :

Odpowiada za zarządzanie koszykiem użytkownika, umożliwia edycję zawartości koszyka, wybór opcji dostawy i płatności oraz finalizację zamówienia.

Funkcja **clothes\_menu**:   
  
Obsługuje funkcje związane z ubraniami, takie jak przeglądanie dostępnych ubrań, sortowanie według ceny czy kolekcji, oraz filtrowanie ubrań.

Funkcja **create\_account** :   
  
Obsługuje proces tworzenia nowego konta, sprawdzając unikalność loginu oraz dodając użytkownika do bazy danych.

Funkcja **login** :   
  
Obsługuje proces logowania, sprawdzając poprawność wprowadzonych danych w porównaniu do informacji w bazie danych.

Funkcja **main** :   
  
Funkcja główna programu, inicjuje obiekt klasy **UI\_App** i uruchamia interfejs użytkownika, prowadzący przez proces logowania i obsługujący główne funkcje programu.

* **User.py** :

Obraz zawierający tekst, zrzut ekranu, dokument

Opis wygenerowany automatycznie

Funkcja **\_\_init\_\_** :

Inicjalizuje obiekt User z podanymi atrybutami.

Funkcja **user\_menu** :

Odpowiada za interakcję z użytkownikiem poprzez wyświetlanie menu z opcjami związanymi z zarządzaniem kontem. Umożliwia użytkownikowi wybór między wyświetlaniem, aktualizacją atrybutów, usunięciem konta lub wyjściem z menu.

Funkcja **display\_attributes** :

Wyświetla atrybuty użytkownika, takie jak login, e-mail, numer telefonu, imię, nazwisko i rolę.

Funkcja **update\_attributes** :

Pozwala użytkownikowi na aktualizację wybranych atrybutów, takich jak e-mail, numer telefonu, imię i nazwisko. Przyjmuje nowe dane od użytkownika i aktualizuje informacje w bazie danych.

Funkcja **delete\_account** :

Obsługuje proces usuwania konta użytkownika. Weryfikuje potwierdzenie użytkownika przed usunięciem konta, a następnie wykonuje procedurę usuwania konta w bazie danych.

* **Validation\_utlity.py** :

Obraz zawierający tekst, dokument, zrzut ekranu, menu

Opis wygenerowany automatycznie

Funkcja validate\_for\_sql\_injection :

Sprawdza, czy wartość zawiera potencjalnie szkodliwe frazy związane z atakami SQL injection. W przypadku wykrycia takich fraz, wyświetla komunikat ostrzegawczy.

Funkcja validate\_varchar :

Waliduje ciąg znaków (varchar) pod kątem długości, upewniając się, że nie przekracza maksymalnej długości. Zwraca wartość logiczną w zależności od wyniku walidacji.

Funkcja validate\_integer :

Waliduje liczbę całkowitą pod kątem długości, upewniając się, że nie przekracza maksymalnej liczby cyfr. Również sprawdza, czy wartość jest poprawną liczbą całkowitą. Zwraca wartość logiczną w zależności od wyniku walidacji.

Funkcja validate\_decimal :

Waliduje liczbę dziesiętną pod kątem liczby cyfr przed i po przecinku. Również sprawdza, czy wartość jest poprawną liczbą dziesiętną. Zwraca wartość logiczną w zależności od wyniku walidacji.

Funkcja get\_validated\_input\_varchar :

Pobiera wejściowy ciąg znaków od użytkownika, a następnie stosuje walidację dla wartości varchar i SQL injection. Zwraca wprowadzony ciąg znaków, jeśli jest poprawny.

Funkcja get\_validated\_input\_integer :

Pobiera wejściową liczbę całkowitą od użytkownika, a następnie stosuje walidację dla wartości całkowitej. Zwraca wprowadzoną liczbę całkowitą, jeśli jest poprawna.

Funkcja get\_validated\_input\_decimal :

Pobiera wejściową liczbę dziesiętną od użytkownika, a następnie stosuje walidację dla wartości dziesiętnej. Zwraca wprowadzoną liczbę dziesiętną, jeśli jest poprawna.

1. Przetestowanie aplikacji

W celu przetestowania aplikacji napisaliśmy testy. Testy są napisane przy użyciu frameworka pytest oraz unittest.mock do tworzenia atrap obiektów. Testy sprawdzają różne aspekty funkcji i metod w aplikacji, takie jak sprawdzanie poprawności danych, wyświetlanie informacji, aktualizacja atrybutów użytkownika, obsługa zamówień i dostaw, oraz interakcje z bazą danych. Parametryzacja testów pozwala na łatwe dodawanie nowych przypadków testowych.   
  
Poniżej znajduje się kod testów w Pythonie:

import pytest  
from unittest.mock import patch, Mock, MagicMock  
  
from decimal import Decimal  
from db\_communication import DB\_Communication  
from ui\_app import UI\_App  
from user import User  
from validation\_utility import ValidationUtility  
from order import Order  
from delivery import Delivery  
  
  
@pytest.fixture  
def mock\_db():  
 db = Mock(spec=DB\_Communication)  
 db.cursor = Mock()  
 db.conn = Mock()  
 return db  
  
  
@pytest.fixture  
def user():  
 return User(1, 'test\_login', 'password123', 'test@example.com', '1234567890', 'John', 'Doe', 'user')  
  
  
@pytest.mark.parametrize("value, expected", [  
 ("safe\_input", True),  
 ("SELECT \* FROM users", False),  
 ("pass1; DROP TABLE users", False),  
 ("sdffsdfsd-- comment", False),  
])  
def test\_validate\_for\_sql\_injection(value, expected):  
 assert ValidationUtility.validate\_for\_sql\_injection(value) == expected  
  
  
@pytest.mark.parametrize("value, max\_length, expected", [  
 ("short\_string", 20, True),  
 ("a" \* 21, 20, False),  
 (123, 20, False),  
])  
def test\_validate\_varchar(value, max\_length, expected):  
 assert ValidationUtility.validate\_varchar(value, max\_length) == expected  
  
  
@pytest.mark.parametrize("value, max\_length, expected", [  
 ("12345", 5, True),  
 ("123456", 5, False),  
 ("abc", 5, False),  
])  
def test\_validate\_integer(value, max\_length, expected):  
 assert ValidationUtility.validate\_integer(value, max\_length) == expected  
  
  
@pytest.mark.parametrize("value, expected", [  
 ("12.34", True),  
 ("5232132.45", False),  
 ("abc", False),  
])  
def test\_validate\_decimal(value, expected):  
 assert ValidationUtility.validate\_decimal(value) == expected  
  
  
def test\_display\_attributes(capsys, user):  
 user.display\_attributes()  
 captured = capsys.readouterr()  
 assert (  
 'Login: test\_login\nEmail: test@example.com\nPhone: 1234567890\nName: John\nSurname: Doe\nRole: user\n') in captured.out  
  
  
@patch('builtins.input', side\_effect=["newemail@example.com", "", "John", ""])  
def test\_update\_attributes(mock\_input):  
 db\_mock = MagicMock()  
 user = User(1, 'testuser', 'password', 'test@example.com', '1234567890', 'Jane', 'Doe', 'customer')  
 user.update\_attributes(db\_mock)  
 assert user.email == "newemail@example.com"  
 assert user.name == "John"  
  
  
  
@patch('builtins.input', side\_effect=["Test City", "Test Street", "123", "12345", "Test Country"])  
def test\_insert\_new\_delivery(mock\_input):  
 db\_mock = MagicMock()  
 delivery = Delivery()  
 delivery.insert\_new\_delivery(db\_mock)  
  
  
def test\_fetch\_orders\_by\_user():  
 db\_mock = MagicMock()  
 user\_id = 1  
 expected\_orders = [  
 (1, 100, 'completed', 'card', '2024-01-01', 'City', 'Street', 123, '12345', 'Country', 'Cotton', 'M', 'Male',  
 50.0, 'Summer Collection', '2024-01-01', '2024-06-01')  
 ]  
 db\_mock.cursor.fetchall.return\_value = expected\_orders  
 Order.fetch\_orders\_by\_user(db\_mock, user\_id)  
 assert user\_id in Order.orders  
  
  
def test\_cancel\_order\_success():  
 db\_mock = MagicMock()  
 order\_id = 1  
 Order.cancel\_order(db\_mock, order\_id)  
 db\_mock.cursor.callproc.assert\_called\_with("CancelOrder", [order\_id])  
 db\_mock.conn.commit.assert\_called\_once()  
  
  
@patch('builtins.input', side\_effect=["new\_user33", "password123"])  
def test\_complete\_order\_workflow(mock\_input):  
 db\_mock = MagicMock()  
 db\_mock.cursor.fetchone.side\_effect = [  
 ("3", "new\_user33", "password123", "new.user@example.com", "123-456-7890", "Bob", "Doe", "customer"),  
 ]  
  
 app = UI\_App()  
 app.db = db\_mock  
 assert app.login()  
 db\_mock.cursor.fetchone.side\_effect = [  
 (Decimal("60.00"), 3, 3, 3),  
 ]  
 Order.add\_new\_order(db\_mock, Decimal("60.00"), 3, 3, 3)  
 db\_mock.cursor.callproc.assert\_called\_with("addOrder", [Decimal("60.00"), 3, 3, 3])