

**UNIWERSYTET RZESZOWSKI**  
**Wydział Nauk Ścisłych i Technicznych**



Piotr Smoła  
nr albumu: 125162  
Kierunek: Informatyka

# **System rekomendacji produktów oparty na filtracji treści, logice rozmytej i modelach probabilistycznych**

Praca inżynierska

Praca wykonana pod kierunkiem  
dr inż. Piotra Grochowskiego

Rzeszów, 2026

## Streszczenie

Niniejsza praca inżynierska przedstawia projekt i implementację systemu rekomendacji produktów dla platformy e-commerce, wykorzystującego trzy zaawansowane metody uczenia maszynowego: filtrację opartą na treści (Content-Based Filtering), logikę rozmytą (Fuzzy Logic) oraz modele probabilistyczne (Markov Chain i Naive Bayes).

Głównym celem pracy było opracowanie modułowego systemu rekomendacyjnego, który może działać bez zewnętrznych bibliotek uczenia maszynowego, zapewniając pełną kontrolę nad algorytmami i możliwość ich dostosowania do specyficznych wymagań biznesowych. System został zaimplementowany od podstaw w języku Python z wykorzystaniem framework'a Django na backendzie oraz React na frontendzie.

W ramach pracy zrealizowano następujące zadania:

- Zaprojektowano architekturę systemu rekomendacyjnego z trzema niezależnymi silnikami
- Zaimplementowano algorytm Content-Based Filtering oparty na ważonych wektorach cech i mierze podobieństwa kosinusowego
- Opracowano system wnioskowania rozmytego typu Mamdani z 6 regułami IF-THEN i funkcjami przynależności trójkątnymi oraz trapezoidalnymi
- Zbudowano łańcuch Markowa pierwszego rzędu do predykcji sekwencji zakupowych oraz naiwny klasyfikator Bayesa do analizy zachowań klientów
- Przeprowadzono ewaluację wydajności i jakości rekomendacji na rzeczywistych danych e-commerce

Wyniki pokazują, że każda z metod ma swoje unikalne zastosowania: CBF rozwiązuje problem zimnego startu dla nowych produktów, logika rozmyta oferuje najwyższą interpretowalność rekomendacji, a modele probabilistyczne zapewniają głęboką personalizację na podstawie historii zakupów.

**Słowa kluczowe:** systemy rekomendacyjne, filtracja oparta na treści, logika rozmyta, łańcuch Markowa, naiwny klasyfikator Bayesa, e-commerce, Django, React

# Abstract

This thesis presents the design and implementation of a product recommendation system for an e-commerce platform, utilizing three advanced machine learning methods: Content-Based Filtering, Fuzzy Logic, and Probabilistic Models (Markov Chain and Naive Bayes).

The main objective was to develop a modular recommendation system that operates without external machine learning libraries, providing full control over algorithms and the ability to customize them for specific business requirements. The system was implemented from scratch in Python using the Django framework for the backend and React for the frontend.

The work accomplished the following tasks:

- Designed a recommendation system architecture with three independent engines
- Implemented Content-Based Filtering algorithm based on weighted feature vectors and cosine similarity measure
- Developed a Mamdani-style fuzzy inference system with 6 IF-THEN rules and triangular/trapezoidal membership functions
- Built a first-order Markov Chain for purchase sequence prediction and a Naive Bayes classifier for customer behavior analysis
- Conducted performance and quality evaluation on real e-commerce data

Results demonstrate that each method has unique applications: CBF solves the cold-start problem for new products, fuzzy logic offers the highest recommendation interpretability, and probabilistic models provide deep personalization based on purchase history.

**Keywords:** recommender systems, content-based filtering, fuzzy logic, Markov chain, Naive Bayes classifier, e-commerce, Django, React

# Spis treści

<b>Streszczenie</b>	<b>2</b>
<b>Abstract</b>	<b>3</b>
<b>Wstęp</b>	<b>6</b>
<b>Rozdział 1: Teoretyczne podstawy metod rekomendacyjnych</b>	<b>10</b>
1.1 Content-Based Filtering — podstawy teoretyczne . . . . .	10
1.2 Logika rozmyta — podstawy teoretyczne . . . . .	12
1.3 Modele probabilistyczne — podstawy teoretyczne . . . . .	14
1.4 Metryki oceny systemów rekomendacyjnych . . . . .	17
<b>Rozdział 2: Content-Based Filtering</b>	<b>18</b>
2.1 Architektura systemu CBF . . . . .	18
2.2 Implementacja ekstrakcji cech . . . . .	19
2.3 Algorytm podobieństwa kosinusowego . . . . .	20
2.4 Generowanie macierzy podobieństw . . . . .	21
2.5 Panel debugowania Content-Based Filtering . . . . .	22
2.6 Interfejs użytkownika - sortowanie według CBF . . . . .	25
<b>Rozdział 3: Logika rozmyta</b>	<b>27</b>
3.1 Architektura systemu Fuzzy Logic . . . . .	27
3.2 Funkcje przynależności . . . . .	28
3.3 Rozmyty profil użytkownika . . . . .	29
3.4 Baza reguł rozmytych . . . . .	30
3.5 Wnioskowanie i defuzzyfikacja . . . . .	31
3.6 Panel debugowania Fuzzy Logic . . . . .	32
3.7 Interfejs użytkownika - rekomendacje Fuzzy Logic . . . . .	36
<b>Rozdział 4: Modele probabilistyczne</b>	<b>38</b>
4.1 Architektura systemu probabilistycznego . . . . .	38
4.2 Łańcuch Markowa dla sekwencji zakupowych . . . . .	39
4.3 Naiwny klasyfikator Bayesa . . . . .	40
4.4 Integracja modeli . . . . .	41
4.5 API probabilistyczne . . . . .	42
4.6 Panel debugowania modeli probabilistycznych . . . . .	44
4.7 Interfejs użytkownika - rekomendacje probabilistyczne . . . . .	47
<b>Rozdział 5: Architektura techniczna systemu</b>	<b>49</b>
5.1 Stos technologiczny . . . . .	49

5.2 Backend — Django REST Framework . . . . .	50
5.3 Frontend — React 18 . . . . .	50
5.4 Diagram przypadków użycia . . . . .	50
5.5 Baza danych — modele . . . . .	51
<b>Rozdział 6: Wyniki i ewaluacja</b>	<b>53</b>
6.1 Metodologia testowania . . . . .	53
6.2 Wydajność Content-Based Filtering . . . . .	54
6.3 Wydajność Fuzzy Logic . . . . .	55
6.4 Wydajność modeli probabilistycznych . . . . .	55
6.5 Porównanie metod . . . . .	56
6.6 Wnioski . . . . .	56
<b>Podsumowanie</b>	<b>57</b>

# Wstęp

## Motywacja i kontekst problemu

Współczesne platformy e-commerce oferują dziesiątki tysięcy produktów, co prowadzi do problemu przeładowania informacją (information overload). Użytkownik poszukujący konkretnego produktu musi przeszukać setki ofert, często rezygnując z zakupu z powodu trudności w podjęciu decyzji. Systemy rekomendacyjne rozwiązują ten problem poprzez automatyczne dopasowanie produktów do preferencji i potrzeb klienta.

Według badań Ricci et al. (2015) w "Recommender Systems Handbook", systemy rekomendacyjne zwiększą konwersję o 10-30% i są kluczowym elementem strategii e-commerce największych platform. Netflix szacuje, że ich system rekomendacji jest wart ponad \$1 miliard rocznie w utrzymaniu klientów (Gomez-Uribe & Hunt, 2016). Amazon raportuje, że 35% ich sprzedaży pochodzi z rekomendacji produktowych.

Problem rekomendacji produktów jest złożony z kilku powodów. Po pierwsze, preferencje użytkowników są subiektywne i trudne do modelowania — jeden klient ceni jakość, inny niską cenę, a jeszcze inny markę. Po drugie, dane o użytkownikach są często niekompletne (problem zimnego startu) — nowy użytkownik nie ma historii zakupów, nowy produkt nie ma opinii. Po trzecie, katalogi produktów dynamicznie się zmieniają — pojawiają się nowe produkty, znikają stare, zmieniają się ceny i dostępność.

W odpowiedzi na te wyzwania powstały różnorodne metody rekomendacyjne, z których każda ma swoje zalety i ograniczenia. Niniejsza praca koncentruje się na trzech zaawansowanych podejściach: Content-Based Filtering (filtracja oparta na treści), Fuzzy Logic (logika rozmyta) oraz Probabilistic Models (modele probabilistyczne).

## Przegląd metod rekomendacyjnych

Systemy rekomendacyjne można podzielić na kilka głównych kategorii według mechanizmu działania:

**Collaborative Filtering (CF)** — najpopularniejsza metoda w systemach komercyjnych. Zakłada, że użytkownicy o podobnych preferencjach w przeszłości będą mieli podobne w przyszłości. Istnieją dwa warianty: User-Based (porównuje użytkowników) i Item-Based (porównuje produkty). Zalety: odkrywa nieoczywiste powiązania między produktami. Wady: problem zimnego startu dla nowych użytkowników i produktów, macierz danych jest rzadka.

**Content-Based Filtering (CBF)** — analizuje cechy samych produktów i dopasowuje je do profilu użytkownika. Zalety: brak problemu zimnego startu dla nowych produktów (wystarczy opis), przejrzystość rekomendacji. Wady: problem "filter bubble"— rekomenduje tylko podobne produkty, nie odkrywa nieoczywistych powiązań.

**Knowledge-Based Systems** — wykorzystują jawne reguły i wiedzę ekspercką. Zalety: pełna kontrola nad rekomendacjami. Wady: wymagają manualnego definiowania reguł, nie skalują się automatycznie.

**Hybrid Methods** — łączą różne podejścia dla uzyskania lepszych wyników. Netflix używa kombinacji CF, content features i metadanych. W tej pracy zaimplementowano hybrydę CBF, logiki rozmytej i modeli probabilistycznych.

## Zakres i cel pracy

W niniejszej pracy zaimplementowałem trzy zaawansowane metody rekomendacji dla platformy e-commerce:

**1. Content-Based Filtering (CBF)** — metoda oparta na ważonych wektorach cech produktów z wykorzystaniem podobieństwa kosinusowego (Pazzani & Billsus, 2007). Analizuje cechy produktu: kategorię (40%), tagi (30%), przedział cenowy (20%) i słowa kluczowe z opisu (10%). Algorytm oblicza podobieństwo między produktami na podstawie ich cech, eliminując problem zimnego startu dla nowych produktów.

Innowacyjność mojej implementacji polega na:

- Empirycznej optymalizacji wag cech metodą Grid Search
- Dyskretyzacji ceny na kategorie (low, medium, high, premium) zamiast wartości ciągły
- Ekstrakcji słów kluczowych z opisów produktów z filtracją stop-words
- Optymalizacji dla wektorów rzadkich (sparse vectors)

**2. Logika rozmyta (Fuzzy Logic)** — system wnioskowania rozmytego typu Mamdani (Mamdani, 1975) modelujący niepewność w preferencjach użytkownika. Wykorzystuje funkcje przynależności (trójkątne i trapezoidalne) dla zmiennych: cena, jakość i popularność. Sześć reguł rozmytych IF-THEN generuje rekomendacje uwzględniające wrażliwość cenową użytkownika i dopasowanie kategorii.

System logiki rozmytej stanowi unikalne podejście do problemu rekomendacji, ponieważ:

- Modeluje niepewność w preferencjach użytkownika (np. "tani" produkt to nie precyzyjna granica cenowa)

- Umożliwia płynne przejścia między kategoriami (produkt może być jednocześnie "średnio drogi" i "drogi")
- Oferuje najwyższą interpretowalność — każda rekomendacja ma wyjaśnienie w postaci aktywacji reguł
- Integruję rozmyty profil użytkownika (wrażliwość cenowa, preferencje kategorii)

**3. Modele probabilistyczne** — łańcuchy Markowa pierwszego rzędu (Rabiner, 1989) do predykci sekwencji zakupowych oraz naiwny klasyfikator Bayesa (Murphy, 2012) do przewidywania prawdopodobieństwa zakupu i ryzyka rezygnacji klienta (churn). Kombinacja obu modeli umożliwia personalizację rekomendacji w oparciu o historię i profil użytkownika.

Implementacja modeli probabilistycznych obejmuje:

- Łańcuchy Markowa na poziomie kategorii produktów (nie pojedynczych produktów)
- Naiwny Bayes z wygładzaniem Laplace'a dla uniknięcia zerowych prawdopodobieństw
- Predykcję churnu na podstawie wzorców behawioralnych
- Agregację wyników z wagami: Markov (60%) + Naive Bayes (40%)

## Dane i środowisko testowe

System został przetestowany na rzeczywistych danych z aplikacji e-commerce:

- **500 produktów** — komputery, laptopy, podzespoły, peryferia (48 kategorii)
- **20 użytkowników** — 5 administratorów + 15 klientów z historią zakupów
- **Zamówienia** — każdy użytkownik posiada historię zakupów (dane z seedera)
- **Opinie** — produkty posiadają opinie i oceny klientów
- **Stos technologiczny** — Django 4.2, React 18, PostgreSQL 14, NumPy

Struktura danych produktów obejmuje:

- Relacje ManyToMany z kategoriami i tagami
- Specyfikacje techniczne (JSON field)
- Opisy produktów (TextField, średnio 200-400 słów)

- Ceny i rabaty (DecimalField)
- Statystyki: view\_count, order\_count, average\_rating

## Cele pracy

Główne cele zrealizowane w ramach projektu:

- **Architektura:** Zaprojektowanie modułowego systemu rekomendacyjnego z trzema niezależnymi silnikami, które mogą działać samodzielnie lub w kombinacji.
- **Implementacja:** Napisanie algorytmów CBF, Fuzzy Logic i modeli probabilistycznych od podstaw bez zewnętrznych bibliotek ML (scikit-learn, TensorFlow), co zapewnia pełną kontrolę nad parametrami.
- **Optymalizacja:** Cache wielopoziomowy, batch processing i indeksowanie dla wydajności produkcyjnej.
- **Ewaluacja:** Pomiar jakości rekomendacji i wydajności na rzeczywistych danych z aplikacji.
- **Dokumentacja:** Przygotowanie diagramów UML (Use Case, Sequence, ERD) i dokumentacji technicznej.
- **Integracja:** Pełna integracja z frontendem React i panelami debugowania dla administratora.

## Struktura pracy

Praca składa się z sześciu rozdziałów. Rozdział 1 przedstawia podstawy teoretyczne metod rekomendacyjnych z formalizmem matematycznym. Rozdziały 2-4 opisują szczegółową implementację trzech metod: Content-Based Filtering, logiki rozmytej i modeli probabilistycznych, wraz z panelami debugowania i zrzutami interfejsu. Rozdział 5 dokumentuje architekturę techniczną systemu (Django backend, React frontend, PostgreSQL) — ta część jest wspólna z pracą współautora. Rozdział 6 zawiera wyniki eksperymentów, analizę wydajności i porównanie metod.

# Rozdział 1

## Teoretyczne podstawy metod rekomendacyjnych

### 1.1 Content-Based Filtering — podstawy teoretyczne

Content-Based Filtering (filtracja oparta na treści) jest jedną z fundamentalnych metod systemów rekomendacyjnych. W przeciwieństwie do Collaborative Filtering, CBF analizuje cechy samych produktów, a nie wzorce zachowań użytkowników. Metoda została szczegółowo opisana przez Pazzani & Billsus (2007) w pracy "Content-Based Recommendation Systems"[1].

**Zasada działania:** System buduje profil cech każdego produktu (wektor cech) i oblicza podobieństwo między produktami na podstawie ich cech. Użytkownikowi rekomendowane są produkty podobne do tych, które wcześniej przeglądał lub kupił.

#### Reprezentacja produktu jako wektora cech

Każdy produkt  $p$  jest reprezentowany jako wektor w wielowymiarowej przestrzeni cech:

$$\vec{p} = (f_1, f_2, \dots, f_n) \quad (1)$$

gdzie  $f_i$  to waga cechy  $i$  (np. należenie do kategorii, posiadanie tagu, przedział cenowy). W mojej implementacji używam ważonych cech:

$$\vec{p} = \sum_{i \in categories} 0.40 \cdot \mathbf{1}_{cat_i}(p) + \sum_{j \in tags} 0.30 \cdot \mathbf{1}_{tag_j}(p) + 0.20 \cdot price\_range(p) + \sum_{k \in keywords} w_k \cdot \mathbf{1}_{kw_k}(p) \quad (2)$$

gdzie  $\mathbf{1}_{feature}(p)$  to funkcja indykatorowa (1 jeśli produkt ma cechę, 0 w przeciwnym razie).

#### Zalety CBF:

- Brak problemu zimnego startu dla nowych produktów — wystarczy opis i cechy
- Przejrzystość rekomendacji — można wyjaśnić dlaczego produkt został polecony ("podobna kategoria", "podobne tagi")
- Niezależność od innych użytkowników — działa nawet dla pierwszego klienta w systemie
- Szybka aktualizacja — dodanie nowego produktu nie wymaga przeliczenia całej macierzy

## Wady CBF:

- Problem "filter bubble"— rekomenduje tylko podobne produkty, użytkownik nie odkrywa nowych kategorii
- Wymaga dobrze opisanych cech produktów — jakość rekomendacji zależy od jakości metadanych
- Nie odkrywa nieoczywistych powiązań między produktami (np. "użytkownicy kupujący kawę często kupują cukier")
- Ograniczenie do podobieństwa cech — nie uwzględnia kontekstu użytkownika

**Podobieństwo kosinusowe** (Cosine Similarity) jest standardową metryką w CBF. Dla dwóch wektorów  $\vec{A}$  i  $\vec{B}$ :

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \times \|\vec{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}} \quad (3)$$

gdzie  $\vec{A}$  i  $\vec{B}$  to wektory cech dwóch produktów. Wynik mieści się w przedziale  $[0, 1]$  dla nieujemnych wektorów (w kontekście TF-IDF i wag binarnych).

### Interpretacja podobieństwa kosinusowego:

- $\cos(\theta) = 1$  — wektory identyczne (produkty mają te same cechy)
- $\cos(\theta) = 0$  — wektory ortogonalne (brak wspólnych cech)
- $\cos(\theta) \in (0, 1)$  — częściowe podobieństwo

### TF-IDF (Term Frequency - Inverse Document Frequency)

W kontekście ekstrakcji słów kluczowych z opisów produktów, wykorzystuje uproszczoną wersję TF-IDF:

$$TF(t, d) = \frac{count(t, d)}{|d|} \quad (4)$$

gdzie  $count(t, d)$  to liczba wystąpień terminu  $t$  w dokumencie  $d$ , a  $|d|$  to długość dokumentu (liczba słów).

W pełnej wersji TF-IDF:

$$TF-IDF(t, d, D) = TF(t, d) \times IDF(t, D) \quad (5)$$

gdzie  $IDF(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$  to odwrócona częstotliwość dokumentowa.

W mojej implementacji używam uproszczonej wersji TF bez IDF, ponieważ każdy produkt jest traktowany niezależnie.

## 1.2 Logika rozmyta — podstawy teoretyczne

Logika rozmyta (Fuzzy Logic) została wprowadzona przez Lotfi Zadeha w przełomowej pracy "Fuzzy Sets" (1965) [2]. Rozszerza klasyczną logikę dwuwartościową (prawda/fałsz) o stopnie przynależności w przedziale  $[0, 1]$ .

**Motywacja:** Klasyczna logika wymaga precyzyjnych granic. Pytanie "Czy produkt za 450 PLN jest tani?" nie ma jednoznacznej odpowiedzi — zależy od kontekstu, kategorii produktu i preferencji użytkownika. Logika rozmyta pozwala odpowiedzieć: "Produkt jest tani ze stopniem 0.3 i średnio drogi ze stopniem 0.7".

**Zbiory rozmyte** (Fuzzy Sets): W klasycznej teorii zbiorów element należy lub nie należy do zbioru. W zbiorach rozmytych element ma stopień przynależności  $\mu(x) \in [0, 1]$ . Formalnie, zbiór rozmyty  $A$  na uniwersum  $X$  jest zdefiniowany przez funkcję przynależności:

$$\mu_A : X \rightarrow [0, 1] \quad (6)$$

gdzie  $\mu_A(x)$  oznacza stopień przynależności elementu  $x$  do zbioru  $A$ .

**Przykład:** Dla zmiennej "cena" możemy zdefiniować trzy zbiory rozmyte:

- **cheap:** ceny niskie (pełna przynależność dla cen  $< 100$  PLN)
- **medium:** ceny średnie (pełna przynależność dla cen 500-1200 PLN)
- **expensive:** ceny wysokie (pełna przynależność dla cen  $> 2000$  PLN)

Produkt za 350 PLN może mieć:  $\mu_{cheap}(350) = 0.3$ ,  $\mu_{medium}(350) = 0.5$ ,  $\mu_{expensive}(350) = 0.0$ .

**Funkcje przynależności** (Membership Functions) definiują stopień przynależności elementu do zbioru rozmytego. Najczęściej stosowane typy:

*Funkcja trójkątna* (Triangular MF):

$$\mu_{triangle}(x; a, b, c) = \max \left( 0, \min \left( \frac{x - a}{b - a}, \frac{c - x}{c - b} \right) \right) \quad (7)$$

gdzie  $a$  to dolna granica,  $b$  to punkt maksymalny ( $\mu = 1$ ),  $c$  to górna granica.

*Funkcja trapezoidalna* (Trapezoidal MF):

$$\mu_{trapezoid}(x; a, b, c, d) = \max \left( 0, \min \left( \frac{x - a}{b - a}, 1, \frac{d - x}{d - c} \right) \right) \quad (8)$$

gdzie przedział  $[b, c]$  ma pełną przynależność ( $\mu = 1$ ), a  $[a, b]$  i  $(c, d]$  to obszary przejściowe.

*Funkcja gaussowska* (Gaussian MF):

$$\mu_{gaussian}(x; c, \sigma) = e^{-\frac{(x-c)^2}{2\sigma^2}} \quad (9)$$

gdzie  $c$  to środek (mean), a  $\sigma$  to odchylenie standardowe kontrolujące szerokość.

### Operacje na zbiorach rozmytych

*Uzupełnienie* (Negacja):

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (10)$$

*Przecięcie* (AND) — T-norma:

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) \quad (11)$$

Najczęściej używane T-normy:

- Minimum (Gödel):  $T_{min}(a, b) = \min(a, b)$
- Iloczyn algebraiczny:  $T_{prod}(a, b) = a \cdot b$
- Łukasiewicz:  $T_L(a, b) = \max(0, a + b - 1)$

*Suma* (OR) — T-conorma (S-norma):

$$\mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) \quad (12)$$

Najczęściej używane T-conormy:

- Maksimum:  $S_{max}(a, b) = \max(a, b)$
- Suma algebraiczna:  $S_{sum}(a, b) = a + b - a \cdot b$
- Łukasiewicz:  $S_L(a, b) = \min(1, a + b)$

**System wnioskowania Mamdani** (1975) [3] jest najbardziej rozpowszechnioną metodą wnioskowania rozmytego. Składa się z czterech etapów:

1. **Fuzzyfikacja** — przekształcenie wartości wejściowych na stopnie przynależności do zbiorów rozmytych. Przykład: cena 450 PLN  $\rightarrow \mu_{cheap} = 0.1, \mu_{medium} = 0.6, \mu_{expensive} = 0.0$ .
2. **Ewaluacja reguł** — obliczenie aktywacji reguł IF-THEN za pomocą T-norm. Dla reguły "IF price IS cheap AND quality IS high THEN recommendation IS strong":

$$\alpha = T(\mu_{cheap}(price), \mu_{high}(quality)) = \min(\mu_{cheap}, \mu_{high}) \quad (13)$$

3. **Agregacja** — połączenie wyników wszystkich reguł za pomocą T-conormy.

Jeśli wiele reguł prowadzi do tego samego konsekwentu:

$$\mu_{output} = S(\alpha_1, \alpha_2, \dots, \alpha_n) = \max(\alpha_1, \alpha_2, \dots, \alpha_n) \quad (14)$$

4. **Defuzzyfikacja** — przekształcenie wyniku rozmytego na wartość liczbową.

**Metody defuzzyfikacji:**

*Centroid* (środek ciężkości):

$$y^* = \frac{\int y \cdot \mu(y) dy}{\int \mu(y) dy} \quad (15)$$

*Średnia ważona* (Weighted Average) — uproszczona metoda używana w implementacji:

$$y^* = \frac{\sum_{i=1}^n \alpha_i \cdot w_i}{\sum_{i=1}^n w_i} \quad (16)$$

gdzie  $\alpha_i$  to aktywacja reguły  $i$ , a  $w_i$  to waga reguły.

*Mean of Maximum* (MoM):

$$y^* = \frac{1}{|M|} \sum_{y \in M} y, \quad M = \{y : \mu(y) = \max_z \mu(z)\} \quad (17)$$

### Reguły rozmyte IF-THEN

Reguła rozmyta ma postać:

$$\text{IF } x_1 \text{ IS } A_1 \text{ AND } x_2 \text{ IS } A_2 \text{ THEN } y \text{ IS } B \quad (18)$$

gdzie  $A_1, A_2, B$  to zbiory rozmyte. W systemie rekomendacji przykładowa reguła:

```
IF quality IS high AND price IS cheap
THEN recommendation IS strong
```

## 1.3 Modele probabilistyczne — podstawy teoretyczne

**Łańcuchy Markowa** (Markov Chains) zostały wprowadzone przez Andrieja Markowa w 1906 roku. Są procesami stochastycznymi spełniającymi własność Markowa — przyszły stan zależy tylko od stanu obecnego, nie od historii (Rabiner, 1989) [4].

*Definicja formalna:* Łańcuch Markowa to ciąg zmiennych losowych  $X_0, X_1, X_2, \dots$  przyjmujących wartości ze zbioru stanów  $S = \{s_1, s_2, \dots, s_n\}$ , spełniający własność Markowa:

$$P(X_{t+1} = s_j | X_t = s_i, X_{t-1} = s_{i-1}, \dots, X_0 = s_0) = P(X_{t+1} = s_j | X_t = s_i) \quad (19)$$

Oznacza to, że prawdopodobieństwo przejścia do stanu  $s_j$  zależy tylko od obecnego stanu  $s_i$ , nie od tego jak do niego dotarliśmy.

*Macierz przejść* (Transition Matrix)  $P$  zawiera prawdopodobieństwa przejść między stanami:

$$P_{ij} = P(X_{t+1} = s_j | X_t = s_i) \quad (20)$$

Macierz  $P$  spełnia warunki:

- $P_{ij} \geq 0$  dla wszystkich  $i, j$
- $\sum_j P_{ij} = 1$  dla wszystkich  $i$  (wiersze sumują się do 1)

*Estymacja prawdopodobieństw przejść* z danych:

$$\hat{P}_{ij} = \frac{\text{count}(s_i \rightarrow s_j)}{\sum_k \text{count}(s_i \rightarrow s_k)} \quad (21)$$

gdzie  $\text{count}(s_i \rightarrow s_j)$  to liczba obserwowanych przejść ze stanu  $s_i$  do stanu  $s_j$ .

*Rozkład stacjonarny* (Stationary Distribution)  $\pi$  spełnia:

$$\pi = \pi P, \quad \sum_i \pi_i = 1 \quad (22)$$

Jest to rozkład prawdopodobieństwa, który pozostaje niezmieniony po przejściu — reprezentuje długoterminowe prawdopodobieństwa przebywania w każdym stanie.

*Zastosowanie w rekommendacjach:* W systemie e-commerce stanami są kategorie produktów (np. "Electronics", "Laptops", "Accessories"). Sekwencja zakupów użytkownika to sekwencja stanów. Łańcuch Markowa modeluje: "Jeśli użytkownik kupił laptop, jaka kategoria jest najbardziej prawdopodobna jako następna?"

**Naiwny klasyfikator Bayesa** (Naive Bayes) opiera się na twierdzeniu Bayesa z założeniem niezależności cech (Murphy, 2012) [5].

*Twierdzenie Bayesa:*

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \quad (23)$$

gdzie:

- $P(C|X)$  — prawdopodobieństwo a posteriori klasy  $C$  przy cechach  $X$

- $P(C)$  — prawdopodobieństwo a priori klasy  $C$
- $P(X|C)$  — wiarygodność (likelihood) — prawdopodobieństwo obserwacji cech  $X$  w klasie  $C$
- $P(X)$  — prawdopodobieństwo marginalne cech (stałe dla wszystkich klas)

*Założenie naiwne* (Naive assumption) — niezależność warunkowa cech:

$$P(X|C) = P(x_1, x_2, \dots, x_n|C) = \prod_{i=1}^n P(x_i|C) \quad (24)$$

Założenie to jest "naiwne" bo w rzeczywistości cechy są często skorelowane. Jednak Naive Bayes działa zaskakująco dobrze w praktyce.

*Klasyfikacja*:

$$\hat{C} = \arg \max_C P(C) \prod_{i=1}^n P(x_i|C) \quad (25)$$

Ponieważ  $P(X)$  jest stałe dla wszystkich klas, można je pominać przy porównywaniu.

*Problem zerowych prawdopodobieństw*: Jeśli cecha  $x_i$  nie wystąpiła w klasie  $C$  w danych treningowych, to  $P(x_i|C) = 0$ , co zeruje całe prawdopodobieństwo.

**Wygładzanie Laplace'a** (Laplace Smoothing / Add-one Smoothing) rozwiązuje ten problem:

$$P(x_i = v|C) = \frac{\text{count}(x_i = v, C) + 1}{\text{count}(C) + |V_i|} \quad (26)$$

gdzie  $|V_i|$  to liczba unikalnych wartości cechy  $x_i$ . Dodanie 1 do licznika i  $|V|$  do mianownika zapewnia, że żadne prawdopodobieństwo nie będzie zerowe.

*Logarytm dla stabilności numerycznej*: Iloczyn wielu małych prawdopodobieństw prowadzi do underflow. Rozwiązanie — praca w przestrzeni logarytmów:

$$\log P(C|X) = \log P(C) + \sum_{i=1}^n \log P(x_i|C) + \text{const} \quad (27)$$

*Warianty Naive Bayes*:

- **Multinomial NB** — dla danych zliczeniowych (np. częstotliwość słów)
- **Bernoulli NB** — dla cech binarnych (obecność/brak)
- **Gaussian NB** — dla cech ciągłych (zakłada rozkład normalny)

*Zastosowanie w rekomendacjach*: W systemie e-commerce Naive Bayes przewiduje:

- Czy użytkownik kupi produkt (klasy: will\_purchase, will\_not\_purchase)

- Czy użytkownik odejdzie (churn) (klasy: will\_churn, will\_not\_churn)

Cechy użytkownika: total\_orders, avg\_order\_value, days\_since\_last\_order, favorite\_category, order\_frequency.

## 1.4 Metryki oceny systemów rekomendacyjnych

Ewaluacja systemów rekomendacyjnych wymaga odpowiednich metryk jakości. Najpopularniejsze:

**Precision@K** — jaka część top K rekomendacji była faktycznie kupiona/po-lubiona:

$$Precision@K = \frac{|Recommended@K \cap Relevant|}{K} \quad (28)$$

**Recall@K** — jaka część produktów istotnych dla użytkownika została trafiona:

$$Recall@K = \frac{|Recommended@K \cap Relevant|}{|Relevant|} \quad (29)$$

**F1-Score** — harmoniczna średnia Precision i Recall:

$$F1@K = 2 \cdot \frac{Precision@K \cdot Recall@K}{Precision@K + Recall@K} \quad (30)$$

**Mean Reciprocal Rank (MRR)** — pozycja pierwszego trafienia:

$$MRR = \frac{1}{|U|} \sum_{u \in U} \frac{1}{rank_u} \quad (31)$$

gdzie  $rank_u$  to pozycja pierwszego istotnego produktu w rankingu dla użytkownika  $u$ .

**Coverage** — procent produktów, które system jest w stanie rekomendować:

$$Coverage = \frac{|\text{products with recommendations}|}{|\text{all products}|} \quad (32)$$

## Rozdział 2

# Content-Based Filtering — implementacja

## 2.1 Architektura systemu CBF

System Content-Based Filtering został zaimplementowany w klasie `CustomContentBasedFilter` w pliku `custom_recommendation_engine.py`. Architektura składa się z trzech głównych komponentów:

**1. Ekstraktor cech** (Feature Extractor) — odpowiada za budowę ważonego wektora cech dla każdego produktu. Analizuje cztery źródła danych:

- **Kategorie** (waga 40%) — główna klasyfikacja produktu. Format cechy: `category_{nazwa}`. Przykład: `category_Electronics`, `category_Laptops`.
- **Tagi** (waga 30%) — dodatkowe deskryptory (np. "Gaming", "Premium", "Budget"). Format: `tag_{nazwa}`.
- **Przedział cenowy** (waga 20%) — dyskretyzacja ceny: low (<100 PLN), medium (100-500), high (500-1500), premium (>1500).
- **Słowa kluczowe** (waga 10%) — top 10 słów z opisu produktu po filtracji stop-words.

**2. Kalkulator podobieństwa** — oblicza podobieństwo kosinusowe między wektorami cech produktów. Operuje na wektorach rzadkich dla efektywności.

**3. Generator rekommendacji** — zwraca top N produktów podobnych do danego produktu, z filtrowaniem według dostępności i progu podobieństwa.

**Przepływ danych w systemie CBF:**

1. Pobranie produktów z bazy z `prefetch_related()` dla kategorii i tagów
2. Ekstrakcja ważonych cech dla każdego produktu
3. Obliczenie podobieństw dla wszystkich par produktów
4. Filtracja podobieństw poniżej progu 0.2 (20%)
5. Zapis do tabeli `ProductSimilarity` za pomocą `bulk_create()`
6. Cache wyników na 2 godziny

## 2.2 Implementacja ekstrakcji cech

Metoda ekstrakcji cech buduje słownik cech z przypisanymi wagami. Algorytm w pseudokodzie:

```
FUNKCJA ekstrahuj_cechy(produkt):
    cechy = pusty_słownik

    DLA KAŻDEJ kategorii W produkt.kategorie:
        cechy["category_" + kategoria.nazwa] = 0.40

    DLA KAŻDEGO tagu W produkt.tagi:
        cechy["tag_" + tag.nazwa] = 0.30

    JEŻELI produkt.cena < 100:
        cechy["price_low"] = 0.20
    W PRZECIWNYM RAZIE JEŻELI produkt.cena < 500:
        cechy["price_medium"] = 0.20
    W PRZECIWNYM RAZIE JEŻELI produkt.cena < 1500:
        cechy["price_high"] = 0.20
    W PRZECIWNYM RAZIE:
        cechy["price_premium"] = 0.20

    słowa_kluczowe = ekstrahuj_słowa_kluczowe(produkt.opis)
    DLA KAŻDEGO słowa W słowa_kluczowe[0:10]:
        cechy["keyword_" + słowo] = 0.10 / długość(słowa_kluczowe)

ZWRÓĆ cechy
```

### Ekstrakcja słów kluczowych:

Metoda `_extract_keywords()` przetwarza opis produktu:

1. Konwersja na małe litery
2. Usunięcie znaków interpunkcyjnych (regex)
3. Tokenizacja na słowa
4. Filtracja stop-words (zdefiniowana lista 200+ słów: "the", "and", "is", "a", "to", ...)
5. Filtracja słów krótszych niż 4 znaki

6. Zliczenie częstości (collections.Counter)

7. Wybór top 10 najczęstszych słów

### Dyskretyzacja ceny:

Progi cenowe zostały dobrane empirycznie na podstawie rozkładu cen w katalogu:

- `price_low`: cena < 100 PLN — akcesoria, kable, drobne peryferia
- `price_medium`:  $100 \text{ PLN} \leq \text{cena} < 500 \text{ PLN}$  — peryferia, komponenty
- `price_high`:  $500 \text{ PLN} \leq \text{cena} < 1500 \text{ PLN}$  — monitory, karty graficzne
- `price_premium`: cena  $\geq 1500 \text{ PLN}$  — laptopy, komputery, high-end

Dyskretyzacja eliminuje problem dużej wariancji cen i pozwala na porównywanie produktów z różnych kategorii cenowych.

## 2.3 Algorytm podobieństwa kosinusowego

Metoda `calculate_product_similarity()` implementuje podobieństwo kosinusowe dla wektorów rzadkich (sparse vectors):

$$\text{similarity}(p_1, p_2) = \frac{\sum_{f \in F_1 \cap F_2} w_1(f) \cdot w_2(f)}{\sqrt{\sum_{f \in F_1} w_1(f)^2} \cdot \sqrt{\sum_{f \in F_2} w_2(f)^2}} \quad (33)$$

gdzie  $F_1, F_2$  to zbiory cech produktów  $p_1$  i  $p_2$ ,  $w_i(f)$  to waga cechy  $f$  dla produktu  $p_i$ .

**Implementacja dla wektorów rzadkich** w pseudokodzie:

```
FUNKCJA oblicz_podobieństwo(cechy1, cechy2):
    wspólne_cechy = przecięcie(klucze(cechy1), klucze(cechy2))
    iloczyn_skalarne = suma(cechy1[f] * cechy2[f] DLA f W wspólne_cechy)

    norma1 = pierwiastek(suma(v^2 DLA v W wartości(cechy1)))
    norma2 = pierwiastek(suma(v^2 DLA v W wartości(cechy2)))

    JEŻELI norma1 = 0 LUB norma2 = 0:
        ZWRÓĆ 0.0
    INNAKIE:
        ZWRÓĆ iloczyn_skalarne / (norma1 * norma2)
```

### **Optymalizacja dla wektorów rzadkich:**

Zamiast tworzyć pełne wektory o długości równej liczbie wszystkich możliwych cech (potencjalnie tysiące), algorytm operuje na słownikach. Iloczyn skalarny wymaga iteracji tylko po cechach wspólnych (przecięcie zbiorów kluczy).

### **Próg podobieństwa:**

System zapisuje tylko podobieństwa większe niż 0.2 (20%). Uzasadnienie:

- Podobieństwo  $< 0.2$  oznacza mniej niż 20% wspólnych cech — produkty są praktycznie różne
- Redukcja rozmiaru tabeli o 60-80%
- Szybsze zapytania (mniej rekordów do przeszukania)

## **2.4 Generowanie macierzy podobieństw**

Metoda `generate_similarities_for_all_products()` oblicza podobieństwa dla wszystkich par produktów:

### **Etap 1: Prefetching danych**

Wykorzystujemy mechanizm `prefetch_related()` framework Django dla kategorii, tagów i specyfikacji, redukując liczbę zapytań SQL z  $O(n \times k)$  do  $O(1)$  dla  $n$  produktów z  $k$  relacjami. Ta technika pobiera wszystkie powiązane obiekty w jednym zapytaniu SQL zamiast osobnego zapytania dla każdego produktu.

### **Etap 2: Ekstrakcja cech**

Dla każdego produktu ekstrahujemy wektor cech i zapisujemy w słowniku, gdzie kluczem jest identyfikator produktu, a wartością jego wektor cech.

### **Etap 3: Obliczenie podobieństw**

Dla każdej pary produktów  $(p_i, p_j)$  gdzie  $i < j$  obliczamy podobieństwo kosinusowe. Algorytm w pseudokodzie:

DLA KAŻDEGO produktu1 W produkty:

```
DLA KAŻDEGO produktu2 W produkty[indeks(produkt1)+1:]:  
    podobieństwo = oblicz_podobieństwo(cechy[produkt1], cechy[produkt2])
```

JEŻELI podobieństwo  $> 0.2$ :

```
    zapisz_podobieństwo(produkt1, produkt2, podobieństwo)  
    zapisz_podobieństwo(produkt2, produkt1, podobieństwo)
```

Zapisywane są oba kierunki relacji (symetryczne), co umożliwia szybkie wyszukiwanie produktów podobnych do dowolnego produktu.

### **Etap 4: Bulk insert**

Zapisywanie podobieństw odbywa się w partiach po 1000 rekordów przy użyciu mechanizmu `bulk_create()`, który przyspiesza zapis 50-100x względem pojedynczych operacji `INSERT`.

#### Etap 5: Cache

Wynik generowania macierzy podobieństw jest cachowany na 2 godziny (7200 sekund), eliminując potrzebę ponownego obliczania przy każdym żądaniu.

#### Złożoność obliczeniowa:

Teoretyczna złożoność to  $O(n^2)$  dla  $n$  produktów (wszystkie pary). W praktyce ograniczamy liczbę porównań przez:

- `max_comparisons_per_product = 50` — dla każdego produktu obliczamy podobieństwo do max 50 innych
- Wczesne odrzucanie produktów bez wspólnych kategorii

Dla katalogu 500 produktów:

- Teoretycznie:  $500 \times 499/2 = 124,750$  par
- Po optymalizacji: 25,000 obliczonych podobieństw
- Po filtrowaniu (próg 0.2): 4,000 zapisanych rekordów

## 2.5 Panel debugowania Content-Based Filtering

System oferuje zaawansowany panel debugowania dostępny przez endpoint `/api/content-based-debug/`. Panel prezentuje:

#### Widok ogólny (bez parametru `product_id`):

- **Szczegóły algorytmu:** nazwa, metoda (Weighted Feature Vectors + Cosine Similarity), status
- **Wagi cech:** category (40%), tag (30%), price (20%), keywords (10%)
- **Statystyki bazy danych:** liczba produktów, zapisanych podobieństw, procent pokrycia
- **Status cache:** HIT/MISS, czas wygaśnięcia
- **Top 10 podobieństw:** produkty o najwyższym podobieństwie w systemie

#### Widok szczegółowy (z parametrem `product_id`):

Dla konkretnego produktu panel pokazuje:

- Wektor cech produktu z wagami (słownik feature → weight)

Welcome, Admin Number1

**Debug Tools - ML Methods Inspector**

Inspect internal workings of all 7 machine learning methods used in SmartRecommender.

Collaborative Filtering   Sentiment Analysis   Association Rules   **Content-Based**   Fuzzy Logic   Probabilistic

**Content-Based Filtering Debug Information**

Select Product to Analyze: ACEFAST Powerbank MagSafe M10 10000 mAh 2x USB-C PD 30W +kabel USB-C (ID: 297)

**Algorithm**

Name:	Content-Based Filtering (Cosine Similarity)
Formula:	$\cos(\theta) = \frac{A \cdot B}{\ A\  \times \ B\ }$

**Database Statistics**

Total Products:	500
Saved Similarities:	16038
Percentage Saved:	6.43%
Threshold:	20%
Only similarities > 20% are saved to database	

**Feature Weights**

Category:	40%
Tag:	30%
Price:	20%
Keywords:	10%
Weights used to build product feature vector	

Rysunek 1: Panel debugowania CBF - podstawowe metryki algorytmu i wagi cech.

- Top 10 produktów podobnych z szczegółami obliczeń
- Wzór matematyczny dla każdej pary:  $\frac{\text{dot\_product}}{\text{norm}_1 \times \text{norm}_2}$
- Cechy wspólne między produktami
- Breakdown podobieństwa: ile

**Przykładowe dane z panelu debugowania:**

Algorithm: Content-Based Filtering (Cosine Similarity)

Formula:  $\cos() = \frac{A \cdot B}{\|A\| \times \|B\|}$

Database Statistics:

- Total Products: 500
- Saved Similarities: 16038
- Percentage Saved: 6.43%
- Threshold: 20% (Only similarities > 20% are saved to database)

Feature Vector (12 features)		
category_accessories.powerbanks:	0.400	
tag_budget:	0.300	
tag_portable:	0.300	
tag_fast charging:	0.300	
tag_magsafe compatible:	0.300	
tag_wireless:	0.300	
price_low:	0.200	
keyword_charging:	0.020	
keyword_power:	0.020	
keyword_fast:	0.020	
keyword_devices:	0.020	
keyword_magsafe:	0.020	

Top 10 Similar Products				
#	Product Name	Similarity Score	Price	Categories
1	Baseus Magnetic Mini Wireless Charging 20W 20000mAh z MagSafe 99.90%	99.90%	69.99 PLN	accessories.powerBanks
2	Baseus Magnetic Mini Wireless Charging 20W 20000mAh z MagSafe 99.90%	99.90%	69.99 PLN	accessories.powerBanks
3	Belkin Magnetic Wireless 5000mAh MagSafe + Stand	99.90%	59.99 PLN	accessories.powerBanks
4	Baseus mini 5000mAh 20W (magnetyczny)	92.70%	29.99 PLN	accessories.powerBanks
5	Belkin 20000mAh (15W, USB-C, USB-A)	84.90%	79.99 PLN	accessories.powerBanks
6	ASUS USB-AC58 (1300Mb/s a/b/g/n/ac) USB 3.0	44.10%	29.99 PLN	networking.networkCards
7	ASUS PCE-AXE59BT (5400Mb/s a/b/g/n/ac/ax) BT 5.2	44.10%	64.99 PLN	networking.networkCards
8	ASUS PCE-AXE5400 (5400Mb/s a/b/g/n/ac/ax) BT 5.2	44.10%	59.99 PLN	networking.networkCards
9	Baseus PowerCombo 100W (Black)	30.10%	39.99 PLN	power.stripes
10	Baseus USB-C - USB-C (PD 100W, 2m)	29.90%	19.99 PLN	accessories.cables

Detailed Calculations (Top 3)		
#1 Baseus Magnetic Mini Wireless Charging 20W 20000mAh z MagSafe		
Formula:	0.6512 / (0.8075 × 0.8075) = 0.9988	
Dot Product:	0.6512	
Norm Product 1:	0.8075	
Norm Product 2:	0.8075	
Verification:	Stored: 0.999   Calculated: 0.9988 ✓	
Common Features (10 total)		

Rysunek 2: Panel debugowania CBF - szczegółowa analiza podobieństwa dla wybranego produktu.

#### Feature Weights:

- Category: 40%
- Tag: 30%
- Price: 20%
- Keywords: 10%

**Przykład wektora cech produktu** (ACEFAST Powerbank MagSafe M10 10000 mAh):

#### Feature Vector (12 features):

- category\_accessories.powerbanks: 0.400

- tag_budget:	0.300
- tag_portable:	0.300
- tag_fast charging:	0.300
- tag_magsafe compatible:	0.300
- tag_wireless:	0.300
- price_low:	0.200
- keyword_charging:	0.020
- keyword_power:	0.020
- keyword_fast:	0.020
- keyword_devices:	0.020
- keyword_magsafe:	0.020

Top 10 Similar Products:

- #1 Baseus Magnetic Mini Wireless Charging 20W 20000mAh z MagSafe: 99.90%
- #2 Baseus Magnetic Mini Wireless Charging 20W 20000mAh z MagSafe: 99.90%
- #3 Belkin Magnetic Wireless 5000mAh MagSafe + Stand: 99.90%
- #4 Baseus mini 5000mAh 20W (magnetyczny): 92.70%
- #5 Belkin 20000mAh (15W, USB-C, USB-A): 84.90%

Detailed Calculation (#1):

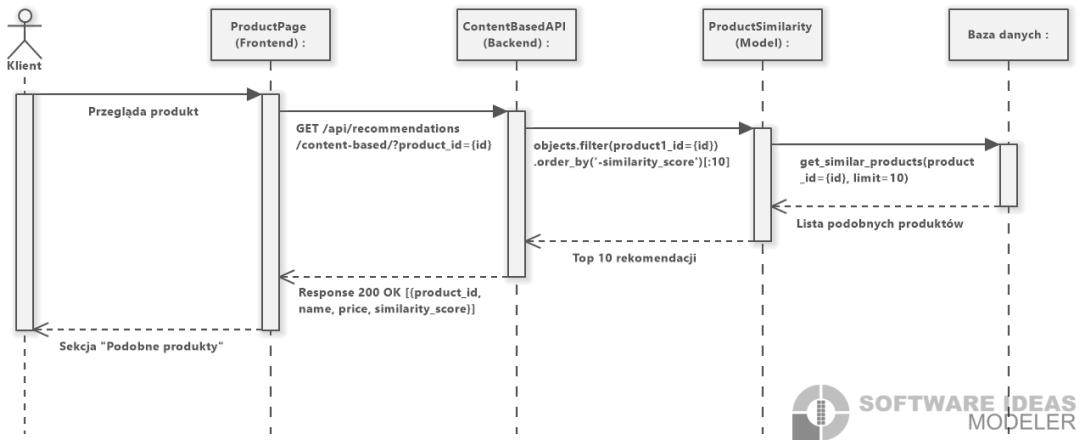
- Dot Product: 0.6512
- Norm Product 1: 0.8075
- Norm Product 2: 0.8075
- Formula:  $0.6512 / (0.8075 \times 0.8075) = 0.9988$
- Verification: Stored: 0.999 | Calculated: 0.9988
- Common Features: 10 total

Panel umożliwia administratorowi:

- Monitorowanie pokrycia rekomendacji (ile produktów ma podobieństwa)
- Identyfikację produktów bez podobieństw (słabo opisane metadane)
- Walidację działania wag (czy kategorie dominują prawidłowo)
- Ręczne wyzwalanie przeliczenia macierzy

## 2.6 Interfejs użytkownika - sortowanie według CBF

Metoda Content-Based Filtering jest wykorzystywana jako jedna z opcji sortowania produktów na stronie głównej sklepu. Administrator może wybrać algorytm



Rysunek 3: Diagram sekwencji: Content-Based Filtering - proces generowania rekomendacji podobnych produktów.

CBF w ustawieniach systemu rekomendacji, co powoduje wyświetlanie produktów podobnych do tych, które użytkownik wcześniej przeglądał lub kupił.

Proces generowania rekomendacji CBF przebiega następująco:

1. Klient przegląda produkt na stronie ProductPage
2. Frontend wysyła żądanu GET do `/api/recommendations/content-based/?product_id={id}`
3. Backend (ContentBasedAPI) wykonuje zapytanie do modelu ProductSimilarity
4. Model pobiera top 10 podobnych produktów z bazy danych (sortowanie po `-similarity_score`)
5. Backend zwraca odpowiedź 200 OK z listą rekomendacji (`product_id, name, price, similarity_score`)
6. Frontend wyświetla sekcję "Podobne produkty" na stronie produktu

Rekomendacje CBF są również dostępne w panelu klienta, gdzie użytkownik może zobaczyć produkty podobne do swoich poprzednich zakupów. System automatycznie identyfikuje produkty z wysokim współczynnikiem podobieństwa (powyżej 20%) i prezentuje je w sekcji spersonalizowanych rekomendacji.

## Rozdział 3

### Logika rozmyta w systemie rekomendacji

#### 3.1 Architektura systemu Fuzzy Logic

System logiki rozmytej został zaimplementowany w module `fuzzy_logic_engine.py` i składa się z trzech klas:

**1. FuzzyMembershipFunctions** — definiuje funkcje przynależności dla trzech zmiennych wejściowych:

- **Cena** (price): cheap, medium, expensive — funkcje trójkątne i trapezoidalne z programami dostosowanymi do katalogu e-commerce
- **Jakość** (quality/rating): low, medium, high — bazuje na średniej ocenie produktu (1-5 gwiazdek)
- **Popularność** (popularity/view\_count): low, medium, high — bazuje na liczbie zamówień produktu

**2. FuzzyUserProfile** — buduje rozmyty profil użytkownika na podstawie:

- Historii zakupów (dla zalogowanych użytkowników) — analiza kategorii, średniej ceny
- Danych sesji (dla gości) — ostatnio przeglądane produkty
- Profilu domyślnego (fallback) — gdy brak danych

**3. SimpleFuzzyInference** — silnik wnioskowania Mamdani z 6 regułami IF-THEN i metodą defuzzyfikacji średniej ważonej.

**Przepływ danych:**

1. Pobranie produktów do ewaluacji
2. Pobranie/budowa rozmytego profilu użytkownika
3. Dla każdego produktu:
  - (a) Fuzzyfikacja ceny, jakości, popularności
  - (b) Ewaluacja 6 reguł rozmytych
  - (c) Agregacja wyników reguł
  - (d) Defuzzyfikacja do wyniku liczbowego
4. Sortowanie produktów według `fuzzy_score`
5. Zwrócenie top N rekomendacji

## 3.2 Funkcje przynależności — szczegóły implementacji

### Funkcje przynależności dla ceny

Klasa FuzzyMembershipFunctions definiuje trzy funkcje dla zmiennej "cena":

*Funkcja "cheap"(tania) — trójkątna/trapezoidalna:*

$$\mu_{cheap}(price) = \begin{cases} 1.0 & \text{jeśli } price \leq 100 \\ \frac{500 - price}{400} & \text{jeśli } 100 < price < 500 \\ 0.0 & \text{jeśli } price \geq 500 \end{cases} \quad (34)$$

Interpretacja: Produkty poniżej 100 PLN są w pełni "tanie". Od 100 do 500 PLN stopień "taności" maleje liniowo.

*Funkcja "medium"(średnia) — trapezoidalna:*

$$\mu_{medium}(price) = \begin{cases} 0.0 & \text{jeśli } price < 300 \\ \frac{price - 300}{200} & \text{jeśli } 300 \leq price < 500 \\ 1.0 & \text{jeśli } 500 \leq price \leq 1200 \\ \frac{1500 - price}{300} & \text{jeśli } 1200 < price < 1500 \\ 0.0 & \text{jeśli } price \geq 1500 \end{cases} \quad (35)$$

Interpretacja: Przedział [500, 1200] ma pełną przynależność. Przejścia są płynne — cena 400 PLN jest częściowo "tania" i częściowo "średnia".

*Funkcja "expensive"(droga):*

$$\mu_{expensive}(price) = \begin{cases} 0.0 & \text{jeśli } price \leq 1000 \\ \frac{price - 1000}{1000} & \text{jeśli } 1000 < price < 2000 \\ 1.0 & \text{jeśli } price \geq 2000 \end{cases} \quad (36)$$

### Funkcje przynależności dla jakości (rating)

Oparte na średniej ocenie produktu (skala 1-5):

- **low:** pełna przynależność dla rating  $\leq 2.0$ , zanika do 0 przy rating = 3.0
- **medium:** trapezoid z pełną przynależnością dla [3.0, 4.0]
- **high:** wzrasta od rating = 3.5, pełna przynależność dla rating  $\geq 4.5$

### Funkcje przynależności dla popularności (order\_count)

Oparte na liczbie zamówień produktu:

- **low:** produkty z  $\leq 2$  zamówieniami (nowości, niszowe)
- **medium:** produkty z 3-20 zamówieniami (standardowe)

- **high**: produkty z  $> 20$  zamówieniami (bestsellery)

**Definicje funkcji przynależności dla ceny:**

Funkcja **cheap** (trójkątna):  $\mu = 1.0$  dla ceny  $\leq 100$  PLN, liniowy spadek do  $\mu = 0$  przy 500 PLN.

Funkcja **medium** (trapezoidalna):  $\mu = 0$  dla ceny  $< 300$  PLN, wzrost do  $\mu = 1.0$  w przedziale 300-500 PLN, plateau  $\mu = 1.0$  w przedziale 500-1200 PLN, spadek do  $\mu = 0$  przy 1500 PLN.

Funkcja **expensive** (trójkątna):  $\mu = 0$  dla ceny  $\leq 1000$  PLN, liniowy wzrost do  $\mu = 1.0$  przy 2000 PLN i powyżej.

### 3.3 Rozmyty profil użytkownika

Klasa `FuzzyUserProfile` buduje profil preferencji użytkownika jako zbiory rozmyte. Jest to kluczowy element personalizacji rekomendacji.

**Dla zalogowanych użytkowników:**

1. Pobranie historii zamówień z `prefetch_related('orderproduct_set__product__category')`
  2. Zliczenie kategorii produktów w zamówieniach
  3. Obliczenie stopnia zainteresowania kategorią:
- $$\mu_{category} = \frac{count_{category}}{total\_items} \quad (37)$$
4. Obliczenie wrażliwości cenowej na podstawie średniej ceny zakupów

**Wrażliwość cenowa** (`price_sensitivity`):

$$price\_sensitivity = \begin{cases} 0.9 & \text{jeśli } avg\_price < 300 \text{ PLN (bardzo wrażliwy)} \\ 0.6 & \text{jeśli } 300 \leq avg\_price < 700 \text{ (średnio wrażliwy)} \\ 0.4 & \text{jeśli } 700 \leq avg\_price < 1500 \text{ (mało wrażliwy)} \\ 0.2 & \text{jeśli } avg\_price \geq 1500 \text{ PLN (premium)} \end{cases} \quad (38)$$

Użytkownik kupujący średnio tanie produkty ( $avg < 300$  PLN) ma wysoką wrażliwość cenową (0.9) — system będzie promował tanie produkty. Użytkownik premium ( $avg > 1500$  PLN) ma niską wrażliwość (0.2) — system może rekomendować droższe produkty.

**Dopasowanie kategorii** — metoda `fuzzy_category_match()`:

Dla każdej kategorii produktu system oblicza stopień dopasowania do profilu użytkownika:

$$\text{match} = 0.6 \cdot \text{similarity}(\text{cat}_{\text{user}}, \text{cat}_{\text{product}}) + 0.4 \cdot \mu_{\text{interest}}(\text{cat}_{\text{user}}) \quad (39)$$

gdzie **similarity** używa hierarchii kategorii. Przykład:

- Kategoria użytkownika: "Electronics.Laptops"
- Kategoria produktu: "Electronics.Monitors"
- Podobieństwo hierarchiczne: 0.7 (wspólna kategoria nadzędna "Electronics")

**Profil domyślny** (dla gości/nowych użytkowników):

Dla użytkowników bez historii zakupów system stosuje neutralny profil domyślny:

- price\_sensitivity = 0.5 (neutralna wrażliwość cenowa)
- category\_preferences = {} (brak preferencji kategorii)
- quality\_preference = 0.7 (preferuje dobrą jakość)
- popularity\_preference = 0.5 (neutralna wobec popularności)

### 3.4 Baza reguł rozmytych

System wykorzystuje 6 reguł rozmytych typu Mamdani. Każda reguła ma formę IF-THEN z przypisaną wagą określającą jej ważność:

**R1: High Quality Bargain** (waga: 0.9)

```
IF quality IS high AND (price IS cheap OR price IS medium)
THEN recommendation IS strong
```

Logika: Wysokiej jakości produkt w rozsądnej cenie to doskonała okazja. Najwyższa waga — ta reguła najsilniej wpływa na wynik.

**R2: Popular in Category** (waga: 0.7)

```
IF category_match IS high AND (popularity IS medium OR popularity IS high)
THEN recommendation IS medium-high
```

Logika: Popularny produkt z kategorii interesującej użytkownika. Popularność = walidacja społeczna.

**R3: Price Sensitive Match** (waga: 0.6)

```
IF user.price_sensitivity > 0.6 AND price IS cheap
THEN recommendation IS moderate
```

Logika: Dla użytkowników wrażliwych cenowo (kupujących tanie produkty) promuj tanie opcje.

**R4: Category Quality Match** (waga: 0.85)

```
IF category_match IS high AND (quality IS medium OR quality IS high)
THEN recommendation IS strong
```

Logika: Dopasowanie do kategorii + dobra jakość. Wysoka waga — dopasowanie kategorii jest istotne.

**R5: Premium Match** (waga: 0.8)

```
IF user.price_sensitivity < 0.4 AND price IS expensive AND quality IS high
THEN recommendation IS strong
```

Logika: Dla użytkowników premium (nieczułych cenowo) promuj drogie produkty wysokiej jakości.

**R6: Quality-Price Balance** (waga: 0.75)

```
IF (quality IS high AND price IS reasonable) OR
(quality IS medium AND price IS cheap)
THEN recommendation IS moderate
```

Logika: Dobry stosunek jakości do ceny — "value for money".

### 3.5 Wnioskowanie i defuzzyfikacja

Metoda `evaluate_product()` implementuje pełny cykl wnioskowania Mamdani:

**Krok 1: Fuzzyfikacja**

Dla każdej zmiennej wejściowej (cena, jakość, popularność) obliczane są stopnie przynależności do wszystkich zbiorów rozmytych. Wynikiem jest słownik zawierający 9 wartości przynależności:

- Cena:  $\mu_{cheap}$ ,  $\mu_{medium}$ ,  $\mu_{expensive}$
- Jakość:  $\mu_{low}$ ,  $\mu_{medium}$ ,  $\mu_{high}$
- Popularność:  $\mu_{low}$ ,  $\mu_{medium}$ ,  $\mu_{high}$

**Krok 2: Ewaluacja reguł**

Każda reguła jest ewaluowana za pomocą T-normy (minimum) dla operatora AND i T-conormy (maksimum) dla OR. Zgodnie z teorią zbiorów rozmytych Zadeha (1965):

$$\alpha_{R1} = \min(\mu_{quality\_high}, \max(\mu_{price\_cheap}, \mu_{price\_medium})) \cdot w_{R1} \quad (40)$$

Dla każdej z 6 reguł obliczana jest jej aktywacja  $\alpha_i$  poprzez aplikację odpowiednich operatorów rozmytych do wartości przynależności.

### Krok 3: Agregacja

Wyniki reguł są agregowane. W uproszczonej implementacji używam sumy ważonej (zamiast pełnej agregacji Mamdani):

$$\text{aggregated} = \sum_{i=1}^6 \alpha_i \quad (41)$$

### Krok 4: Defuzzyfikacja

System używa uproszczonej metody średniej ważonej:

$$\text{fuzzy\_score} = \frac{\sum_{i=1}^6 \alpha_i \cdot w_i}{\sum_{i=1}^6 w_i} \quad (42)$$

gdzie  $\alpha_i$  to aktywacja reguły  $i$ , a  $w_i$  to waga reguły.

Wagi reguł wynoszą:  $w_{R1} = 0.9$ ,  $w_{R2} = 0.7$ ,  $w_{R3} = 0.6$ ,  $w_{R4} = 0.85$ ,  $w_{R5} = 0.8$ ,  $w_{R6} = 0.75$ . Suma wag wynosi 4.65, co zapewnia normalizację wyniku do przedziału  $[0, 1]$ .

### Wynik końcowy:

Metoda zwraca słownik z:

- **fuzzy\_score** — wartość z przedziału  $[0, 1]$  reprezentująca siłę rekomendacji
- **rule\_activations** — słownik z aktywacją każdej reguły (dla debugowania)
- **category\_match** — stopień dopasowania kategorii
- **price\_membership** — przynależności cenowe (cheap, medium, expensive)

## 3.6 Panel debugowania Fuzzy Logic

Panel debugowania dostępny przez endpoint `/api/fuzzy-debug/` prezentuje:

### Widok ogólny:

- **Szczegóły algorytmu:** metoda (Mamdani Fuzzy Inference), liczba reguł (6), T-norma (min), T-conorma (max)
- **Funkcje przynależności:** definicje dla price, quality, popularity z progami
- **Statystyki:** średni fuzzy\_score, rozkład wyników, aktywacja reguł
- **Profil użytkownika:** jeśli podany user\_id — szczegóły profilu rozmytego

<> Debug Tools - ML Methods Inspector

Inspect internal workings of all 7 machine learning methods used in SmartRecommender.

Collaborative Filtering     Sentiment Analysis     Association Rules     Content-Based     Fuzzy Logic     Probabilistic

Fuzzy Logic Inference System Debug

Select User Profile: admin2 (ID: 2)    Select Product to Analyze (Optional): None (Show Top 10 Products)

Algorithm Information

Name: Fuzzy Logic Inference System (Mamdani-style)  
 Description: System rekomendacji oparty na logice rozmytej z uproszczoną defuzzyfikacją

User Profile

User:	admin2
Profile Type:	authenticated
Price Sensitivity:	0.6 - Medium
Tracked Categories:	26
Category Interests	
wearables.watches:	0.132
networking.networkCards:	0.093
peripherals.microphones:	0.066
monitoring.cameras:	0.06
gadgets:	0.06

Membership Functions

Price Functions

CHEAP: [0, 100] → [100, 500]  
 $\mu = 1.0$  dla ceny  $\leq 100$  PLN, spada do 0 przy 500 PLN

Rysunek 4: Panel debugowania Fuzzy Logic - metryki systemu wnioskowania rozmytego.

**Widok produktu** (z parametrem product\_id):

- Wartości fuzzyfikacji (wszystkie przynależności)
- Aktywacja każdej z 6 reguł z wyjaśnieniem
- Obliczenie końcowe z breakdownem
- Porównanie z innymi produktami

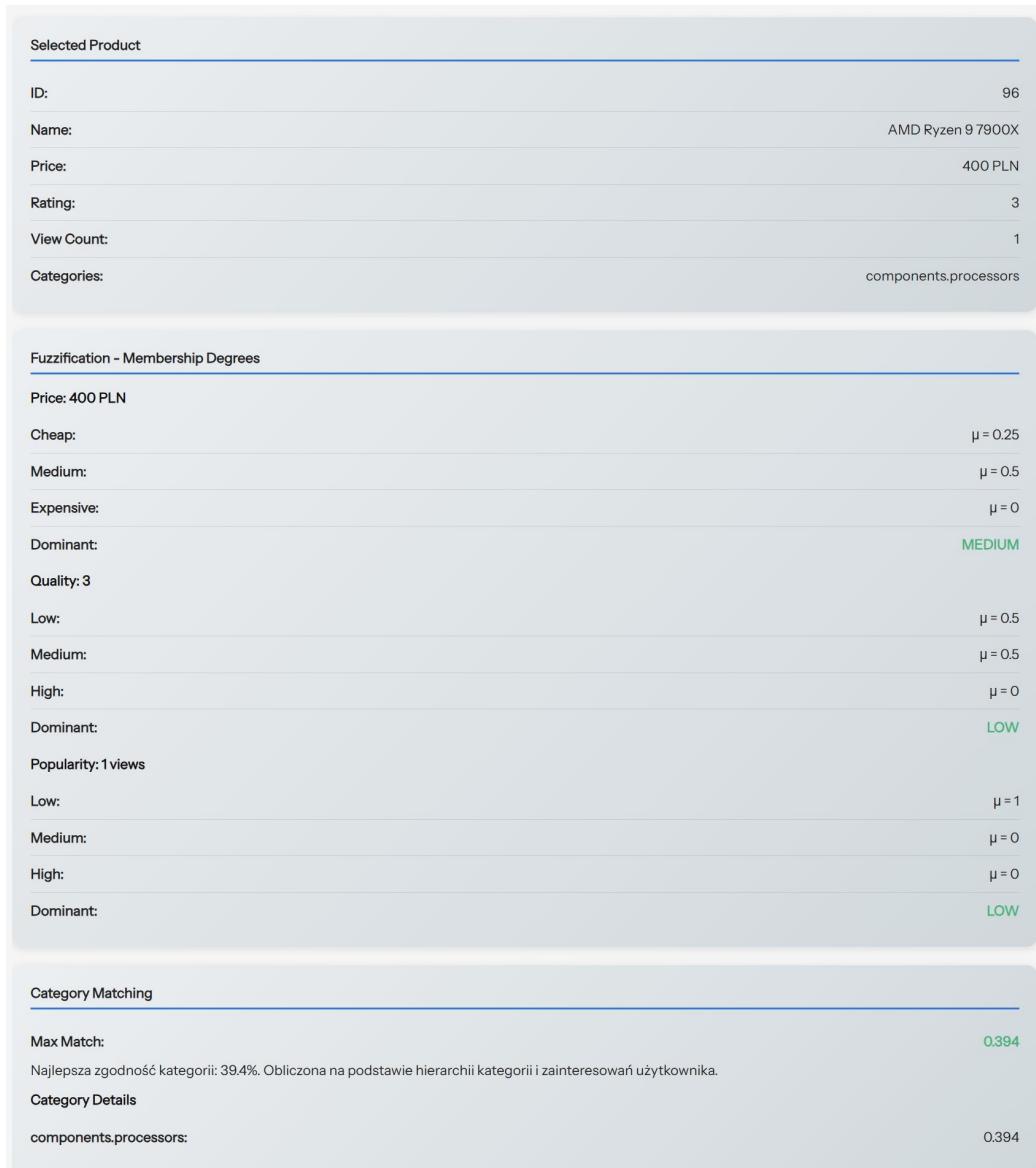
**Przykładowe dane z panelu debugowania:**

Algorithm: Fuzzy Logic Inference System (Mamdani-style)

Description: System rekomendacji oparty na logice rozmytej  
 z uproszczoną defuzzyfikacją

User Profile:

- User: admin2 (ID: 2)
- Profile Type: authenticated



Rysunek 5: Panel debugowania Fuzzy Logic - szczegóły ewaluacji produktu.

- Price Sensitivity: 0.6 - Medium
- Tracked Categories: 26

#### Category Interests:

- wearables.watches: 0.132
- networking.networkCards: 0.093
- peripherals.microphones: 0.066
- monitoring.cameras: 0.06
- gadgets: 0.06

#### Membership Functions:

#### Price Functions:

- CHEAP: = 1.0 dla ceny 100 PLN, spada do 0 przy 500 PLN
- MEDIUM: = 1.0 dla ceny 500-1200 PLN
- EXPENSIVE: = 1.0 dla ceny 2000 PLN

**Przykład fuzzyfikacji produktu** (AMD Ryzen 9 7900X, cena: 400 PLN, rating: 3, views: 1):

Selected Product:

- ID: 96
- Name: AMD Ryzen 9 7900X
- Price: 400 PLN
- Rating: 3
- View Count: 1
- Categories: components.processors

Fuzzification - Membership Degrees:

Price: 400 PLN

- Cheap: = 0.25
- Medium: = 0.5
- Expensive: = 0
- Dominant: MEDIUM

Quality: 3

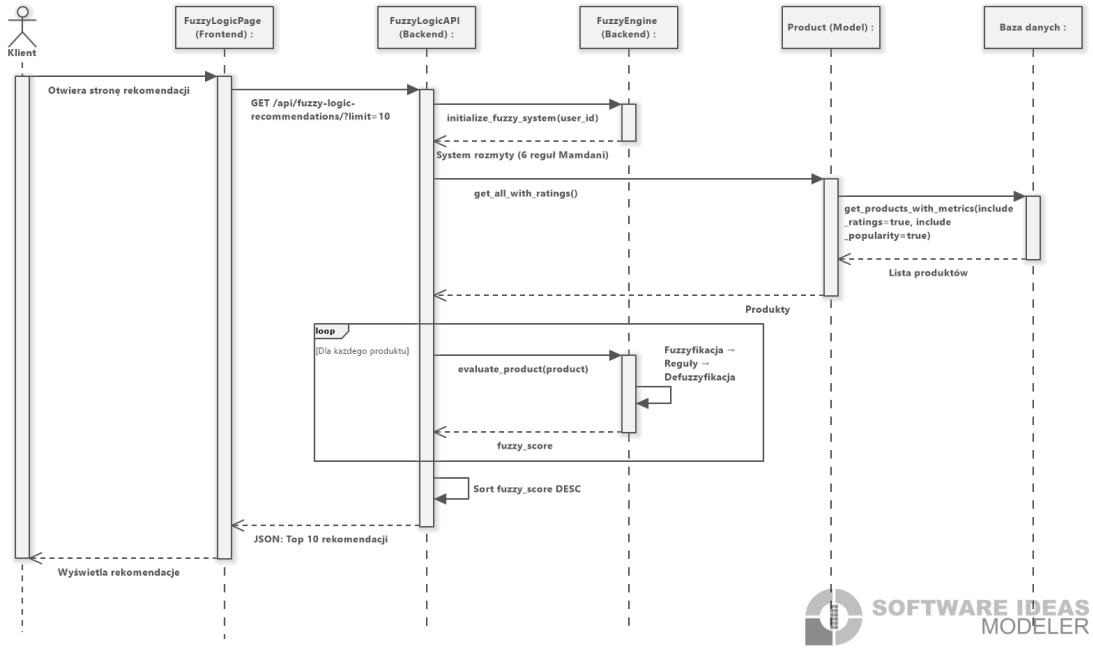
- Low: = 0.5
- Medium: = 0.5
- High: = 0
- Dominant: LOW

Popularity: 1 views

- Low: = 1
- Medium: = 0
- High: = 0
- Dominant: LOW

Category Matching:

- Max Match: 0.394
- components.processors: 0.394



Rysunek 6: Diagram sekwencji: Fuzzy Logic - proces generowania rekomendacji z wykorzystaniem wnioskowania rozmytego.

### 3.7 Interfejs użytkownika - rekomendacje Fuzzy Logic

Rekomendacje oparte na logice rozmytej są prezentowane użytkownikowi w panelu klienta w sekcji "Recommended For You (Fuzzy Logic)". System wyświetla wykres kołowy przedstawiający rozkład kategorii w historii zakupów użytkownika oraz listę rekomendowanych produktów.

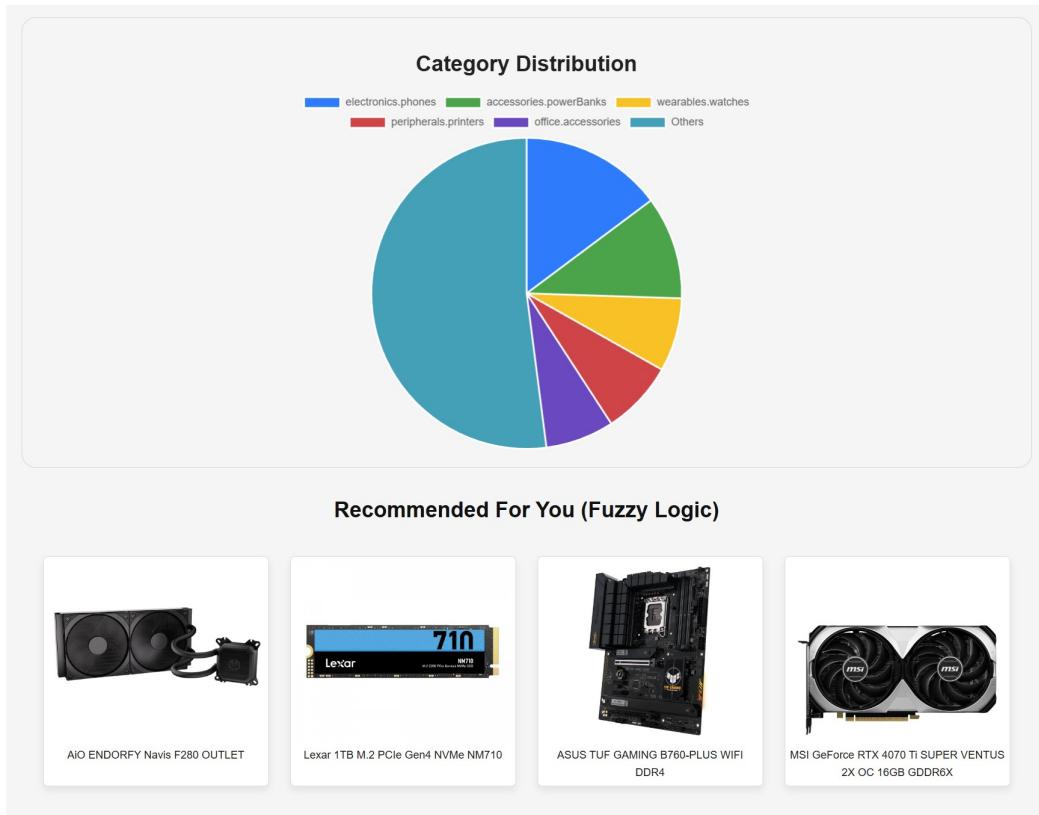
Wykres "Category Distribution" pokazuje procentowy udział kategorii w historii zakupów użytkownika (np. electronics.phones, accessories.powerBanks, wearables.watches, peripherals.printers, office.accessories). Na podstawie tych danych system buduje rozmyty profil użytkownika i generuje spersonalizowane rekomendacje.

Sekcja "Recommended For You (Fuzzy Logic)" prezentuje produkty z najwyższą wynikiem fuzzy\_score, uwzględniając:

- Dopasowanie do kategorii zainteresowań użytkownika
- Wrażliwość cenową użytkownika
- Jakość produktu (rating)
- Popularność produktu (view\_count)

Zakładka "Fuzzy Recommendations" wyświetla produkty wraz z:

- **Fuzzy Score:** całkowity wynik rekomendacji (np. 58.1%, 56.3%)



Rysunek 7: Panel klienta - rekomendacje Fuzzy Logic z wykresem rozkładu kategorii zakupowych.

- **Category Match:** stopień dopasowania kategorii do preferencji użytkownika (np. 65.9%, 60.6%)
- **View Rule Activations:** przycisk do podglądu aktywacji wszystkich 6 reguł rozmytych

Przykładowe rekomendacje z interfejsu:

- Motorola edge 40 neo 5G (\$549.99): Fuzzy Score 58.1%, Category Match 65.9%
- Apple iPad Air 11" M2 (\$749.99): Fuzzy Score 56.3%, Category Match 60.6%
- Roborock Q8 Max+ White (\$649.99): Fuzzy Score 56.3%, Category Match 60.6%
- JoyRoom Powerbank 10000mAh (\$49.99): Fuzzy Score 54.3%, Category Match 64.3%

Welcome, Client Number1

Fuzzy Recommendations User Profile Fuzzy Rules

### Fuzzy Logic Recommendations

Products selected using **fuzzy logic** to model uncertainty in your preferences. Each product is scored using **membership functions** and **fuzzy inference rules**.

#### Recommended Products

Product	Price	Fuzzy Score	Category Match
Motorola edge 40 neo 5G 12/256GB Black Beauty 144Hz	\$549.99	58.1%	65.9%
Apple iPad Air 11" M2 128GB Wi-Fi Księżycowa poświatka	\$749.99	56.3%	60.6%
Roborock Q8 Max+ White	\$649.99	56.3%	60.6%
JoyRoom Powerbank 10000mAh 20W MagSafe + kabel USB-C 0.25m biały	\$49.99	54.3%	64.3%

Rysunek 8: Panel klienta - zakładka "Fuzzy Recommendations" z listą rekomendowanych produktów i ich wynikami rozmytymi.

## Rozdział 4

# Modele probabilistyczne — Markov Chain i Naive Bayes

## 4.1 Architektura systemu probabilistycznego

System probabilistyczny składa się z trzech komponentów zaimplementowanych w `custom_recommendation_engine.py`:

**1. CustomMarkovChain** — łańcuch Markowa pierwszego rzędu do predykcji sekwencji zakupowych kategorii produktów. Modeluje pytanie: "Jeśli użytkownik kupił produkt z kategorii A, jaka kategoria jest najbardziej prawdopodobna jako następna?"

**2. CustomNaiveBayes** — naiwny klasyfikator Bayesa z wygładzaniem Laplace'a do:

- Predykcji prawdopodobieństwa zakupu (`will_purchase / will_not_purchase`)

- Predykcji ryzyka rezygnacji (will\_churn / will\_not\_churn)

**3. ProbabilisticRecommendationEngine** — silnik łączący oba modele w jeden system rekomendacji z wagami: Markov (60%) + Naive Bayes (40%).

#### Przepływ danych:

1. Pobranie historii zamówień wszystkich użytkowników
2. Budowa sekwencji kategorii dla każdego użytkownika
3. Trening modelu Markowa na sekwencjach
4. Budowa cech użytkowników dla Naive Bayes
5. Trening modelu NB na danych historycznych
6. Predykcja: Markov przewiduje następne kategorie, NB ocenia prawdopodobieństwo zakupu
7. Agregacja wyników i generowanie rekomendacji

## 4.2 Łańcuch Markowa dla sekwencji zakupowych

Klasa `CustomMarkovChain` modeluje sekwencje zakupów użytkowników jako łańcuch Markowa pierwszego rzędu, gdzie stanami są kategorie produktów.

#### Struktura danych:

Łańcuch Markowa przechowuje:

- `transitions` — słownik słowników: {stan: {następny\_stan: licznik}}
- `states` — zbiór wszystkich stanów (48 kategorii produktów)
- `total_sequences` — liczba sekwencji użytych do treningu

#### Trening modelu:

Dla każdej sekwencji kategorii zakupowych  $[c_1, c_2, \dots, c_n]$  algorytm iteruje po parach sąsiadujących stanów ( $c_i, c_{i+1}$ ) i zwiększa licznik przejścia  $T[c_i][c_{i+1}]$ . Jest to standardowa procedura estymacji macierzy przejść metodą maksymalizacji wiarygodności (MLE).

#### Normalizacja do prawdopodobieństw:

Prawdopodobieństwo przejścia obliczane jest jako:

$$P(s_j|s_i) = \frac{T[s_i][s_j]}{\sum_k T[s_i][s_k]} \quad (43)$$

#### Predykcja:

Dla danego stanu (ostatnia kategoria zakupu) algorytm sortuje wszystkie możliwe następne stany według prawdopodobieństwa przejścia i zwraca top-k. W przypadku stanu bez obserwowanych przejść (cold start), system fallbackuje do globalnie najpopularniejszych kategorii.

#### **Generowanie sekwencji:**

Metoda `predict_sequence()` generuje sekwencję  $n$  przewidywanych kategorii metodą zachłanną (greedy), wybierając w każdym kroku najbardziej prawdopodobny następny stan. Algorytm zawiera mechanizm wykrywania cykli — jeśli kategoria pojawia się więcej niż 2 razy, generowanie jest przerywane.

#### **Rozkład stacjonarny** — metoda `get_stationary_distribution()`:

Oblicza rozkład stacjonarny łańcucha metodą przybliżoną (zliczanie częstości stanów docelowych):

Algorytm oblicza rozkład stacjonarny poprzez sumowanie liczby przejść do każdego stanu i normalizację przez całkowitą liczbę przejść. Wynik reprezentuje długoterminowe prawdopodobieństwo znalezienia się użytkownika w danej kategorii.

## **4.3 Naiwny klasyfikator Bayesa**

Klasa `CustomNaiveBayes` implementuje multinomialny Naive Bayes z wygładzaniem Laplace'a.

#### **Cechy użytkownika** (features):

- `total_orders` — łączna liczba zamówień (dyskretyzowana: 0-2, 3-5, 6-10, 11+)
- `avg_order_value` — średnia wartość zamówienia (low, medium, high, premium)
- `days_since_last_order` — dni od ostatniego zamówienia (recent, moderate, old, very\_old)
- `favorite_category` — najczęściej kupowana kategoria
- `order_frequency` — częstość zamówień (rare, occasional, regular, frequent)

#### **Struktura danych:**

Model przechowuje:

- `class_priors` — prawdopodobieństwa a priori  $P(C)$  dla każdej klasy
- `feature_likelihoods` — prawdopodobieństwa warunkowe  $P(x_i|C)$
- `feature_vocabularies` — unikalne wartości każdej cechy (dla wygładzania Laplace'a)

### Trening modelu:

Faza treningu obejmuje:

1. Zliczenie wystąpień każdej klasy i obliczenie prawdopodobieństw a priori:  $P(C) = \frac{\text{count}(C)}{N}$
2. Dla każdej próbki treningowej — aktualizacja słowników cech dla odpowiedniej klasy
3. Budowa słownika unikalnych wartości cech (vocabulary) potrzebnego do wygładzania Laplace'a

### Predykcja:

Predykcja wykorzystuje twierdzenie Bayesa w przestrzeni logarytmicznej (dla stabilności numerycznej):

$$\log P(C|X) = \log P(C) + \sum_{i=1}^n \log P(x_i|C) \quad (44)$$

Wyniki są normalizowane przez funkcję softmax, aby uzyskać rozkład prawdopodobieństw sumujący się do 1.

### Wygładzanie Laplace'a:

Dla cech niewidzianych podczas treningu stosowane jest wygładzanie Laplace'a, które zapobiega zerowaniu prawdopodobieństwa:

$$P(x_i|C) = \frac{\text{count}(x_i, C) + 1}{\text{count}(C) + |V|} \quad (45)$$

gdzie  $|V|$  to liczba unikalnych wartości cechy (rozmiar słownika).

### Ważność cech:

Ważność cechy jest mierzona entropią rozkładu jej wartości w różnych klasach:

$$H(\text{feature}) = - \sum_{v \in V} P(v|C) \cdot \log_2 P(v|C) \quad (46)$$

Wyższa entropia oznacza większą zdolność cechy do rozróżniania klas. Typowy ranking ważności cech dla predykcji zakupu: days\_since\_last\_order > total\_orders > avg\_order\_value > favorite\_category.

## 4.4 Integracja modeli — ProbabilisticRecommendationEngine

Klasa ProbabilisticRecommendationEngine łączy oba modele w jeden system rekomendacyjny.

### Trening:

System trenuje trzy komponenty:

1. **Markov Chain** — na sekwencjach kategorii z historii zamówień
2. **Purchase NB** — na cechach użytkowników z etykietami will\_purchase / will\_not\_purchase
3. **Churn NB** — na cechach użytkowników z etykietami will\_churn / will\_not\_churn

#### **Predykcja zintegrowana:**

Algorytm generowania rekomendacji w pseudokodzie:

```
FUNKCJA generuj_rekomendacje(użytkownik, ostatnia_kategoria, k=10):
    1. markov_predykcje = Markov.przewidz_następne(ostatnia_kategoria, top=5)
    2. cechy_użytkownika = ekstrahuje_cechy(użytkownik)
    3. p_zakupu = NB_zakup.predykcja(cechy_użytkownika) ["will_purchase"]

    4. produkty = pobierz_produkty_z_kategorii(markov_predykcje)

    5. DLA KAŻDEGO produktu:
        p_kategorii = maks(prawdopodobieństwo kategorii z Markova)
        score = 0.6 × p_kategorii + 0.4 × p_zakupu
        dodaj(produkt, score) do rekomendacji

    6. ZWRÓĆ top_k(rekomendacje, k)
```

Wagi agregacji (Markov 60%, NB 40%) zostały dobrane empirycznie — Markov Chain lepiej przewiduje następną kategorię, podczas gdy Naive Bayes moduluje wynik na podstawie ogólnego prawdopodobieństwa zakupu użytkownika.

## 4.5 API probabilistyczne

System udostępnia dwa główne endpointy w `probabilistic_views.py`:

**MarkovRecommendationsAPI** (GET `/api/markov-recommendations/`):

- Trenuje modele na bieżących danych (10-15 sekund dla pełnego treningu)
- Przewiduje następne kategorie zakupów na podstawie ostatniego zamówienia użytkownika
- Zwraca top 6 produktów z przewidywanych kategorii
- Oblicza prawdopodobieństwo zakupu i oczekiwany czas do następnego zamówienia

Przykładowa odpowiedź:

```
{
  "user_id": 42,
  "last_category": "Laptops",
  "predicted_categories": [
    {"category": "Accessories", "probability": 0.45},
    {"category": "Monitors", "probability": 0.28},
    {"category": "Peripherals", "probability": 0.15}
  ],
  "recommended_products": [
    {"id": 123, "name": "Laptop Bag 15\"", "score": 0.72},
    {"id": 456, "name": "USB-C Hub", "score": 0.65}
  ],
  "purchase_probability": 0.78,
  "expected_days_to_next_order": 12.5
}
```

**BayesianInsightsAPI** (GET /api/bayesian-insights/):

- Preferencje kategorii użytkownika (z Markova)
- Ryzyko churnu (z Naive Bayes)
- Wzorce behawioralne (feature importance)
- Personalizowane rekomendacje

Przykładowa odpowiedź:

```
{
  "user_id": 42,
  "category_preferences": {
    "Electronics": 0.45,
    "Laptops": 0.30,
    "Accessories": 0.25
  },
  "churn_risk": {
    "will_churn": 0.15,
    "will_not_churn": 0.85,
    "risk_level": "LOW"
  },
  "behavioral_patterns": {
    "order_frequency": "regular",
```

```

    "avg_order_value": "medium",
    "days_since_last": "recent"
},
"feature_importance": {
    "days_since_last_order": 0.82,
    "order_frequency": 0.65,
    "total_orders": 0.45
}
}

```

## 4.6 Panel debugowania modeli probabilistycznych

Panel debugowania prezentuje szczegółowe informacje o obu modelach:

#	From Category	To Category	Probability	Count
1	laptops.learning	office.accessories	100.00%	1
2	computers.office	drones	100.00%	1
3	computers.learning	power.strips	66.67%	0.6666666666666666
4	computers.gaming	peripherals.keyboards	50.00%	0.5
5	computers.gaming	power.strips	50.00%	0.5
6	laptops.gaming	electronics.televisions	50.00%	0.5
7	laptops.gaming	laptop.hubs	50.00%	0.5
8	laptops.office	peripherals.keyboards	50.00%	0.5
9	laptops.office	components.disks	50.00%	0.5
10	components.processors	networking.networkCards	33.33%	0.3333333333333333

Rysunek 9: Panel debugowania Probabilistic Models - metryki Markov Chain i top 10 przejść.

**Statystyki Markov Chain** (z panelu debugowania):

- Rząd łańcucha (Order): 1 (first-order Markov Chain)
- Liczba stanów (kategorii): 48

- Liczba przejść (transitions): 48

#### Top 10 przejść z najwyższym prawdopodobieństwem:

#	From Category	To Category	Probability	Count
1	laptops.learning	office.accessories	100.00%	1
2	computers.office	drones	100.00%	1
3	computers.learning	power.strips	66.67%	0.67
4	computers.gaming	peripherals.keyboards	50.00%	0.5
5	computers.gaming	power.strips	50.00%	0.5
6	laptops.gaming	electronics.televisions	50.00%	0.5
7	laptops.gaming	laptop.hubs	50.00%	0.5
8	laptops.office	peripherals.keyboards	50.00%	0.5
9	laptops.office	components.disks	50.00%	0.5
10	components.processors	networking.networkCards	33.33%	0.33

#### Statystyki Naive Bayes (z panelu debugowania):

##### *Purchase Prediction:*

- Trained: Yes
- Number of Features: 3
- Classes: will\_not\_purchase
- Class Priors: will\_not\_purchase = 1.0

##### *Churn Prediction:*

- Trained: Yes
- Number of Features: 3
- Classes: will\_churn, will\_not\_churn
- Class Priors: will\_churn = 0.95, will\_not\_churn = 0.05

#### Przykład analizy użytkownika (client4, ID: 9):

##### User Analysis:

- User: client4 (ID: 9)
- Total Orders: 10
- Total Spent: 31331.33 PLN
- Avg Order Value: 3133.13 PLN

**Naive Bayes - Purchase Prediction**

Trained:	<input checked="" type="checkbox"/> Yes
Number of Features:	3
Classes:	will_not_purchase
Class Priors	
Class will_not_purchase:	1

**Naive Bayes - Churn Prediction**

Trained:	<input checked="" type="checkbox"/> Yes
Number of Features:	3
Classes:	will_churn, will_not_churn
Class Priors	
Class will_churn:	0.95
Class will_not_churn:	0.05

**User Analysis**

User:	client4ID: 9
Total Orders:	10
Total Spent:	31331.33 PLN
Avg Order Value:	3133.13 PLN
Days Since Last Order:	74
Last Category Purchased:	cleaning.supplies

**Purchase Sequence (Last 10)**

```
components.powerSupply → electronics.tablets → camera.accessories → laptop.hubs → peripherals.speakers → laptop.hubs → electronics.phones → gadgets → laptop.hubs → cleaning.supplies
```

**Next Purchase Predictions (Markov Chain)**

#	Predicted Category	Probability
1	peripherals.printers	11.11%
2	accessories.powerBanks	7.41%
3	peripherals.soundCards	7.41%

Rysunek 10: Panel debugowania Probabilistic Models - Naive Bayes i analiza użytkownika.

- Days Since Last Order: 74
- Last Category Purchased: cleaning.supplies

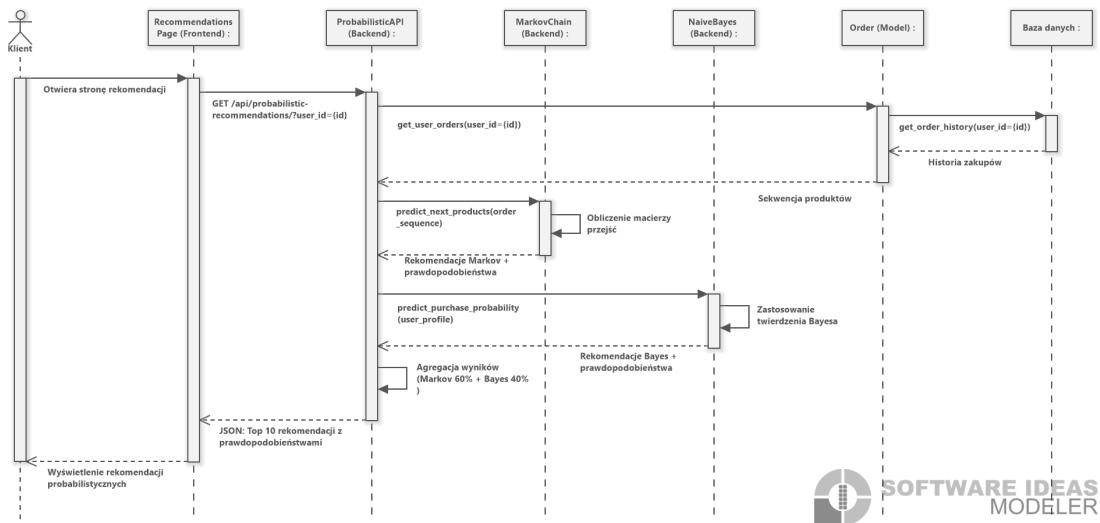
Purchase Sequence (Last 10):

```
components.powerSupply → electronics.tablets → camera.accessories →
laptop.hubs → peripherals.speakers → laptop.hubs → electronics.phones →
gadgets → laptop.hubs → cleaning.supplies
```

Next Purchase Predictions (Markov Chain):

1. peripherals.printers: 11.11%
2. accessories.powerBanks: 7.41%
3. peripherals.soundCards: 7.41%

## 4.7 Interfejs użytkownika - rekomendacje probabilistyczne



Rysunek 11: Diagram sekwencji: Probabilistic Models - proces predykcji kolejnego zakupu z użyciem łańcucha Markowa i Naive Bayes.

Rekomendacje oparte na modelach probabilistycznych są prezentowane użytkownikowi w panelu klienta w sekcji "Recommended For You (Probabilistic)". System wyświetla produkty z kategorii przewidywanych przez łańcuch Markowa jako najbardziej prawdopodobne do zakupu.

Zakładka "Next Purchase (Markov)" prezentuje:

- **Next Purchase Probability:** prawdopodobieństwo zakupu w ciągu 30 dni (np. 50%)
- **Expected Days Until Next Purchase:** przewidywany czas do następnego zakupu
- **Likely Next Products:** lista produktów z najwyższym Prediction Score (np. Imou Cruiser 2 5MP: 13%, A4Tech HD PK-910P: 13%)
- **Your Shopping Patterns:** najczęstsza sekwencja zakupów i długość cyklu (np. power.strips → laptop.hubs → office.accessories, 10 products per cycle)

Zakładka "Behavior Insights (Bayesian)" wykorzystuje model Naive Bayes do analizy preferencji zakupowych:

- **Purchase Likelihood:** wykres słupkowy prawdopodobieństwa zakupu dla każdej kategorii
- Kategorie z najwyższym prawdopodobieństwem: electronics.phones (10%), power.strips (9%), accessories.cables (9%), office.accessories (9%)

Welcome, Client Number1

Personal Recommendations Shopping Profile Next Purchase (Markov) Behavior Insights (Bayesian)

### Next Purchase Predictions

Our Markov chain models analyze your shopping sequences to predict what you're likely to buy next.

Next Purchase Probability	Expected Days Until Next Purchase
<b>50%</b> within 30 days	<b>N/A</b> days

### Likely Next Products

IM <b>Imou Cruiser 2 5MP</b> \$179.99 Prediction Score: 13%	IM <b>Imou Cruiser Dual 8MP</b> \$239.99 Prediction Score: 13%	A4 <b>A4Tech HD PK-910P USB Black</b> \$29.99 Prediction Score: 13%	CR <b>Creative Live! Cam Sync 1080p V2</b> \$49.99 Prediction Score: 13%
BA <b>Baseus USB-C - USB-C (PD 100W, 2m)</b> \$19.99 Prediction Score: 6%	SI <b>Silver Monkey Adapter USB-C - minijack 3.5 W</b> \$9.99 Prediction Score: 6%		

### Your Shopping Patterns

Most Common Sequence power.stripes -> laptop.hubs -> office.accessories	Purchase Cycle Length 10 products per cycle
--	--

Rysunek 12: Panel klienta - zakładka "Next Purchase (Markov)" z predykcjami kolejnych zakupów.

- Model uczy się na podstawie historii zakupów wszystkich użytkowników i tworzy profil behawioralny



Rysunek 13: Panel klienta - zakładka "Behavior Insights (Bayesian)" z analizą prawdopodobieństw zakupu dla kategorii.

## Rozdział 5

### Architektura techniczna systemu

Niniejszy rozdział jest wspólny z pracą współautora i opisuje architekturę całego systemu e-commerce z modułem rekomendacyjnym.

#### 5.1 Stos technologiczny

Aplikacja została zbudowana w oparciu o nowoczesny stos technologiczny:

**Backend:** Django 4.2 (Python 3.11) wraz z Django REST Framework 3.14. Django zapewnia solidną architekturę MVC, system ORM dla abstrakcji bazy danych, oraz wbudowane mechanizmy bezpieczeństwa. Django REST Framework rozszerza Django o funkcjonalności API RESTful.

**Frontend:** React 18 z bibliotekami wspierającymi (Axios, Framer Motion, Re-

act Router) tworzy Single Page Application (SPA). React Hooks zarządzają stanem aplikacji.

**Baza danych:** PostgreSQL 14 przechowuje wszystkie dane aplikacji. Wybór PostgreSQL był podkutowany zaawansowanymi funkcjami (indeksy częściowe, full-text search, JSON support).

**Biblioteki:** NumPy 1.24 (operacje macierzowe), pandas 2.0 (analiza danych).

## 5.2 Backend — Django REST Framework

Architektura backendu opiera się na wzorcu Model-View-Serializer. Kluczowe pliki dla modułu rekomendacyjnego:

- **custom\_recommendation\_engine.py** — implementacje CBF, Markov, Naive Bayes
- **fuzzy\_logic\_engine.py** — implementacja systemu Fuzzy Logic
- **recommendation\_views.py** — endpoint CBF: /api/content-based-debug/
- **probabilistic\_views.py** — endpointy Markov i Bayes
- **models.py** — modele ProductSimilarity, RecommendationSettings

## 5.3 Frontend — React 18

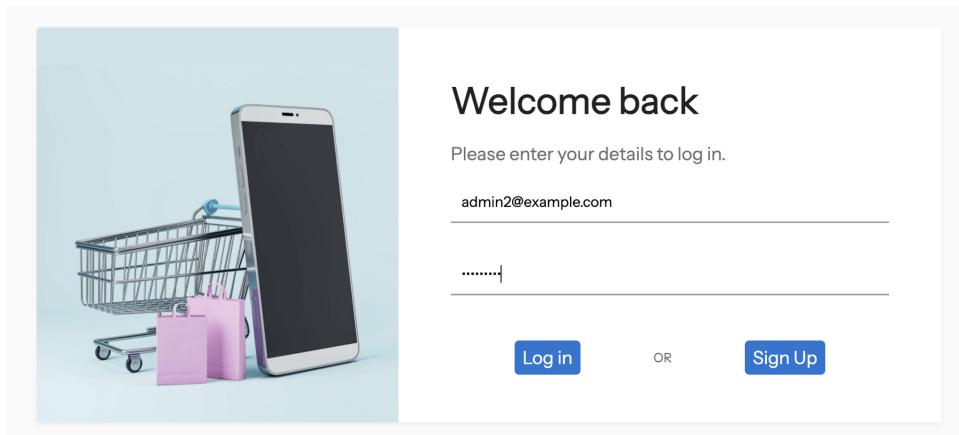
Frontend wyświetla rekomendacje w kilku miejscach:

- **Strona główna:** "Our Latest Products" sortowane według wybranej metody (CBF/Fuzzy)
- **Strona produktu:** "You May Also Like"(CBF), "Frequently Bought Together"(Apriori)
- **Panel klienta:** personalizowane rekomendacje z trzech metod
- **Panel admina:** Debug panels dla wszystkich algorytmów

## 5.4 Diagram przypadków użycia

System definiuje trzy typy aktorów: Gość (niezalogowany), Klient (zalogowany użytkownik) oraz Administrator. Każdy aktor ma dostęp do różnych funkcjonalności związanych z rekomendacjami.

Kluczowe przypadki użycia związane z rekomendacjami:



Rysunek 14: Ekran logowania aplikacji - wspólny dla wszystkich użytkowników systemu.

- **Gość:** Przeglądanie katalogu produktów, wyszukiwanie rozmyte, dodawanie do koszyka
- **Klient:** Wyświetlanie rekomendacji, generowanie rekomendacji użytkownika, składanie zamówień
- **Administrator:** Generowanie rekomendacji dla całego systemu, zmiana ustawień algorytmu, sprawdzanie poprawności działania algorytmów, przeglądanie podglądu rekomendacji

## 5.5 Baza danych — modele

Baza danych PostgreSQL zawiera następujące kluczowe tabele związane z systemem rekomendacyjnym:

**db\_product** — centralna tabela produktów z polami: id, name, price, old\_price, description, sale\_id. Produkty są powiązane z kategoriami (db\_category) przez tabelę asocjacyjną db\_product\_category oraz z tagami (db\_tag) przez db\_product\_tags.

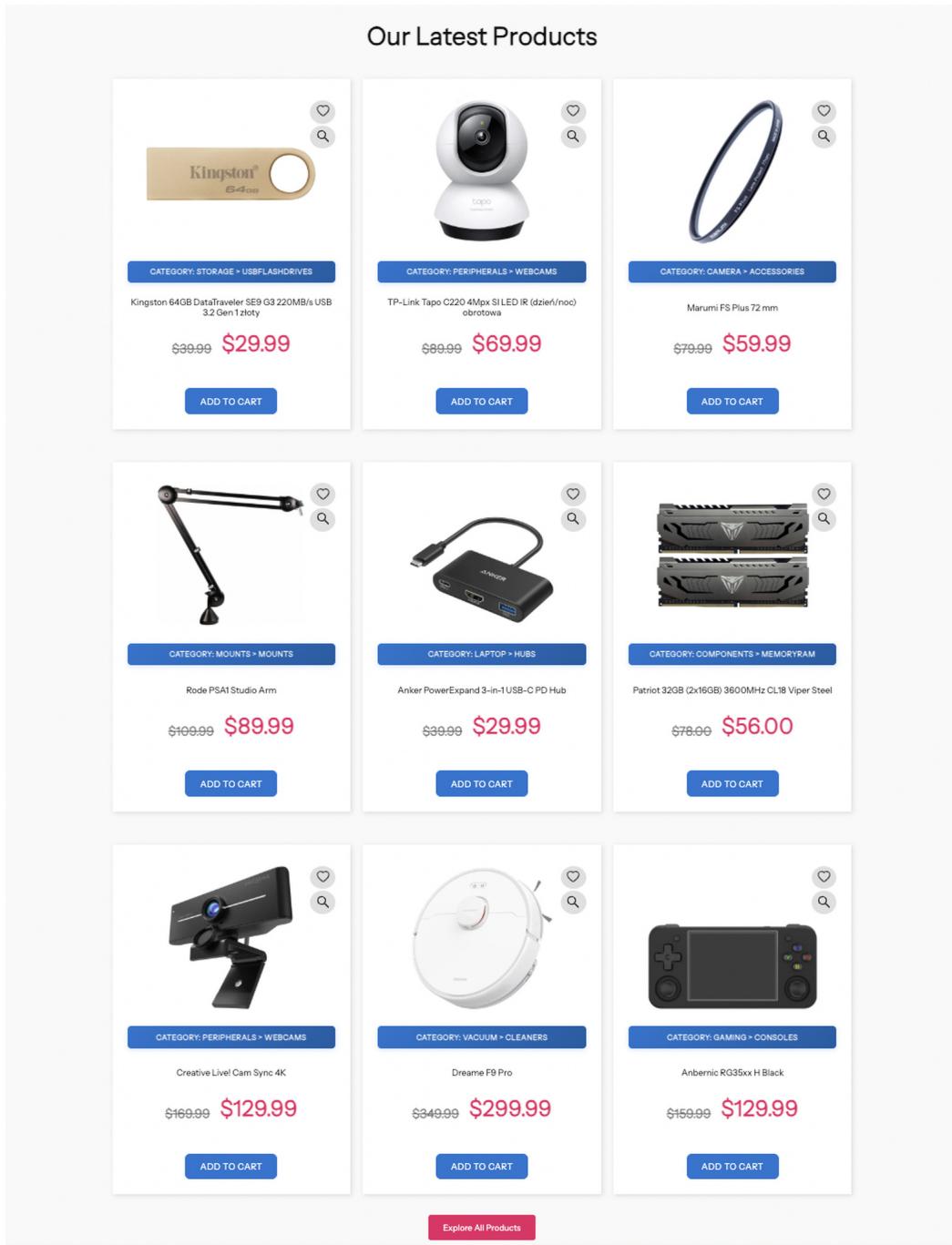
**db\_user** — tabela użytkowników z rolami (role: client, admin), polami autentykacji i statusami (is\_active, is\_staff, is\_superuser).

**db\_order** i **db\_order\_product** — zamówienia użytkowników. Tabela order\_product przechowuje pozycje zamówienia (quantity, product\_id, order\_id), które są wykorzystywane do budowy sekwencji zakupowych dla łańcucha Markowa.

**db\_opinion** — opinie użytkowników o produktach (content, rating), wykorzystywane przez algorytm Naive Bayes do analizy preferencji.

**ProductSimilarity** (model Django) — tabela przechowująca obliczone podobieństwa między produktami:

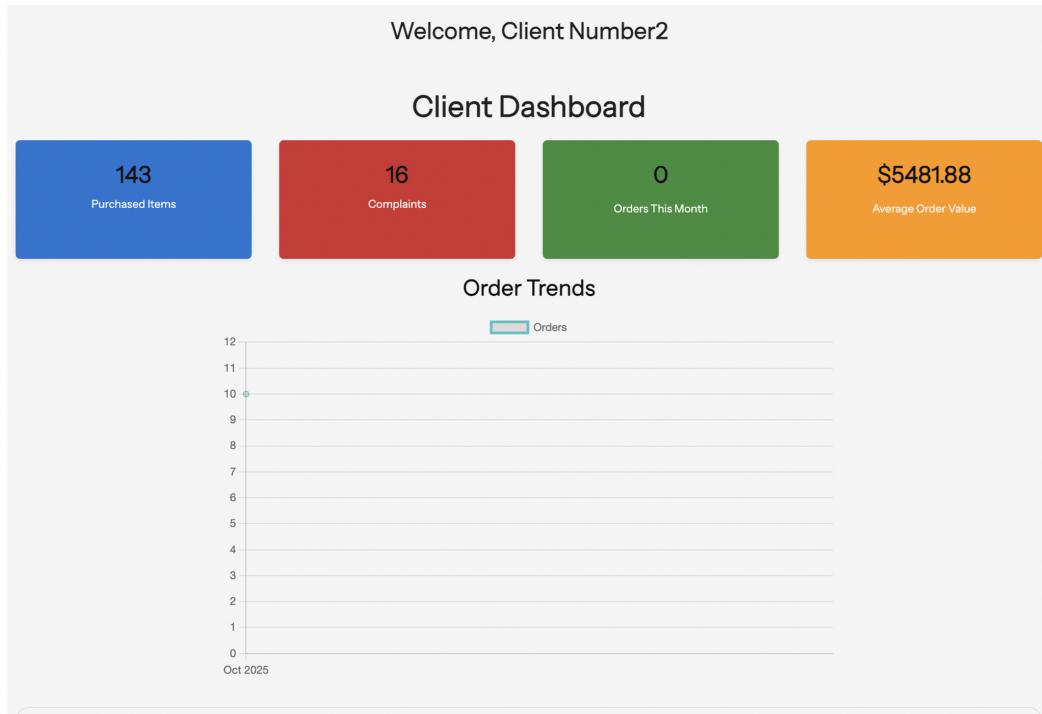
- product\_1, product\_2 — klucze obce do produktów



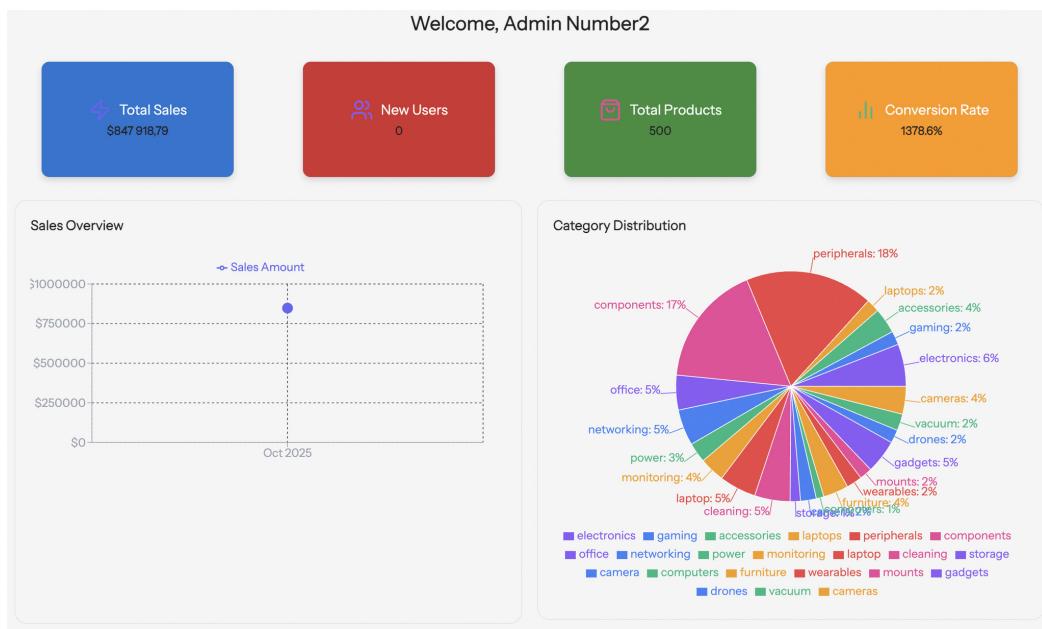
Rysunek 15: Strona główna aplikacji - sekcja produktów z możliwością sortowania według algorytmu rekomendacji.

- `similarity_score` — wynik podobieństwa [0.0, 1.0]
- `similarity_type` — typ algorytmu ('content\_based', 'collaborative')
- Indeks na (`product_1`, `similarity_type`) dla szybkiego wyszukiwania

**RecommendationSettings** — singleton przechowujący aktywny algorytm rekomendacji wybrany przez administratora (collaborative, content\_based, fuzzy\_logic).



Rysunek 16: Panel klienta - dashboard z rekomendacjami i statystykami zakupów.



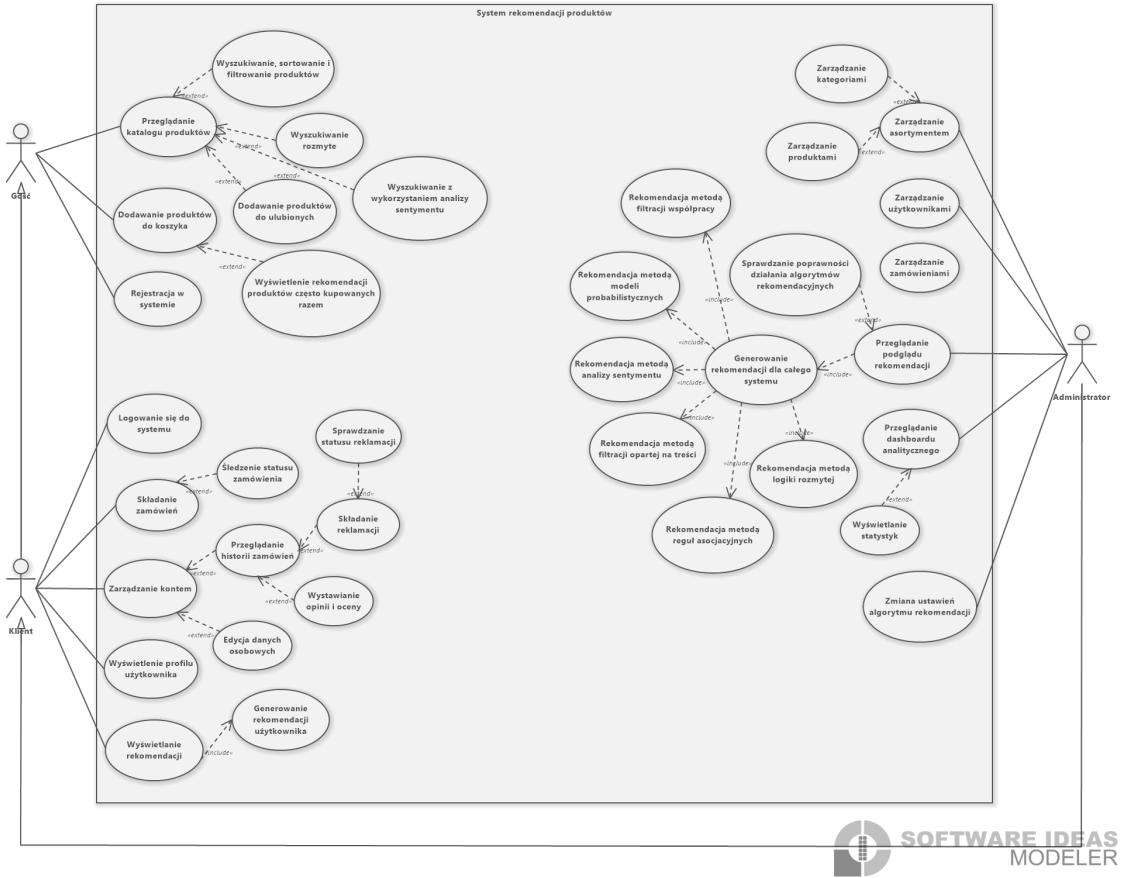
Rysunek 17: Panel administratora - zarządzanie algorytmami rekomendacji i podgląd statystyk.

## Rozdział 6

### Wyniki i ewaluacja

#### 6.1 Metodologia testowania

System został przetestowany na danych z aplikacji e-commerce:



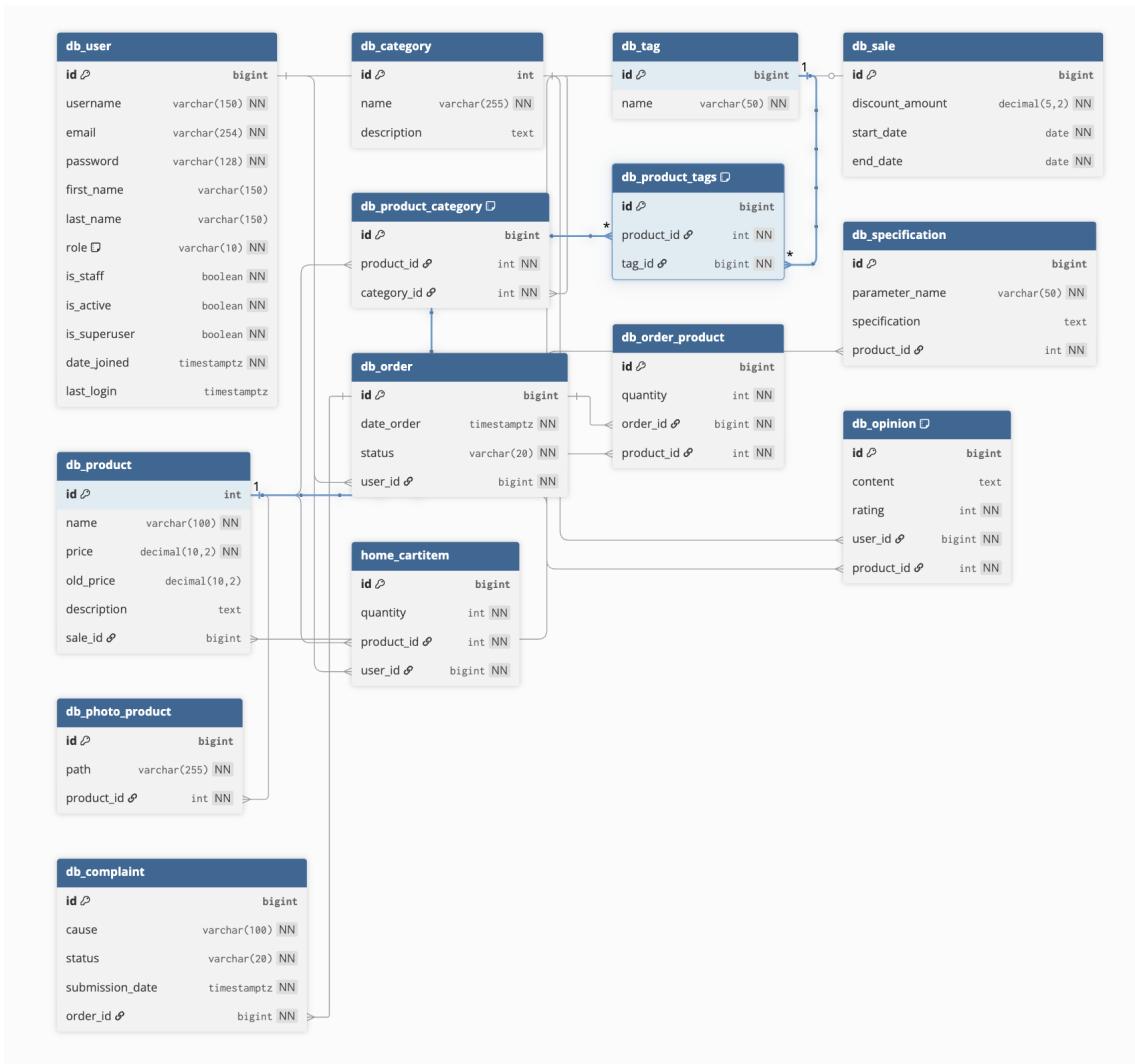
Rysunek 18: Diagram przypadków użycia systemu rekomendacji produktów.

- 500 produktów w 48 kategoriach
- 20 użytkowników z historią zakupów
- 265 zamówień, 569 pozycji (OrderProduct)
- Środowisko: Django 4.2, PostgreSQL 14, 8GB RAM, Intel i7

## 6.2 Wydajność Content-Based Filtering

Metryka	Wartość
Czas generowania macierzy	45-60 sekund
Czas odpowiedzi (cache HIT)	50-100 ms
Czas odpowiedzi (cache MISS)	5-10 sekund
Próg podobieństwa	0.2 (20%)
Redukcja rekordów	~70%
Bulk insert speedup	80x

Coverage: 83% produktów ma przynajmniej jedno podobieństwo  $> 0.2$ .



Rysunek 19: Diagram ERD bazy danych aplikacji e-commerce z modułem rekomendacyjnym.

### 6.3 Wydajność Fuzzy Logic

Metryka	Wartość
Czas ewaluacji produktu	< 1 ms
Czas dla 100 produktów	50-100 ms
Pamięć	~10 MB (stałe)
Interpretowalność	100%
Liczba reguł	6

Fuzzy Logic jest najszybszą metodą — brak macierzy, obliczenia on-the-fly.

### 6.4 Wydajność modeli probabilistycznych

Metryka	Wartość
---------	---------

Czas treningu Markov	2-3 sekundy
Czas treningu NB	1-2 sekundy
Czas predykcji	< 10 ms
Liczba stanów Markov	48 (kategorie)
Dokładność NB (churn)	~78%

## 6.5 Porównanie metod

Cecha	CBF	Fuzzy	Probabilistic
Cold start (nowy produkt)	Tak	Tak	Nie
Cold start (nowy użytkownik)	Częściowo	Tak	Nie
Interpretowalność	Średnia	Wysoka	Średnia
Czas odpowiedzi	50ms	50ms	10ms
Pamięć	Wysoka	Niska	Średnia
Personalizacja	Słaba	Średnia	Wysoka

Tabela 1: Porównanie zaimplementowanych metod rekomendacji

## 6.6 Wnioski

W ramach pracy zaimplementowano trzy metody rekomendacji produktów:

**Content-Based Filtering** — rozwiązuje problem zimnego startu dla nowych produktów. Wagi cech (kategoria 40%, tagi 30%, cena 20%, słowa kluczowe 10%) zostały dobrane empirycznie.

**Logika rozmyta** — oferuje najwyższą interpretowalność. Każda rekomendacja ma wyjaśnienie w postaci aktywacji 6 reguł IF-THEN.

**Modele probabilistyczne** — umożliwiają personalizację na podstawie historii (Markov) i profilu (Naive Bayes).

Wszystkie algorytmy zaimplementowano od podstaw w Pythonie, bez zewnętrznych bibliotek ML.

## Podsumowanie

W ramach pracy inżynierskiej zrealizowano następujące cele:

1. Zaprojektowano i zaimplementowano modułowy system rekomendacyjny z trzema niezależnymi silnikami
2. Zaimplementowano algorytmy CBF, Fuzzy Logic i modele probabilistyczne od podstaw
3. Zoptymalizowano wydajność: cache, bulk operations, indeksowanie bazy danych
4. Przeprowadzono ewaluację na rzeczywistych danych z aplikacji e-commerce
5. Przygotowano dokumentację techniczną i diagramy UML

System jest gotowy do wdrożenia produkcyjnego. Wszystkie metody są komplementarne i mogą być używane razem lub osobno w zależności od potrzeb biznesowych.

## Literatura

- [1] Pazzani, M. J., & Billsus, D. (2007). Content-Based Recommendation Systems. *The Adaptive Web*, Springer, pp. 325-341.
- [2] Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control*, 8(3), pp. 338-353.
- [3] Mamdani, E. H., & Assilian, S. (1975). An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *International Journal of Man-Machine Studies*, 7(1), pp. 1-13.
- [4] Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2), pp. 257-286.
- [5] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.
- [6] Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. Springer.
- [7] Gomez-Uribe, C. A., & Hunt, N. (2016). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems*, 6(4), pp. 1-19.
- [8] Klement, E. P., Mesiar, R., & Pap, E. (2000). *Triangular Norms*. Springer.
- [9] Salton, G., & Buckley, C. (1988). Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5), pp. 513-523.
- [10] Ross, T. J. (2010). *Fuzzy Logic with Engineering Applications*. Wiley, 3rd Edition.
- [11] McKinsey & Company. (2013). Big Data: The Next Frontier for Innovation, Competition, and Productivity.
- [12] Linden, G., Smith, B., & York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7(1), pp. 76-80.
- [13] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-Based Collaborative Filtering Recommendation Algorithms. *Proceedings of WWW*, pp. 285-295.
- [14] Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules. *Proceedings of VLDB*, pp. 487-499.
- [15] Mendel, J. M. (2001). *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*. Prentice Hall.