AKADEMIA TARNOWSKA
Wydział Politechniczny

**Kierunek:** ***Informatyka***

Specjalność: *Informatyka Stosowana*

Specjalizacja: *Inżynieria Oprogramowania*

*2023/2024*

*Jakub Goch*

PRACA INŻYNIERSKA

# *Aplikacja webowa do zakupów artykułów spożywczych*

Promotor pracy:

*mgr inż. Tomasz Gądek*

Tarnów, 2024

# Spis treści

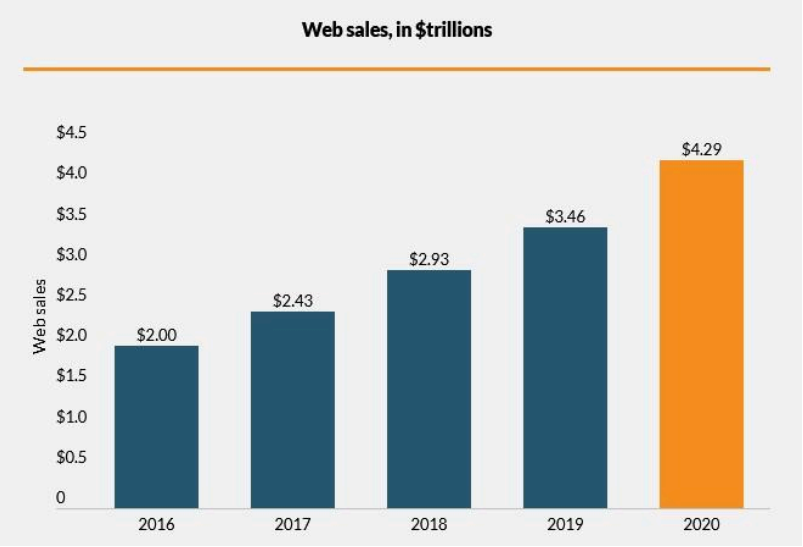
# 1. Wstęp

Celem niniejszej pracy dyplomowej było omówienie aplikacji służącej do zakupów artykułów spożywczych. Strona została napisana w architekturze klient-serwer. W dalszej części pracy przedstawione zostaną wykorzystane technologie, wybrane testy oraz interfejs aplikacji.

## 1.1. Motywacja

Handel elektroniczny znany również pod nazwą e-commerce jest w dzisiejszych czasach częścią życia ludzi. Wiele osób nie wyobrażałaby sobie teraz świata bez możliwości robienia zakupów w formie elektronicznej. Przełomowym momentem w historii sklepów był pierwszy założony sklep internetowy w Polsce. Była to księgarnia **Nepo**, która powstała w 1994 roku. W późniejszym czasie powstawały kolejne sklepy online, które odniosły sukces i dzisiaj można korzystać z ich usług.

Jedną z najpopularniejszych aktywności w internecie są zakupy. Wynika to głównie z tego, że ten sposób jest bardzo wygodny, nie trzeba wychodzić z domu, żeby kupić to co jest potrzebne i często online można znaleźć większy asortyment niż w sklepach stacjonarnych. Przemawiają za tym różnorodne statystyki. Na rynku światowym sprzedaż detaliczna e-commerce w 2017 roku osiągnęła wartość około 2,4 biliona dolarów. Rysunek 1.1 przedstawia wzrost wartości rynku sprzedaży internetowej w latach 2016-2020 [3].




**Rys. 1.1.** Wzrost wartości sprzedaży e-commerce na świecie w latach 2016-2020 [3]

W roku 2020 wartość polskiego rynku e-commerce przekroczyła 100 miliardów złotych i aż 77% Polaków twierdzi, że robi zakupy przez Internet. Wyraźnie wskazuje to na ogromną popularność i dynamiczny rozwój sklepów online [1].

## 1.2. Cel pracy

Celem niniejszej pracy inżynierskiej było zaimplementowanie aplikacji webowej służącej do zakupów artykułów spożywczych. Aplikacja ma zapewnić szybkie i wygodne zamawianie produktów bezpośrednio z lokalnych sklepów.

W Polsce popularne stało się zamawianie jedzenia online przez aplikacje mobilne lub webowe. Jest to pewien sposób na oszczędność czasu, ponieważ nie trzeba nawet wychodzić z domu. Przykładem mogą być platformy takie jak **Uber Eats** czy **Pyszne.pl**, które odniosły sukces. Istnieje jednak pewna luka na rynku, jeśli chodzi o aplikacje umożliwiające zakupy produktów spożywczych ze sklepów stacjonarnych takich jak **Biedronka** lub **Lidl**. Za granicą istnieją rozwiązania takie jak **Instacart**. W Polsce nie jest to jednak jeszcze, aż tak popularne.

## 1.3. Zakres pracy

Zakresem niniejszej pracy było zaimplementowanie aplikacji, która umożliwia zakupy artykułów spożywczych przez użytkowników zapewniając przy tym wygodę użytkowania. System składa się z trzech części:

- Aplikacja internetowa (webowa) - pozwala użytkownikom przy użyciu przeglądarki internetowej na założenie konta, przeglądanie produktów, składanie zamówień oraz dokonywanie płatności.
- Aplikacja serwerowa - odpowiada za odbieranie danych z aplikacji webowej. Obsługuje żądania **HTTP** (ang. *HyperText Transfer Protocol*) użytkowników, które opisane są w rozdziale 3.4 "Dokumentacja API".
- Baza danych - przechowuje informacje na temat produktów, dane użytkowników oraz ich zamówienia. **MongoDB** spełnia wymagania, które stawia rynek e-commerce na temat dostępności i przechowywania danych, co sprawia, że jest dobrym wyborem [1].

# 2. Opis wykorzystanych technologii

Wybór odpowiednich technologii jest bardzo ważny podczas projektowania aplikacji internetowej. Aplikacja powstała z wykorzystaniem **MongoDB**, **Express.js**, **React.js** oraz **Node.js** [11][5][13][12].

Wymienione technologie łączą się w popularny stos technologiczny **MERN**. Jest on odpowiedni do tworzenia sklepu internetowego. Połączenie tych narzędzi zapewnia, że aplikacja będzie wydajna, a także pozwala na dostosowanie do zmieniających się potrzeb rynku e-commerce. W dalszej części tego rozdziału zostanie opisana każda z użytych technologii.

## 2.1. Aplikacja serwerowa

### 2.1.1. Node.js

Do zaimplementowania części serwerowej aplikacji wykorzystano technologię **Node.js**, która zdobyła ogromną popularność i jest coraz częściej wybierana przez programistów. Narzędzie to oparte jest na silniku **Javascript** i pozwala na uruchomienie kodu po stronie serwera [12].

**Node** jest technologią open-source (otwartym oprogramowaniem), która działa na wielu platformach. Służy przede wszystkim do obsługi żądań **HTTP** (ang. *Hypertext Transfer Protocol*) co oznacza, że jest świetnym rozwiązaniem do tworzenia aplikacji serwerowych. **Node** może być wykorzystywany również do obsługi plików, zarządzania bazami danych, a nawet komunikacji np. jako aplikacja czatowa. Jego popularność nie jest przypadkowa i świadczą o tym:

- Wydajność - obsługuje asynchroniczność przez co może operować na wielu równoczesnych połączeniach z małym obciążeniem procesora.
- Wieloplatformowość - pozwala na tworzenie serwerów i aplikacji działających na różnych systemach operacyjnych.
- Społeczność - posiada ogromną społeczność korzystającą z tej technologii, co przekłada się na dostępność bibliotek ułatwiających pracę z tym narzędziem.
- Łatwość nauki - nauka **Node.js** jest dużo prostsza dla osób, które znają już język **JavaScript**.

**Node.js** może być porównywany do innych języków programowania używanych przy implementowaniu serwerów np. **PHP**, **Ruby on Rails**, czy **Java**. Jego przewagą jest jednak to, że programiści mogą pisać kod **JavaScript** po stronie serwera, jak i również po stronie klienta. Zdecydowanie zwiększa to szybkość tworzenia oprogramowania.

### 2.1.2. Express.js

**Express.js** to framework wykorzystywany z **Node.js** przy aplikacjach serwerowych. Jest on napisany w języku **JavaScript**. Służy on programistom do zarządzania **API** i żądaniami **HTTP**, **HTTPS**.

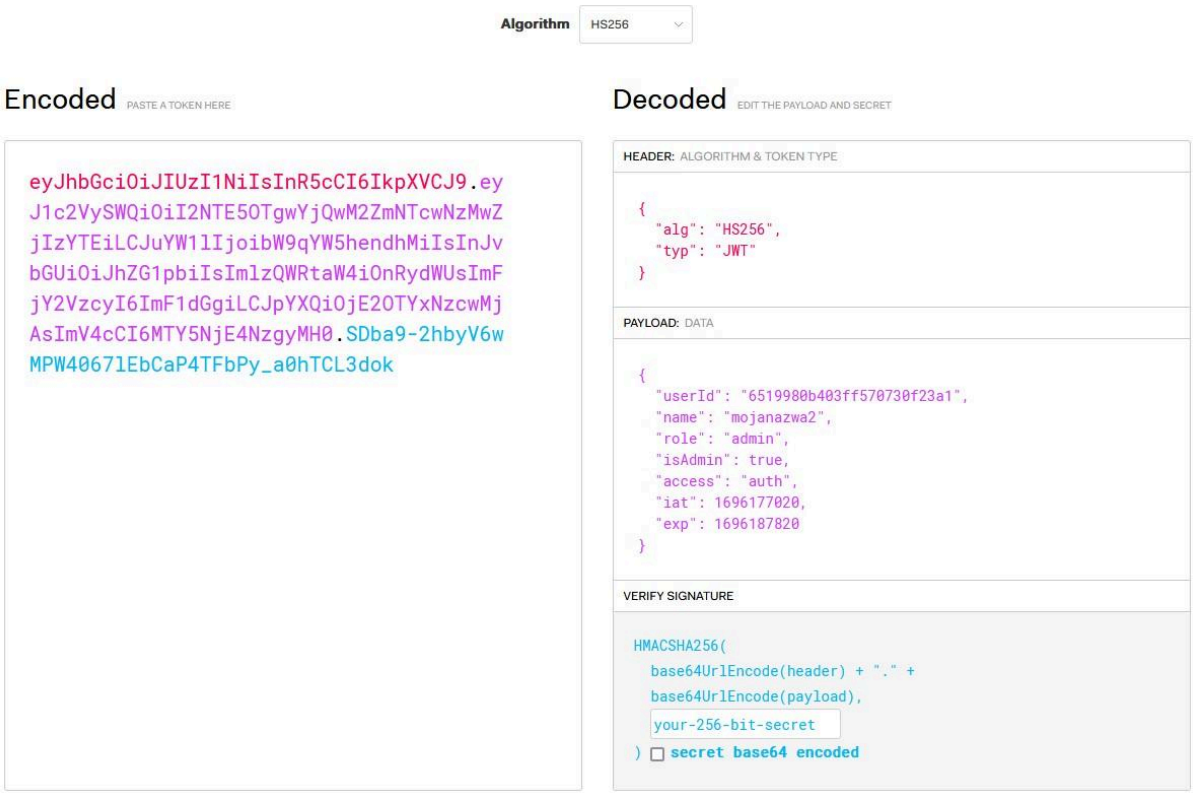
Zarówno poprzednio opisane technologie, takie jak **Node, Express.js** są łatwe do opanowania dla osób, które znają już język **JavaScript**. Dodatkowymi zaletami tego narzędzia jest jego minimalizm i szybkość [5]. Obsługuje on asynchroniczność co bardzo dobrze wpływa na jego wydajność. **Express** jest darmowy na licencji open-source, więc każdy może z niego bezproblemowo korzystać.

### 2.1.3. JSON Web Token

**JWT** (ang. *JSON Web Token*) to jeden z popularniejszych sposobów na uwierzytelnienie użytkowników i wymianę informacji w aplikacjach internetowych poprzez obiekt **JSON** (ang. JavaScript Object Notation). Informacje, które są przesyłane przez **JWT** są podpisywane cyfrowo za pomocą sekretnego klucza i algorytmu **HMAC**, co zapewnia bezpieczeństwo i pozwala zweryfikować dany token [10]. Token **JWT** składa się z trzech elementów:

- Nagłówek (Header) - złożony z typu tokenu oraz algorytmu użytego do jego podpisania.
- Zawartość (Payload) - zawiera informacje przesyłane w tokenie. Przykładowo mogą to być dane logowania użytkownika. Składa się z tzw. claimów (deklaracji).
- Podpis (Signature) - podpis, który jest tworzony przy użyciu tajnego klucza, headera i payloadu [10].

Wszystkie trzy opisane części przedstawione są podczas dekodowania tokenu na oficjalnej stronie **JWT**.




**Rys. 2.1.** Wizualne przedstawienie dekodowania tokenu [10]

## 2.2. Aplikacja klienta

### 2.2.1. JavaScript

**JavaScript** to niezwykle popularny język programowania, który powstał w 1995 roku. Na początku był on stosowany do dodawania programów na strony internetowe przy użyciu przeglądarki **Netscape Navigator** [4].

Zawdzięcza się mu to, jak wygląda aktualnie Internet ze względu na aplikacje webowe powstałe za jego pośrednictwem. Bardzo często używa się go we współpracy z **HTML** (ang. *HyperText Markup Language*) i **CSS** (ang. *Cascading Style Sheets*).

**JavaScript** jest interpretowany co odróżnia go od innych języków programowania. Oznacza to, że kod jest od razu wykonywany na bieżąco w czasie rzeczywistym, a nie kompilowany przed jego uruchomieniem, jak w językach takich jak **C++** czy **Java**. Jest on ciągle rozwijany poprzez wprowadzane standardy ECMAScript.

Na podstawie języka **JavaScript** powstały cieszące się dużym zainteresowaniem frameworki i biblioteki, takie jak **Angular** lub **React**, którego bardziej szczegółowy opis znajduje się w kolejnym rozdziale 2.2.2 "React.js".

### 2.2.2. React.js

**React.js** to biblioteka utworzona przez Jordana Walke opierająca się na języku **JavaScript**. Przy jej pomocy można w bardzo prosty sposób pisać interaktywne strony internetowe [2].

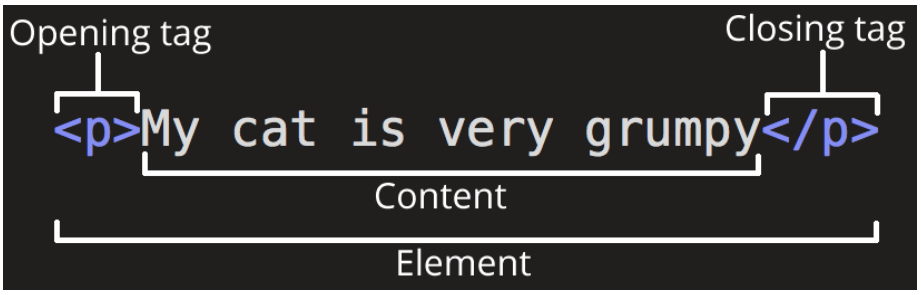
Biblioteka ta cieszy się wielkim zainteresowaniem wśród programistów. Jest to spowodowane tym, że **React** względem innych frameworków jest stosunkowo łatwy do nauki, ma ogromną społeczność i przejrzystą dokumentację. Wiele osób myli **Reacta** z frameworkiem, co nie jest prawdą, ponieważ jest to biblioteka. Nie ma wbudowanych modułów np. do trasowania przez co trzeba dołączać do niego zewnętrzne biblioteki.

**React** bazuje na komponentach, czyli reużywalnych fragmentach kodu, które zwracają **JSX** (ang. JavaScript XML). Pisanie aplikacji z pomocą biblioteki React można porównać do układania puzzli, gdzie każdy element jest oddzielnym komponentem, a wszystkie razem tworzą spójną całość.

### 2.2.3. HTML

**HTML** (ang. *HyperText Markup Language*) to język znaczników, który używany jest do tworzenia stron internetowych [6]. Każdy portal w Internecie wykorzystuje tę technologię budując strukturę strony. Budowa elementu **HTML** na przykładzie paragrafu:

- **znacznik otwierający** - zawiera nazwę elementu "p" zamkniętą w nawiasy ostre.
- **znacznik zamykający** - również zawiera nazwę elementu "p", lecz przed nią pojawia się prawy ukośnik. Całość zamknięta jest w nawiasy ostre.
- **zawartość** - w tym przypadku zawartością paragrafu będzie tekst.




**Rys. 2.2.** Budowa elementu HTML [6]

**HTML** nie jest językiem programowania, a językiem znaczników i przez wielu uważany jest za bardzo prosty do nauczenia. Przy użyciu samego **HTML** możliwe jest napisanie statycznej strony internetowej, a łącząc go z **CSS,** czyli kaskadowymi arkuszami stylów można uzyskać dobrze wyglądającą witrynę internetową.

### 2.2.4. Tailwind CSS

**Tailwind** to framework do **CSS** pozwalający na proste stylowanie aplikacji. Narzędzie to przeszukuje pliki **HTML** i **JavaScript** w projekcie, znajduje nazwy klas dotyczące Tailwinda i zamienia je w zwykły kod **CSS** [16].

Prostota tego rozwiązania polega na tym, że klasyczny kod **CSS** pisze się dłużej, potrzeba do niego osobnych plików z rozszerzeniem .css, a **Tailwind** natomiast pozwala na pisanie nazw klas bezpośrednio w znacznikach **HTML**. Dzięki temu w strukturze folderów projektu można zachować większe zorganizowanie i mniejszy bałagan. **Tailwind** można samodzielnie konfigurować, dodawać własne klasy i używać ich w kodzie.

```
<span className="text-heading-6">
    Taka podstrona nie istnieje!
</span>
```

**Rys. 2.3.** Przykład użycia klasy Tailwind [opracowanie własne]

### 2.2.5. JEST

**JEST** to framework używany do testowania kodu aplikacji. Umożliwia pisanie testów jednostkowych, czyli takich dotyczących mniejszych części kodu oraz testów integracyjnych dotyczących większych fragmentów kodu. Przykładem testu jednostkowego może być sprawdzenie czy napisana funkcja, która powinna zwracać sumę dwóch liczb rzeczywiście prawidłowo ją zwraca.

W aplikacji, która używa technologii **React** można przeprowadzać testy na komponentach. Przykładowo może to być weryfikacja, czy w naszej stopce na stronie wyświetla się odpowiednia treść.

Framework ten jest jednym z popularniejszych, o czym świadczą statystyki przedstawione w dokumentacji technicznej mówiące o 93 milionach pobrań w ciągu jednego miesiąca i prawie 9 milionach zastosowań w repozytoriach **GitHub** [8].

### 2.2.6. React Testing Library

**React Testing Library** (RTL) to biblioteka do testowania kodu aplikacji umożliwiająca interakcje z aplikacją w taki sam sposób, jak robią to użytkownicy [14]. Często stosowany jest wraz z frameworkiem **JEST**.

**RTL** pozwala na sprawdzenie funkcjonalności aplikacji napisanej przy użyciu Reacta np. czy po naciśnięciu w przycisk na stronie ukaże się odpowiedni tekst. Biblioteka ta umożliwia tzw. "mockowanie" danych, czyli używanie przykładowych danych do działania testu.

## 2.3. Baza danych

### 2.3.1. MongoDB

**MongoDB** to najpopularniejsza nierelacyjna baza danych służąca do przechowywania danych. Została ona napisana przez firmę 10 gen w języku C++ w 2009 roku.

Charakterystyczna cecha tej bazy to nierelacyjne podejście co odróżnia ją od popularnych baz SQL takich jak **MySQL** lub **PostgreSQL**. Oznacza to, że w **MongoDB** nie pojawiają się tabele połączone ze sobą relacjami, a kolekcje, które przechowują dokumenty

zawierające dane [11]. Każdy dokument posiada własny unikalny identyfikator, który pozwala go rozpoznać i łączyć z innymi dokumentami. Dane, które się w nich znajdują zapisane są w formacie **JSON**, czyli składają się z par klucz-wartość. Na rysunku 2.4 przedstawiony jest przykładowy dokument bazy **MongoDB**.


**Rys. 2.4.** Przykład dokumentu bazy MongoDB [opracowanie własne]

## 2.4. Ogólne

### 2.4.1. HTTP & HTTPS

**HTTP** (ang. *Hypertext Transfer Protocol*) to protokół warstwy aplikacji, który służy do przesyłania danych między przeglądarką internetową, a serwerem. Działa on zgodnie z modelem klient-serwer, czyli klient wysyła zapytanie do serwera i oczekuje na odpowiedź [7].

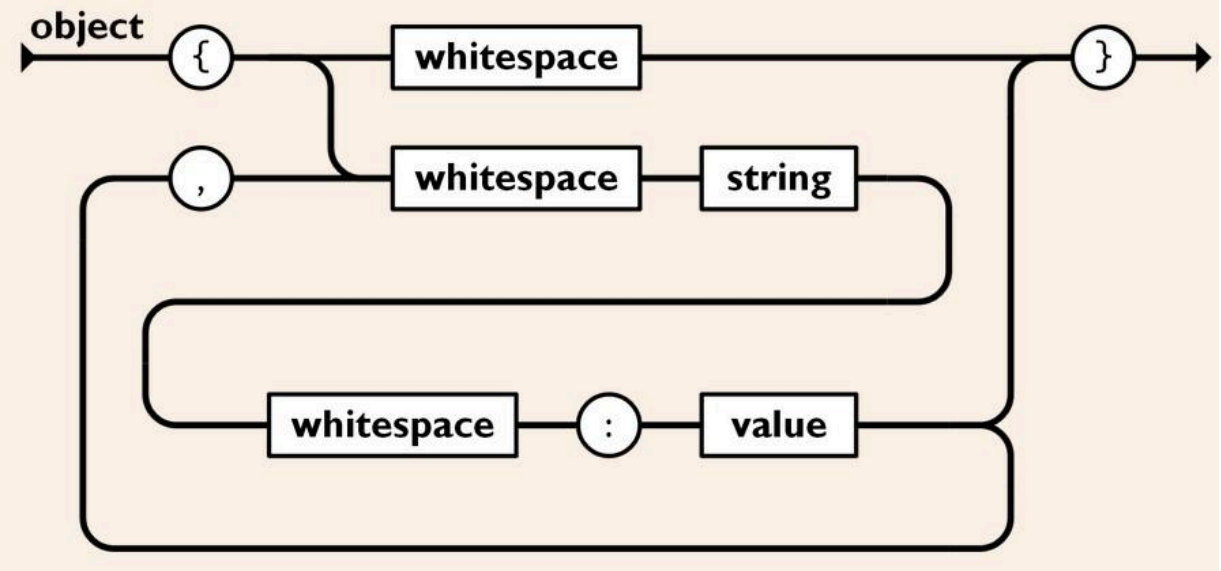
**HTTPS** (ang. *Hypertext Transfer Protocol Secure*) to również protokół **HTTP**, lecz jest chroniony przy pomocy szyfrowania protokołu **SSL/TLS**, który służy do zabezpieczenia przekazywanych danych między przeglądarką, a witryną.


**Rys. 2.5.** Przedstawienie działania protokołu HTTP [opracowanie własne]

### 2.4.2. JSON

**JSON** to format wymiany danych. Jest on bardzo łatwy do odczytania przez ludzi. Dzięki niemu maszyny mogą w prosty sposób generować i analizować dane. **JSON** to format tekstowy, który wywodzi się z języka **JavaScript**. Pozostałe języki programowania takie jak **C**, **C++**, **Java** czy **Python** z łatwością mogą go przetwarzać, ponieważ posiadają wiele bibliotek do obsługi **JSON**.

**JSON** składa się ze zbioru par nazw i wartości i jest nieuporządkowany. Zaczyna się od lewego nawiasu klamrowego, a kończy się prawym nawiasem klamrowym. Między nimi znajduje się nazwa dla naszej wartości i wartość. Oddzielone są dwukropkiem. Po każdej parze występuje przecinek [9]. Rysunek 2.6 przedstawia schemat obiektu **JSON**.




**Rys. 2.6.** Schemat obiektu JSON [9]

### 2.4.3. REST API

**REST** (ang. *Representational State Transfer*) to interfejs przesyłania wiadomości i danych dla architektur takich jak klient-serwer i mikrousług [15]. Cechy **REST API**:

- Zasoby - są to dane, które uzyskujemy używając **REST API**.
- Operacje - obsługuje podstawowe operacje GET, POST, PUT, DELETE.
- Bezstanowość - nie przechowuje stanu klienta między żądaniami.
- Niezależność - klient i serwer są niezależne. Znają tylko nazwy zasobów.

Rysunek 2.7 przedstawia przykład działania **REST API**.


**Rys. 2.7.** Przykład działania REST API [opracowanie własne]

# 3. Implementacja systemu

## 3.1. Zakres funkcji

System składa się z dwóch podstawowych aktorów:

- Użytkownik niezalogowany,
- Użytkownik zalogowany.

Użytkownicy niezalogowani, czyli goście witryny to osoby, które mają najmniej dostępnych uprawnień w aplikacji. Po wejściu na stronę mają możliwość przeglądania produktów, filtrowania i dodania ich do koszyka, logowania i rejestracji. Gdyby gość chciał przejść do płatności związanych z jego koszykiem z produktami, zostanie przekierowany do strony logowania. Musi wtedy się zalogować lub zarejestrować, jeśli konta nie posiada.

Użytkownicy zalogowani posiadają podobne uprawnienia do gości. Mogą przeglądać produkty, filtrować je, dodawać do koszyka, dokonywać płatności, przeglądać swoje zamówienia i edytować dane personalne na koncie. Opisane możliwości aktorów znajdują się w formie graficznej jako diagram przypadków użycia w rozdziale 3.3. “Diagram przypadków użycia”.

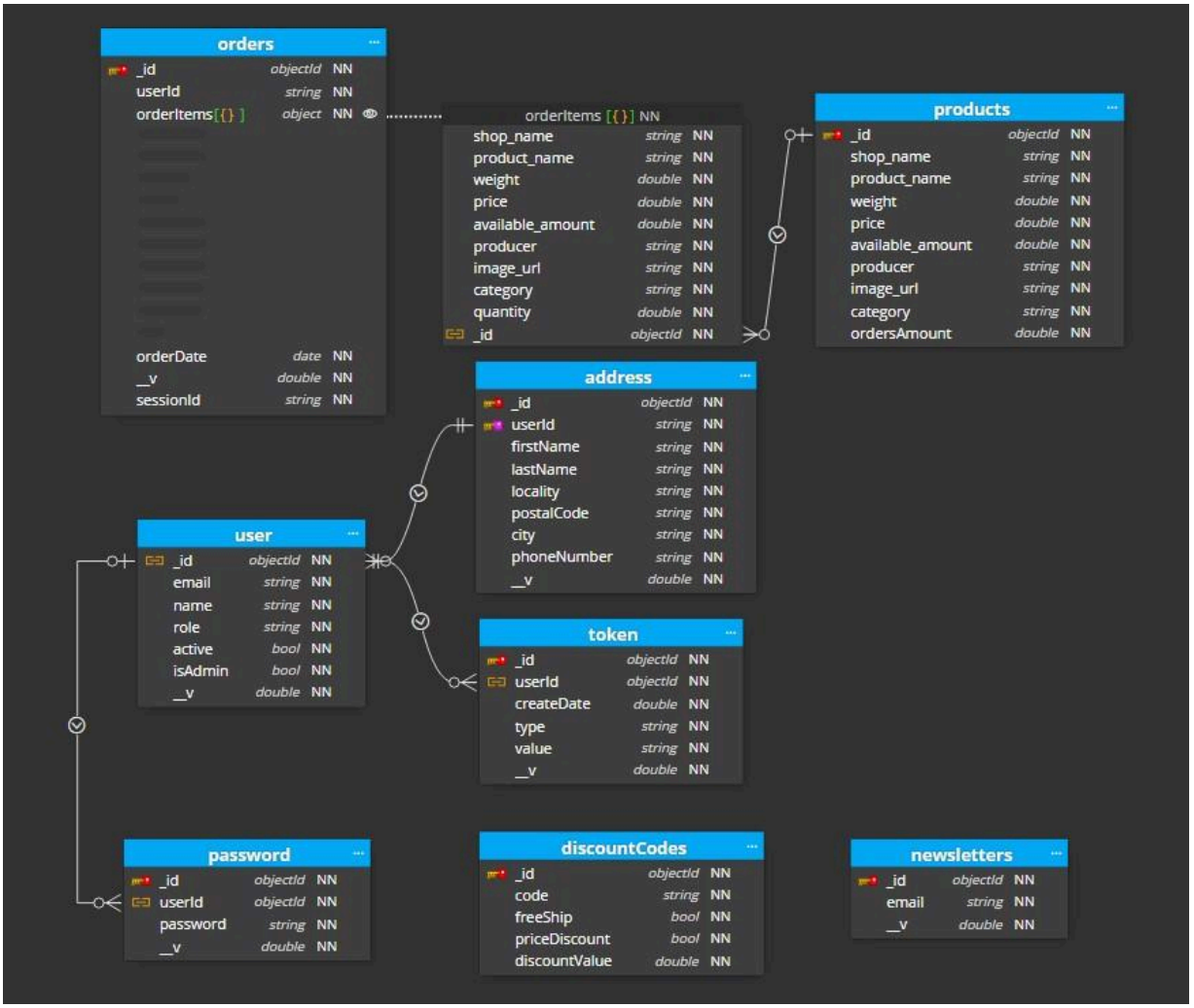
## 3.2. Diagram ERD

**ERD** (ang. *Entity-Relationship Diagram*) to graficzna metoda przedstawienia związków między encjami w projektowaniu systemów informacyjnych. Poniższy diagram znajdujący się na rysunku 3.1 przedstawia rozkład kolekcji znajdujących się w bazie **MongoDB**.

W kolekcji **orders** znajdują się dokumenty z informacjami o zamówieniach realizowanych przez użytkownika. Dotyczy to identyfikatora użytkownika, który złożył zamówienie, produktów, daty i identyfikatora sesji zamówienia. Dzięki identyfikatorowi sesji sprawdzany jest stan płatności, czy została ona zrealizowana. Pole **orderItems** to tablica składająca się z zamówionych produktów. Każdy artykuł jest opisany za pomocą nazwy sklepu, nazwy produktu, wagi, ceny, dostępności, producenta, url do zdjęcia, kategorii oraz ilości danego produktu w zamówieniu. Dodatkowe pole **ordersAmount** służy do sprawdzenia ilości sprzedanych produktów w sklepie.

W kolekcji “user” znajdują się podstawowe informacje na temat konta użytkownika. Dotyczy to jego identyfikatora, e-maila, nazwy, roli, aktywności konta oraz uprawnień. Za pomocą identyfikatora użytkownika łączą się kolekcje **password**, **address** oraz **token**. W pierwszej z nich jest hasło w zakodowanej postaci. Druga z wymienionych kolekcji przechowuje adres użytkownika. Kolekcja **token** zapisuje jednorazowy kod JWT. Dzięki niemu możemy sprawdzić czy sesja danego użytkownika na stronie jest aktywna.

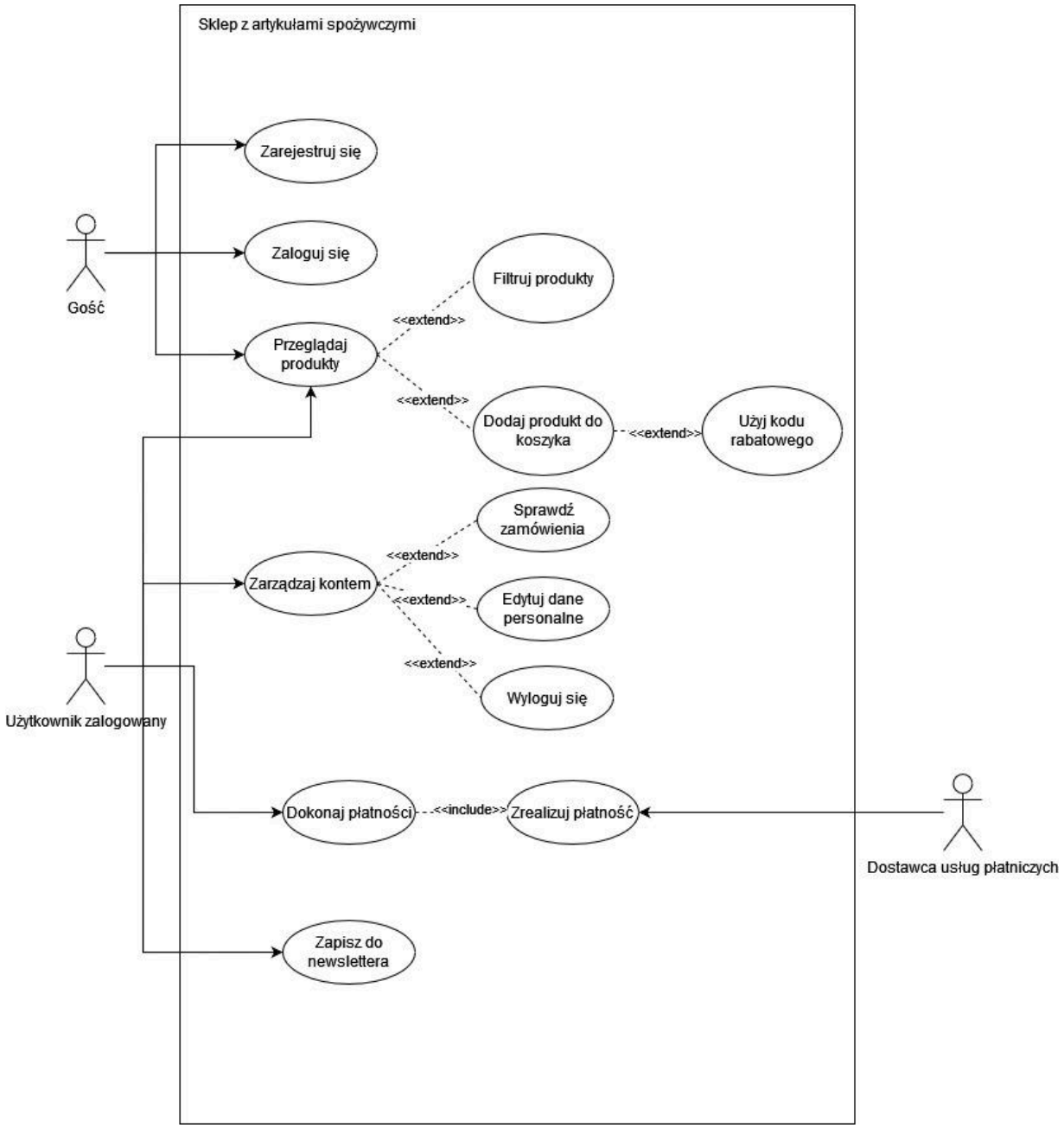
Dwie odrębne kolekcje to **discountCodes** oraz **newsletters**, które nie są powiązane w żaden sposób z innymi kolekcjami. W pierwszej wymienionej znajdują się kody rabatowe, a w drugiej e-maile użytkowników, którzy zgodzili się na otrzymywanie wiadomości z aplikacji.




**Rys. 3.1.** Diagram ERD [Moon Modeler]

## 3.3. Diagram przypadków użycia

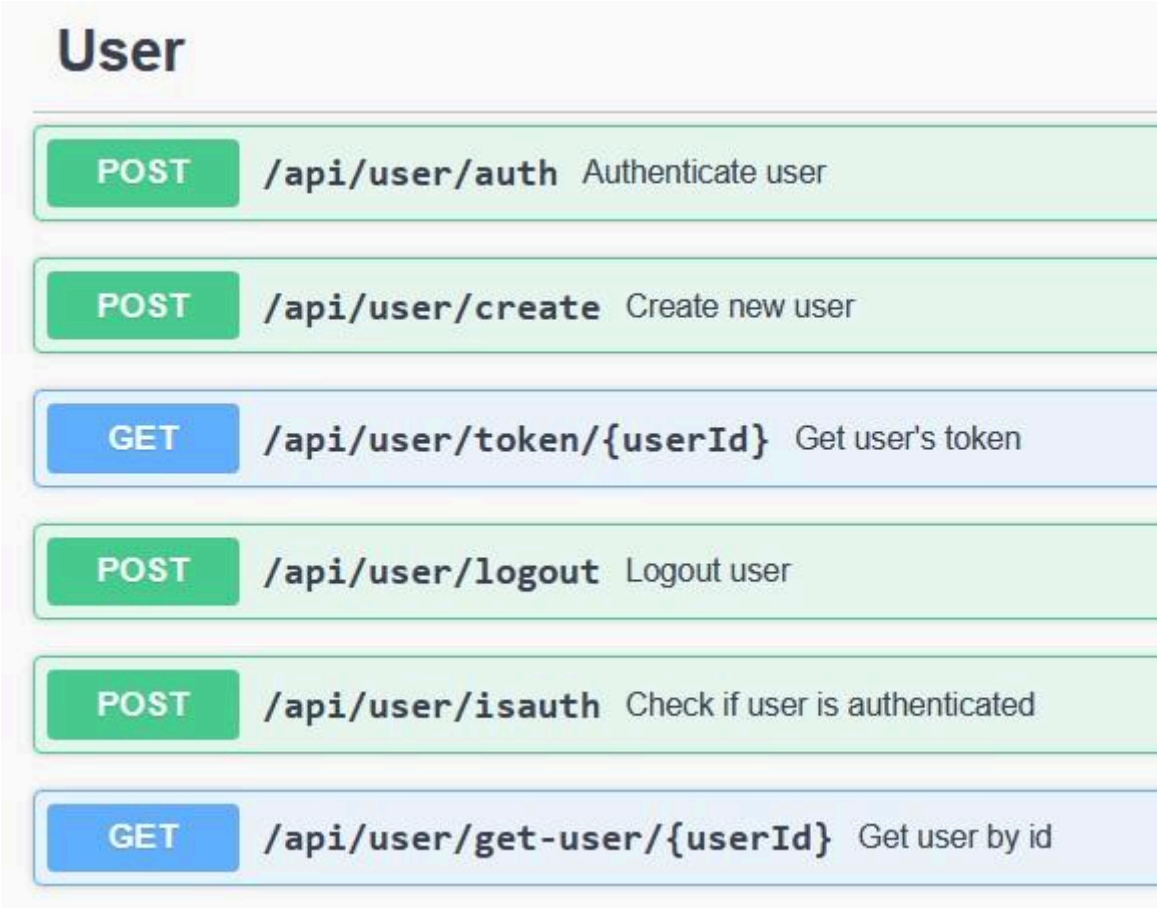
Diagram przypadków użycia przedstawia funkcjonalności, które mogą wykonywać aktorzy.




**Rys. 3.2.** Diagram przypadków użycia [opracowanie własne]

## 3.4. Dokumentacja API

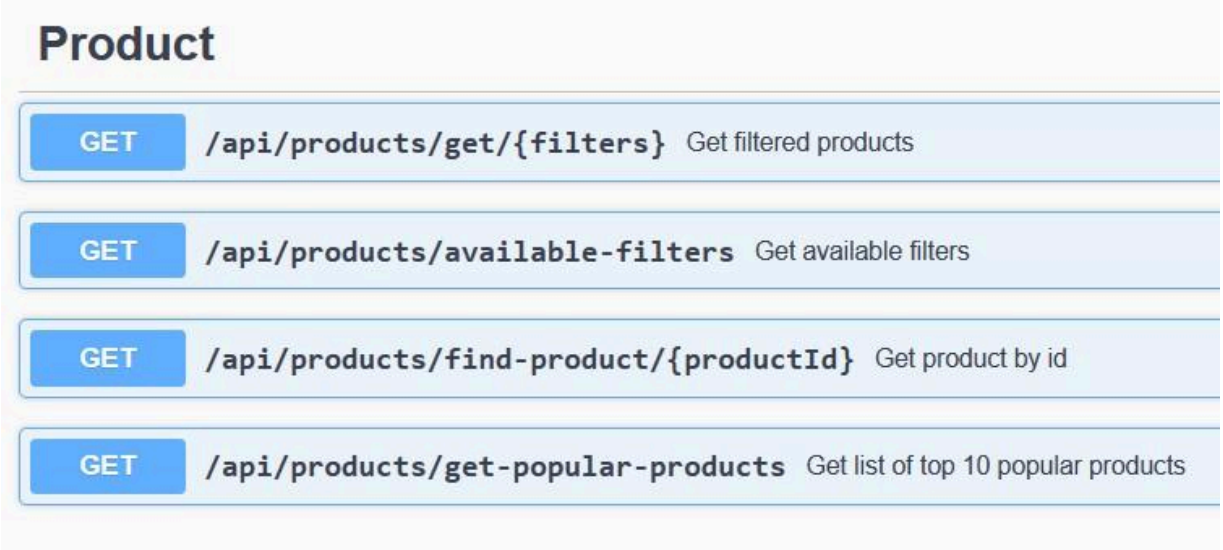
W rozdziale 3.4 znajduje się przedstawienie punktów końcowych API.




**Rys. 3.3.** Punkty końcowe dotyczące użytkownika [Swagger]

Na rysunku 3.3 przedstawione są punkty końcowe API dotyczące użytkowników.

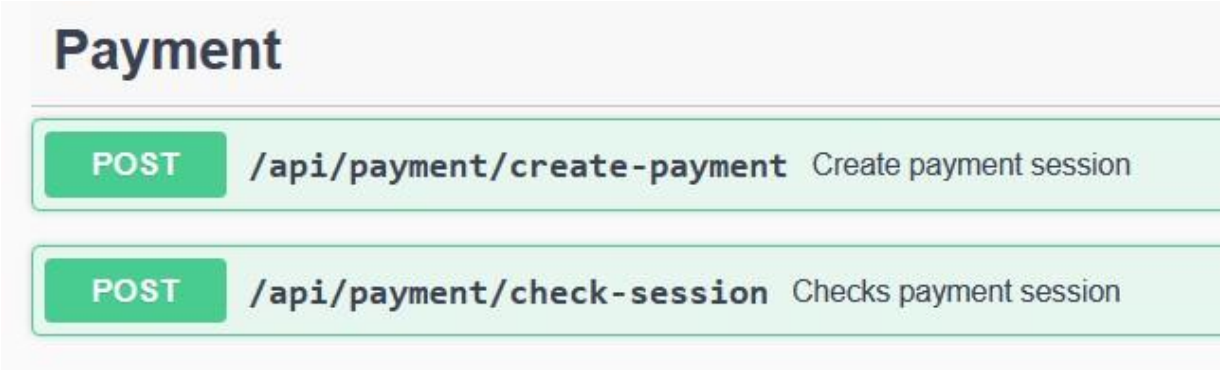
- **/api/user/auth** - dotyczy uwierzytelniania użytkownika. Używany jest podczas logowania się użytkownika do serwisu,
- **/api/user/create** - dotyczy tworzenia nowego użytkownika podczas rejestracji,
- **/api/user/token/{userId}** - zwraca token JWT użytkownika o podanym identyfikatorze,
- **/api/user/logout** - wylogowuje użytkownika z aplikacji,
- **/api/user/isauth** - sprawdza czy użytkownik jest zalogowany i czy jego token JWT jest ważny,
- **/api/user/get-user/{userId}** - zwraca dane użytkownika o podanym identyfikatorze.




**Rys. 3.4.** Punkty końcowe dotyczące produktów [Swagger]

Rysunek 3.4 dotyczy punktów końcowych związanych z produktami.

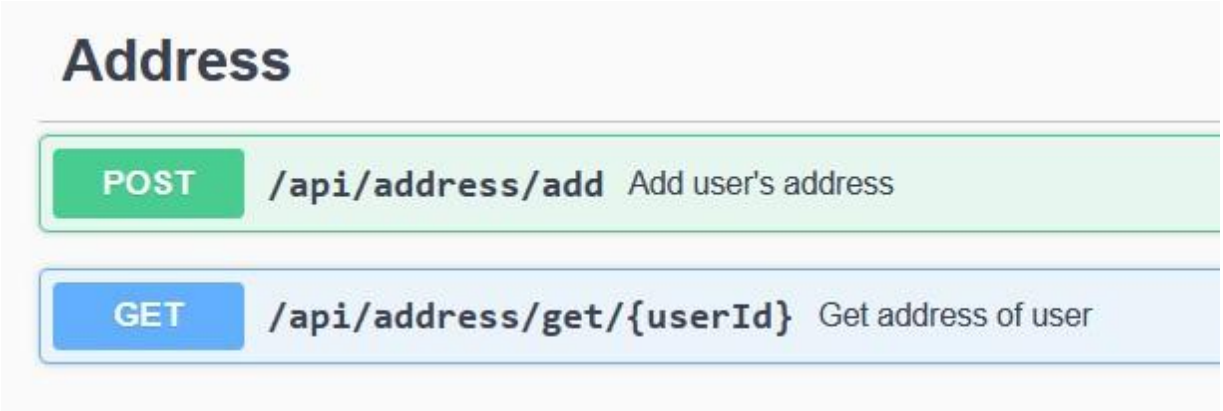
- **/api/products/get/{filters}** - zwraca produkty z zastosowaniem filtrów wybranych przez użytkowników,
- **/api/products/available-filters** - zwraca dostępne filtry, które użytkownik może zastosować podczas przeszukiwania produktów w aplikacji,
- **/api/products/find-product/{productId}** - zwraca produkt o podanym identyfikatorze,
- **/api/products/get-popular-products** - zwraca 10 najpopularniejszych produktów kupowanych przez użytkowników.




**Rys. 3.5.** Punkty końcowe dotyczące płatności [Swagger]

Na rysunku 3.5 znajdują się punkty końcowe dotyczące płatności.

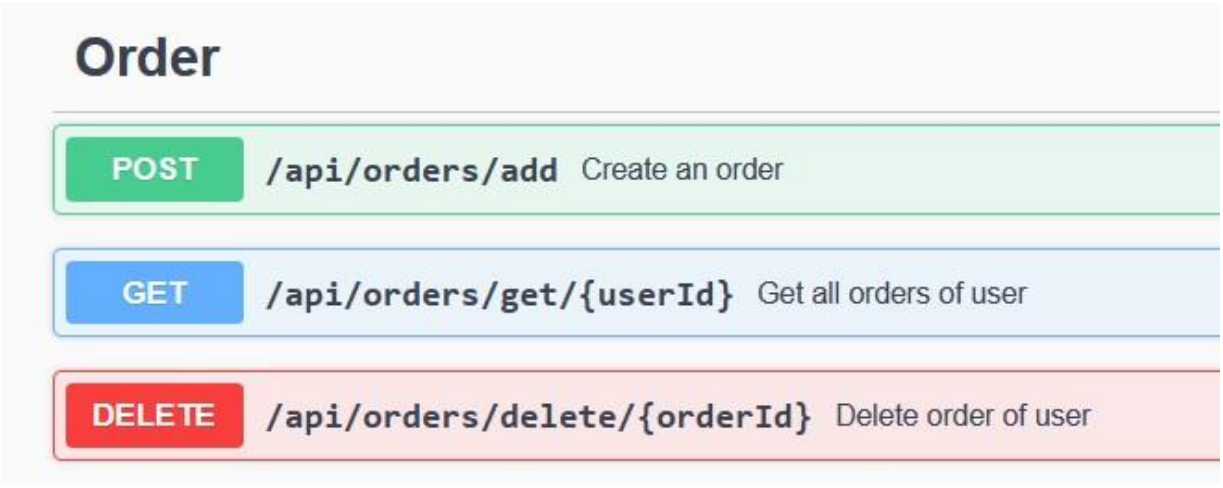
- **/api/payment/create-payment** - tworzy sesję płatności Stripe,
- **/api/payment/check-session** - sprawdza stan sesji płatności.




**Rys. 3.6.** Punkty końcowe dotyczące adresu [Swagger]

Rysunek 3.6 dotyczy punktów końcowych odnośnie adresu.

- **/api/address/add** - zapisuje adres użytkownika do bazy danych,
- **/api/address/get/{userId}** - zwraca adres użytkownika o podanym identyfikatorze.

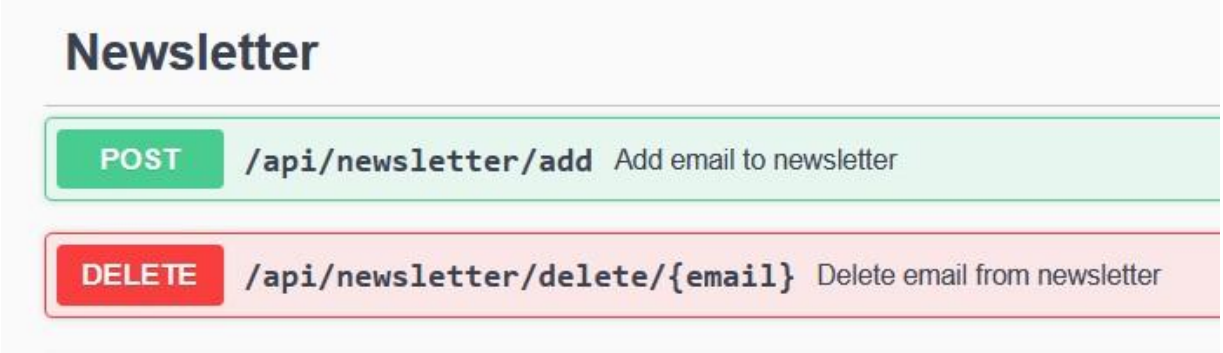



**Rys. 3.7.** Punkty końcowe dotyczące zamówień [Swagger]

Na rysunku 3.7 znajdują się punkty końcowe zamówień.

- **/api/orders/add** - tworzy zamówienie użytkownika i zapisuje je w bazie danych,
- **/api/orders/get/{userId}** - pobiera wszystkie zamówienia użytkownika o podanym identyfikatorze,

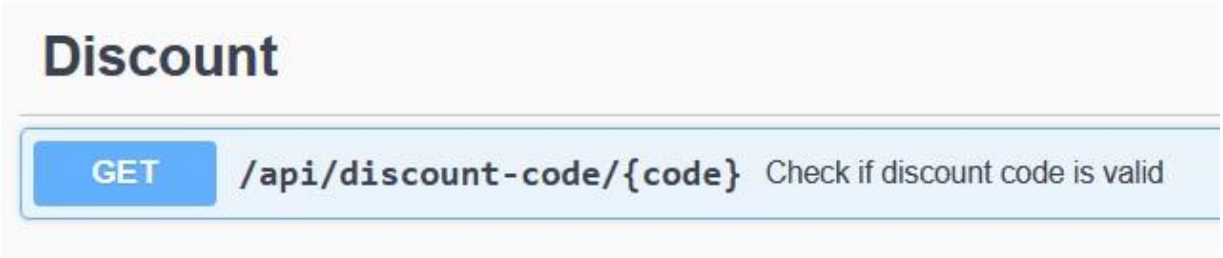
- **/api/orders/delete/{orderId}** - usuwa zamówienie użytkownika.




**Rys. 3.8.** Punkty końcowe dotyczące newslettera [Swagger]

Rysunek 3.8 przedstawia punkty końcowe dotyczące newslettera.

- **/api/newsletter/add** - dodaje adres e-mail użytkownika do newslettera,
- **/api/newsletter/delete/{email}** - usuwa adres e-mail użytkownika z newslettera.




**Rys. 3.9.** Punkty końcowe dotyczące kodów rabatowych [Swagger]

Na rysunku 3.9 znajduje się punkt kontrolny odnośnie kodów rabatowych.

- **/api/discount-code/{code}** - sprawdza czy podany kod rabatowy istnieje i go zwraca.

# 4. Interfejsy użytkownika

W rozdziale 4 przedstawione są interfejsy użytkownika z aplikacji internetowej.

## 4.1. Strona domowa

Strona główna składa się z 6 sekcji:

- Nagłówek,
- Wybór sklepu,
- Newsletter,
- Najpopularniejsze produkty,
- Informacja o promocjach,
- Stopka.

W nagłówku znajdują się kategorie, pole do wyszukiwania produktów, podgląd koszyka oraz konto. Są to najważniejsze elementy, które muszą znajdować się na każdej podstronie, aby użytkownik miał dobre doświadczenia z używania strony. Pozwala to na szybką obsługę strony i komfortowe przemieszczanie się po podstronach.

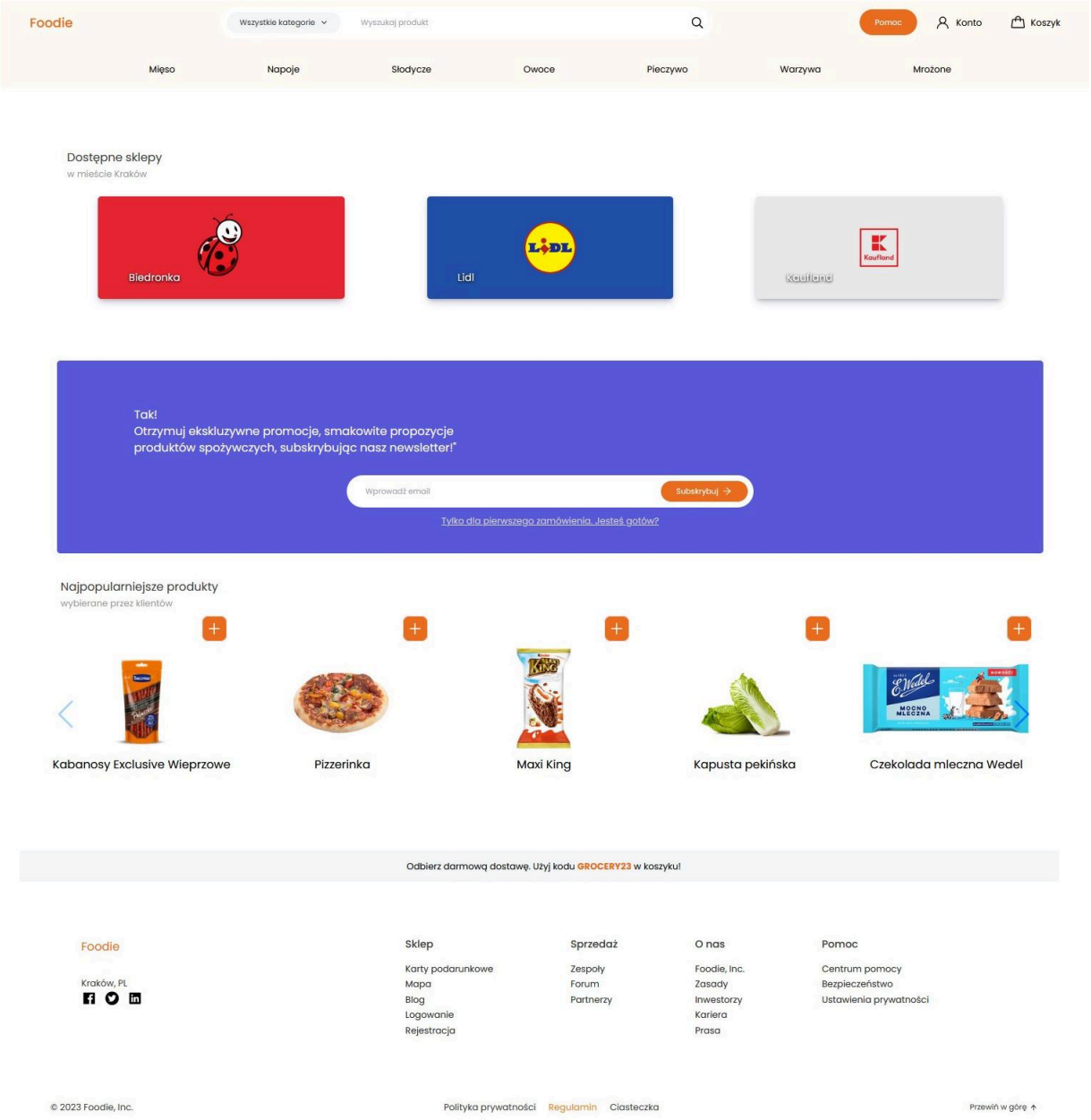
W kolejnej sekcji można wybrać w jakim sklepie użytkownik chce dokonać zakupów. Po naciśnięciu w wybrany sklep, klient zostanie przeniesiony do listy produktów gdzie może filtrować i wyszukiwać odpowiednie artykuły spożywcze.

Newsletter pozwala na przesłanie adresu e-mail do bazy danych w celu otrzymywania w przyszłości wiadomości oraz informacji o promocjach.

W sekcji z najpopularniejszymi produktami znajdują się najczęściej sprzedawane produkty w aplikacji. Można dzięki temu zobaczyć jakie produkty są najczęściej wybierane przez użytkowników.

Informacja o promocjach to “pasek” informacyjny zawierający aktualnie działający kod rabatowy na stronie.

W stopce znajdują się takie informacje jak socialmedia, odnośniki do przydatnych podstron oraz regulamin.




**Rys. 4.1.** Interfejs strony głównej [opracowanie własne]

### 4.2. Rejestracja i logowanie

Na stronie rejestracji użytkownik ma możliwość założenia konta w aplikacji. Do jego utworzenia musi uzupełnić 4 pola:

- Nazwę użytkownika,
- E-mail,
- Hasło,
- Powtórz hasło.

Po naciśnięciu w przycisk **Zarejestruj**, użytkownik uzyskuje swoje własne konto.


**Rys. 4.2.** Interfejs rejestracji [opracowanie własne]

Na stronie logowania klient ma możliwość zalogowania się na swoje konto po uprzednim założeniu go na podstronie do rejestracji. Po zalogowaniu się, użytkownik uzyskuje większe uprawnienia. W odróżnieniu od użytkowników niezalogowanych może dokonywać płatności i robić zakupy na stronie.


**Rys. 4.3.** Interfejs logowania [opracowanie własne]

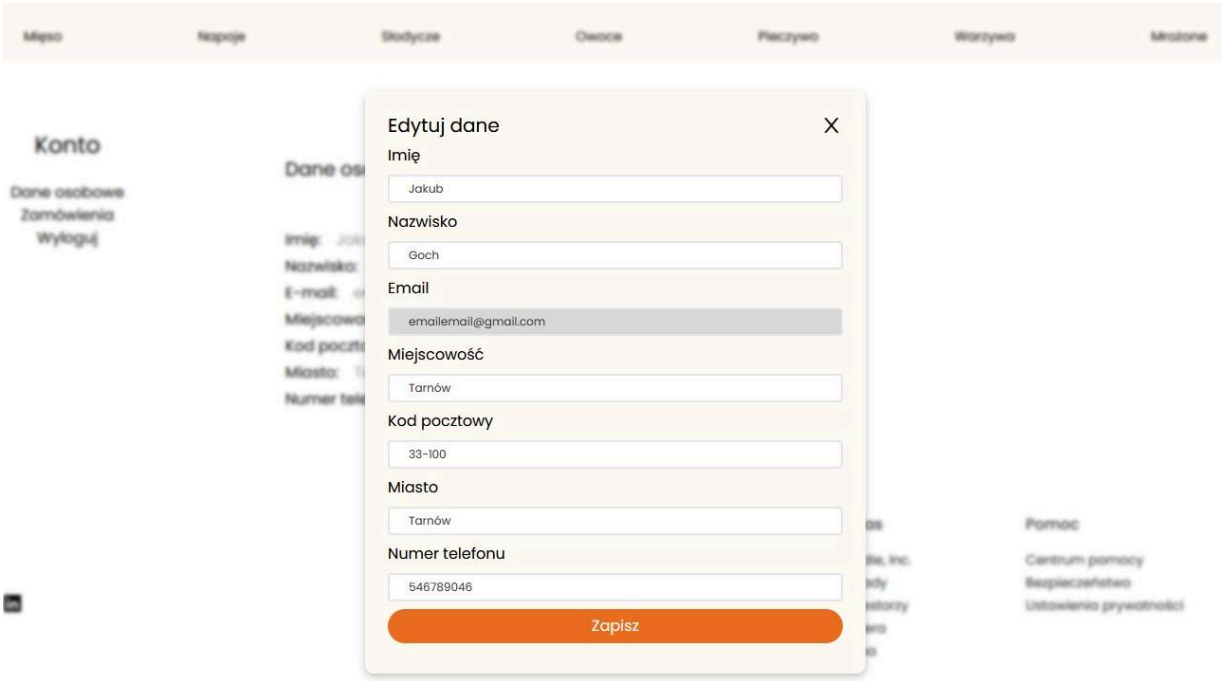
## 4.3. Panel konta użytkownika

Rozdział 4.3 przedstawia interfejsy użytkownika dotyczące panelu konta klienta.

### 4.3.1. Dane osobowe

Po wejściu na podstronę konta użytkownika można znaleźć menu z odnośnikami do podglądu danych osobowych zalogowanego klienta. Są to podstawowe informacje potrzebne do zrealizowania zamówienia złożonego przez użytkownika. Po prawej stronie znajduje się pomarańczowy przycisk z ikoną edycji, który służy do modyfikacji ustawionych danych. Podgląd edycji znajduje się na rysunku 4.5.


**Rys. 4.4.** Interfejs konta użytkownika - dane osobowe [opracowanie własne]

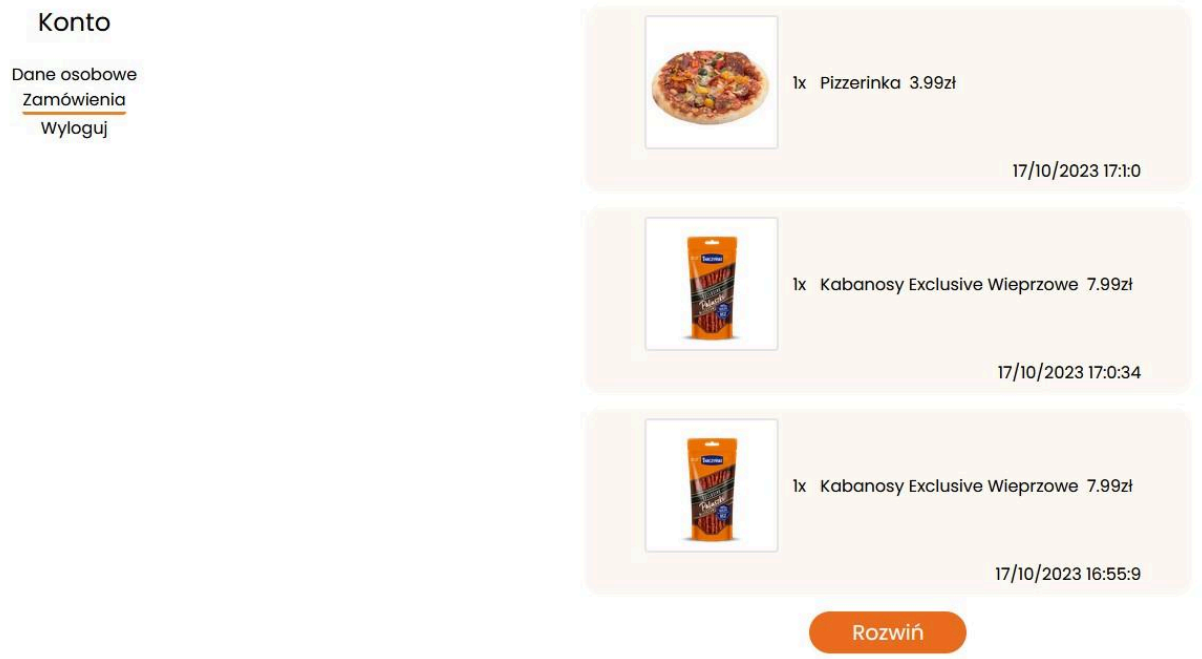



**Rys. 4.5.** Interfejs konta użytkownika - edycja danych osobowych [opracowanie własne]

W oknie edycji danych użytkownika, klient może zmienić aktualnie ustawione informacje.

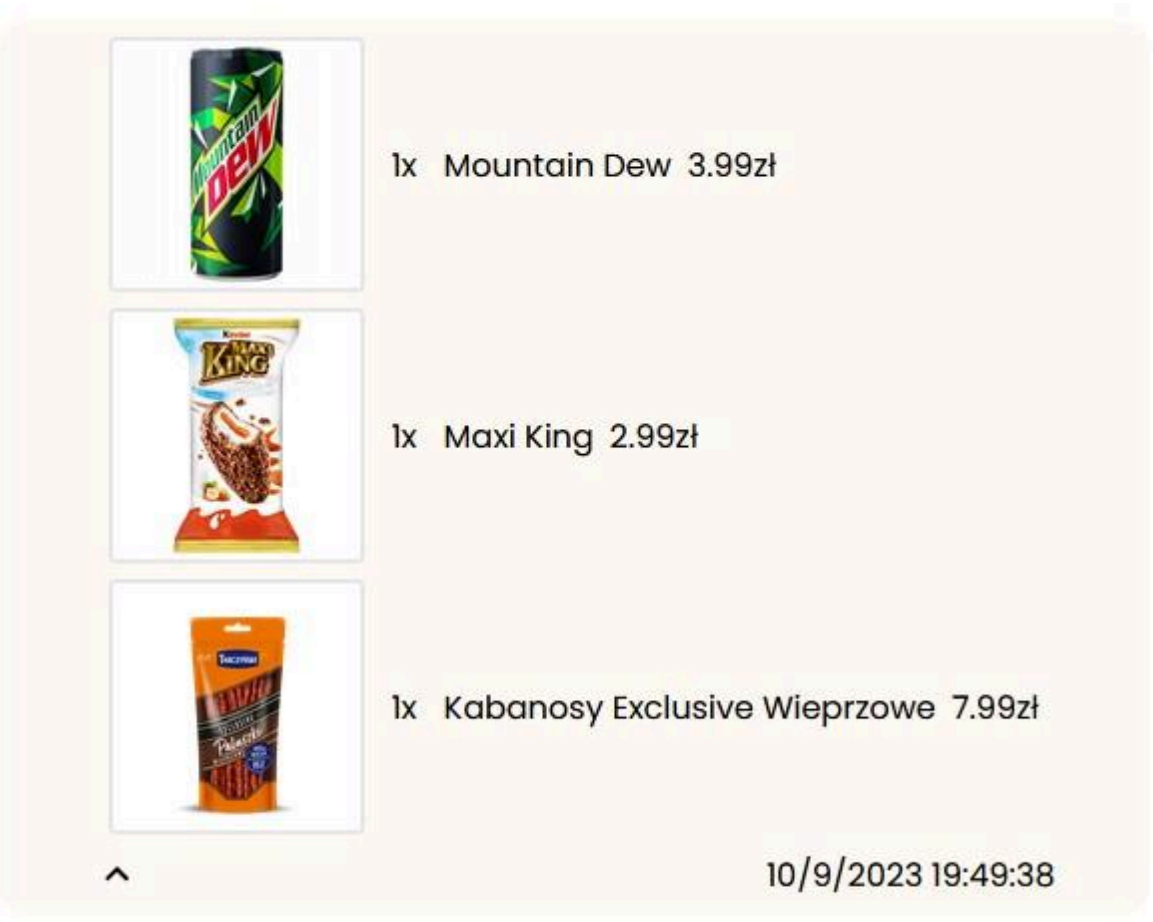
### 4.3.2. Zamówienia

W sekcji związanej z podglądem zamówień użytkownik może zobaczyć swoje zrealizowane zakupy. Przycisk “Rozwiń” powoduje powiększenie się listy o resztę zamówień, które są ukryte. Każde z zamówień zawiera informacje o produktach, ich ilości oraz dacie złożonego zamówienia.




**Rys. 4.6.** Interfejs konta użytkownika - podgląd zamówień [opracowanie własne]

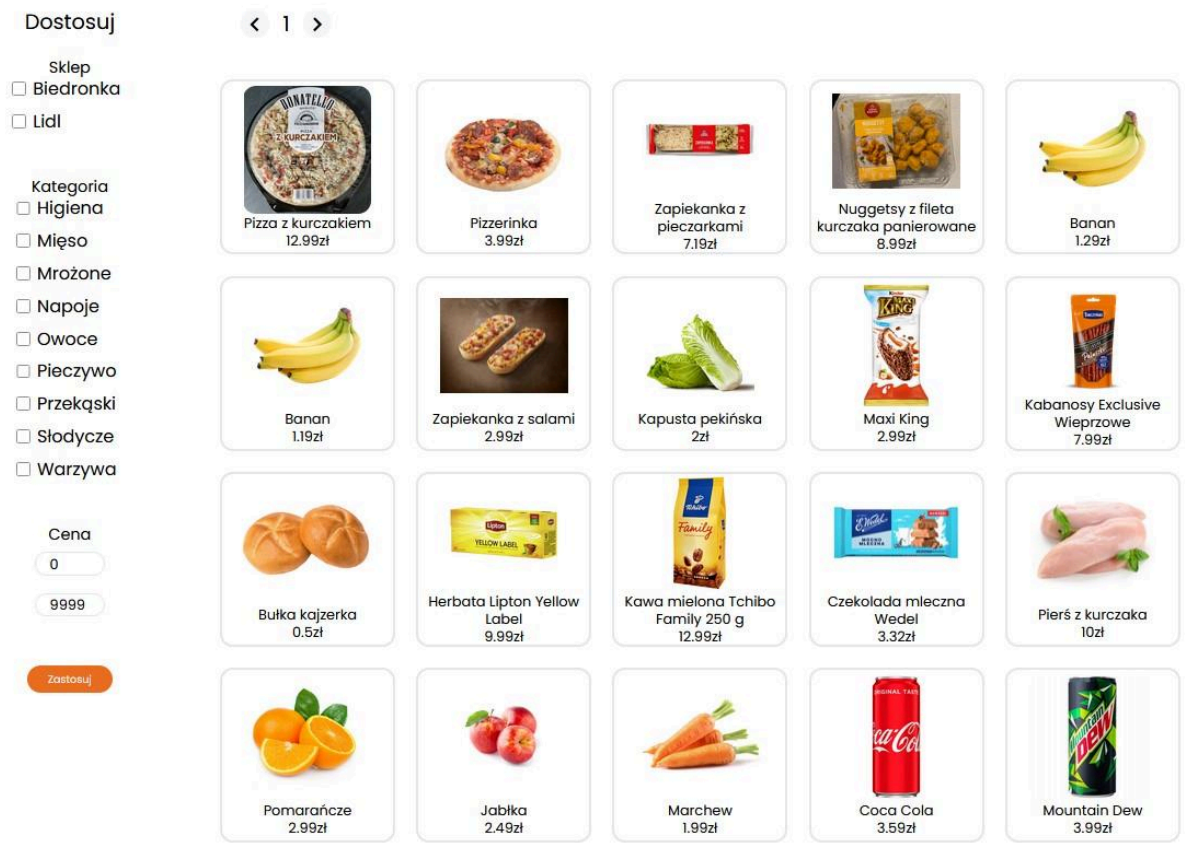
Jeżeli zamówienie zawiera więcej niż jeden produkt, pojawia się dodatkowa ikona strzałki, która pozwala rozwinąć wybrane zamówienie i zobaczyć resztę artykułów. Przykład interfejsu tego rozwiązania znajduje się na rysunku 4.7.




**Rys. 4.7.** Interfejs konta użytkownika - rozwinięte zamówienie [opracowanie własne]

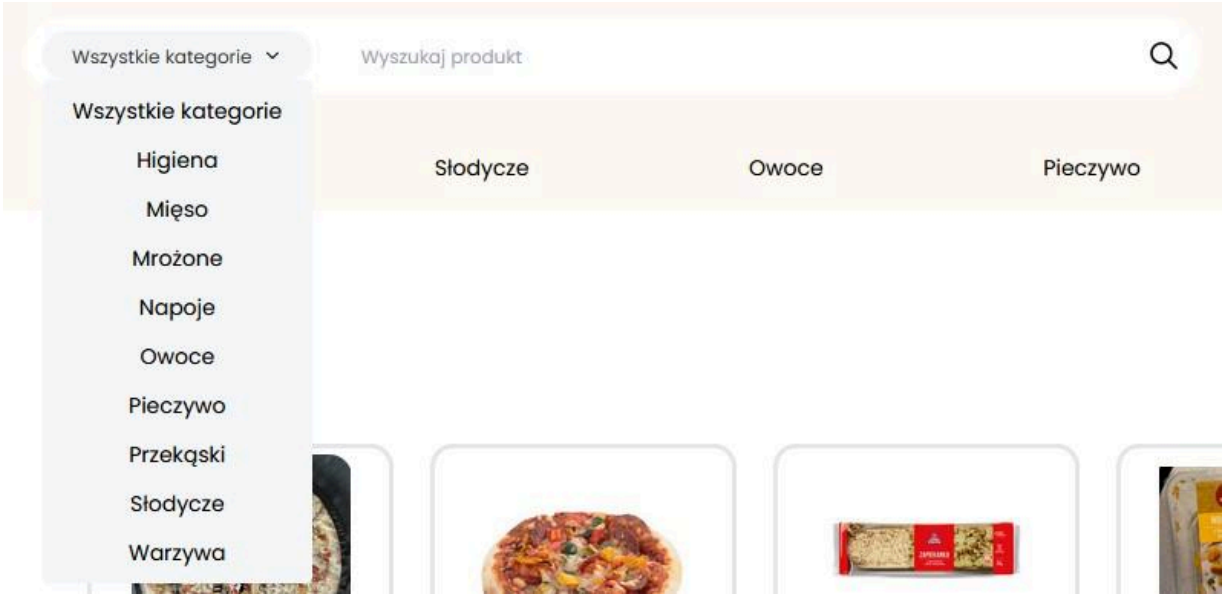
## 4.4. Przeglądanie produktów

Aplikacja umożliwia wygodne przeglądanie produktów. Na rysunku 4.8 przedstawiony jest interfejs listy artykułów w aplikacji. Po lewej stronie znajduje się sekcja z filtrami, która umożliwia klientowi znalezienie odpowiedniego produktu w dużo szybszy sposób. Użytkownik może filtrować produkty w zależności od sklepu, w którym się znajdują, ceny oraz kategorii. Po prawej stronie znajdują się dostępne artykuły. Zastosowana została paginacja, czyli podział wszystkich produktów na mniejsze części (strony), które można zmieniać przy pomocy interfejsu paginacji znajdującego się nad listą produktów.




**Rys. 4.8.** Interfejs przeglądania produktów - lista produktów i filtry [opracowanie własne]

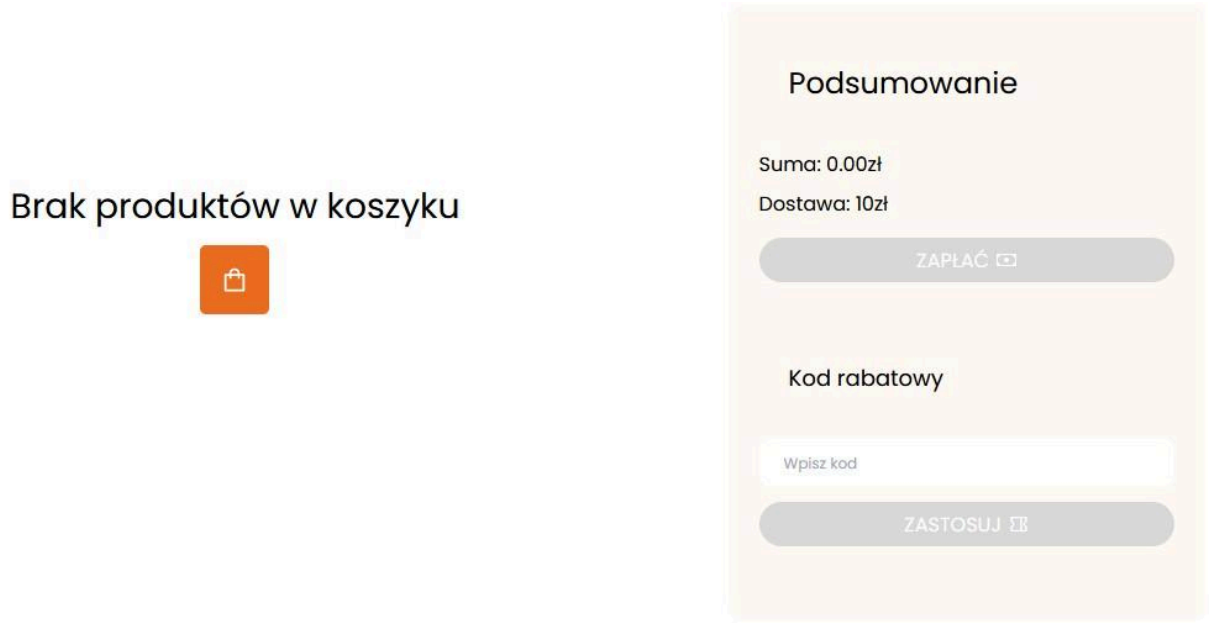
W nagłówku aplikacji znajduje się wyszukiwarka produktów przedstawiona na rysunku 4.9. Klient może wpisać nazwę produktu, którego szuka i nacisnąć w ikonę lupy. Interfejs umożliwia również wybranie kategorii, w której artykuł powinien być przeszukiwany.




**Rys. 4.9.** Interfejs przeglądania produktów - wyszukiwanie po nazwie [opracowanie własne]

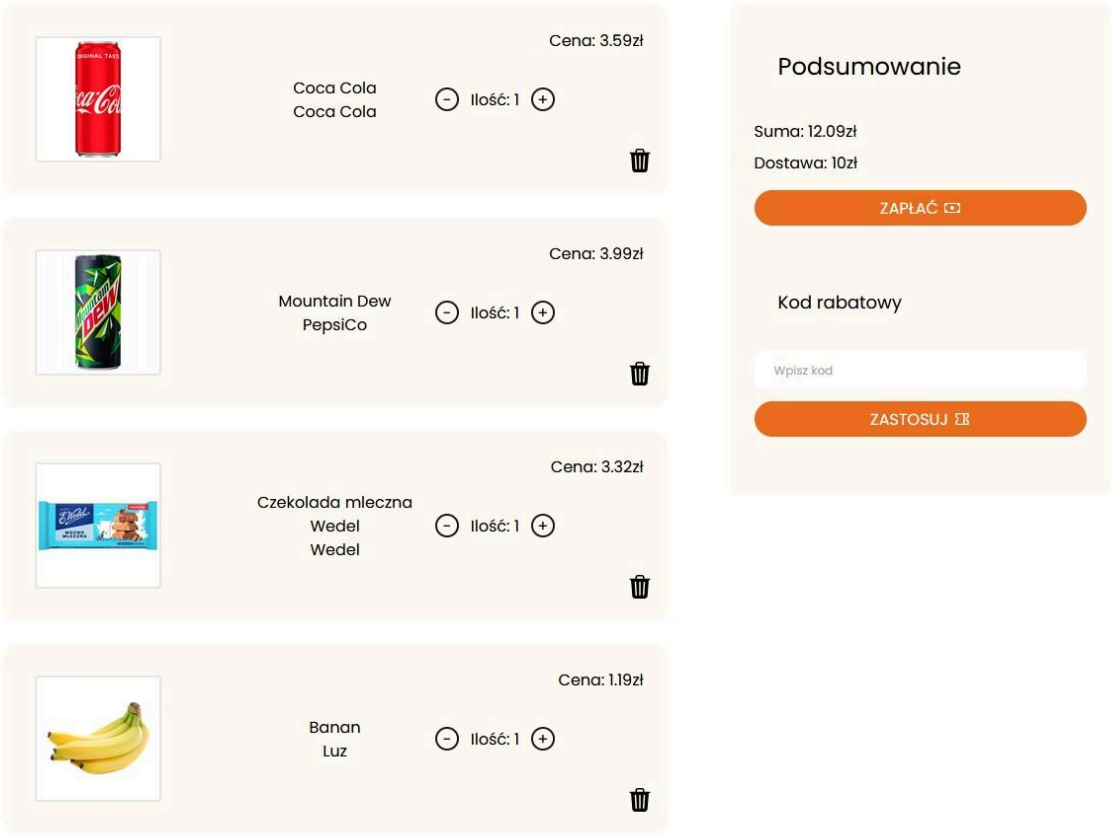
## 4.5. Koszyk użytkownika

Nowi użytkownicy w aplikacji po wejściu w koszyk zobaczą interfejs koszyka, który nie zawiera żadnych produktów. Jest on przedstawiony na rysunku 4.10. Przyciski umożliwiające przejście do strony płatności i użycie kodu rabatowego będą zablokowane aż do momentu dodania chociaż jednego artykułu do koszyka. Cena za produkty powinna wskazywać 0zł.




**Rys. 4.10.** Interfejs koszyka - brak produktów [opracowanie własne]

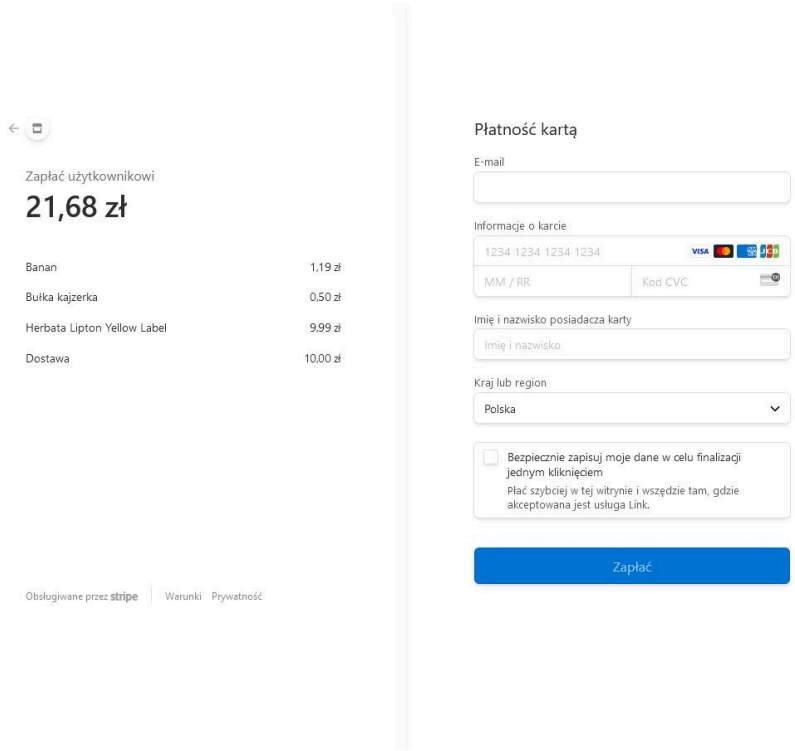
Gdy klient wybierze produkty, które chce zakupić i doda je do koszyka, interfejs zmieni się, a przyciski, które wcześniej były zablokowane teraz będą możliwe do użycia. Wybrane artykuły pojawią się również w interfejsie, dzięki któremu można na nich operować. W koszyku widoczne są nazwy produktów, zdjęcie oraz ich cena. Umożliwia on zwiększenie lub zmniejszenie ilości danego artykułu w koszyku. Można również całkowicie go usunąć.



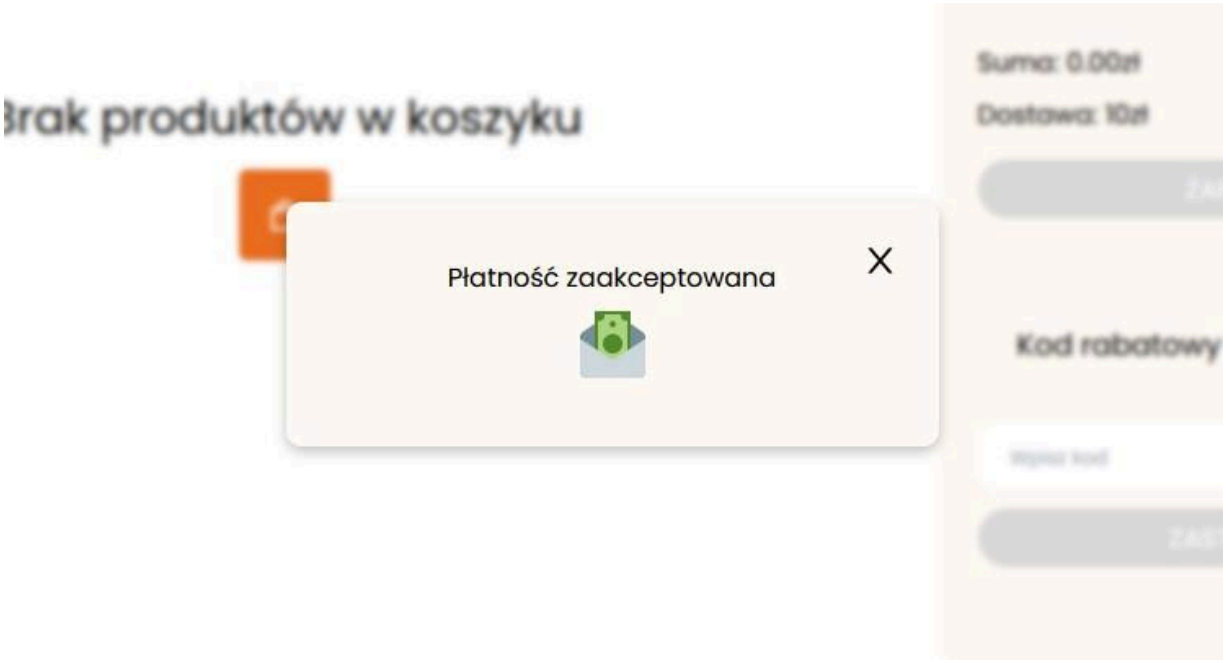

**Rys. 4.11.** Interfejs koszyka - z wybranymi produktami [opracowanie własne]

## 4.6. Strona płatności

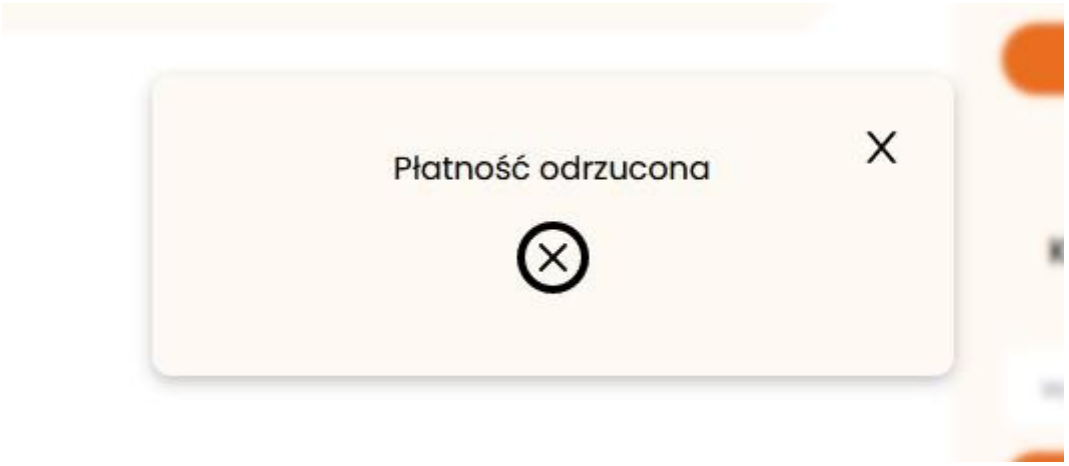
Aplikacja zapewnia dokonywanie płatności za pośrednictwem usługi Stripe. Po dodaniu produktów do koszyka oraz naciśnięciu w przycisk umożliwiający płatność, klient zostaje przekierowany na stronę płatności Stripe. Możliwa metoda zapłaty to karta debetowa lub kredytowa. Po wpisaniu danych płatniczych i zaakceptowaniu płatności użytkownik przekierowany zostaje z powrotem do aplikacji sklepu i otrzymuje informację o tym, czy zamówienie zostało zaakceptowane lub odrzucone.




**Rys. 4.12.** Interfejs płatności - Stripe [opracowanie własne]




**Rys. 4.13.** Interfejs płatności - płatność zaakceptowana [opracowanie własne]




**Rys. 4.14.** Interfejs płatności - płatność odrzucona [opracowanie własne]

# 5. Wybrane testy

W rozdziale 5 znajdują się testy zastosowane w aplikacji w celu sprawdzenia poprawności działania komponentów, funkcji i punktów końcowych API.

## 5.1. Testy jednostkowe

Rozdział 5.1 dotyczy testów jednostkowych, czyli sprawdzania poprawności mniejszych części kodu w aplikacji. Do przeprowadzenia tych testów wykorzystane zostały narzędzia **JEST** i **React Testing Library**.

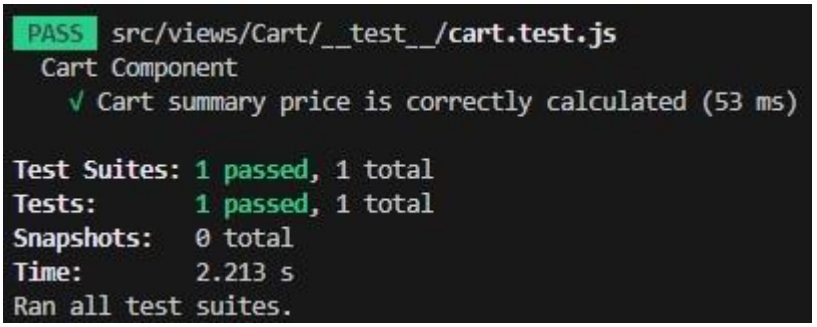
### 5.1.1. Test wartości koszyka

W koszyku użytkownika po dodaniu do niego produktów pojawia się cena finalna wszystkich produktów znajdujących się w koszyku. Kod na rysunku 5.1 przedstawia test sprawdzający, czy cena całego koszyka zgadza się z wartością poszczególnych artykułów.

```
it('Cart summary price is correctly calculated', () => {
    const cart = [
        {id: '64d9fda84e78cdba9b181e75', product_name: 'Banan', price: 1.29, quantity: 1},
        {id: '6505edeedbe1e77026a89a69', product_name: "Czekolada mleczna Wedel", price: 3.32, quantity: 1}
    ];
    const expectedTotalPrice = cart.reduce((total, product) => total + product.price, 0).toFixed(2);

    render(<MemoryRouter><CartSummary cartProducts={cart} setCartProducts={() => {}} setShowPaymentPopup={() => {}}/></MemoryRouter>);

    const sumText = screen.getByTestId('sum-text').textContent;
    expect(sumText).toEqual('Suma: ' + expectedTotalPrice + 'zł');
})
```

**Rys. 5.1.** Test jednostkowy - wartość koszyka [opracowanie własne]

Na początku utworzona została stała **cart**, która zawiera tablicę dwóch przykładowych produktów oraz ich cen i ilości. W stałej **expectedTotalPrice** wyliczana jest wartość pieniężna produktów znajdujących się w **cart**. Następnie w teście załączany jest komponent **CartSummary**, który zawiera informację o wartości koszyka, cenie dostawy itp. W propsach, czyli właściwościach komponentów służących do przekazywania do nich danych przesłane zostały przykładowe produkty w propsie **cartProducts** oraz niezbędne funkcje potrzebne do działania komponentu. Końcowo za pomocą funkcji **expect** sprawdzana zostaje wartość wyświetlana w elemencie z identyfikatorem testowym **sum-text**. Test sprawdza, czy pokrywa się ona z cenami produktów znajdującymi się w koszyku. Rysunek 5.2 przedstawia wynik opisanego testu.




**Rys. 5.2.** Test jednostkowy - wynik uruchomienia testu wartości koszyka [opracowanie własne]

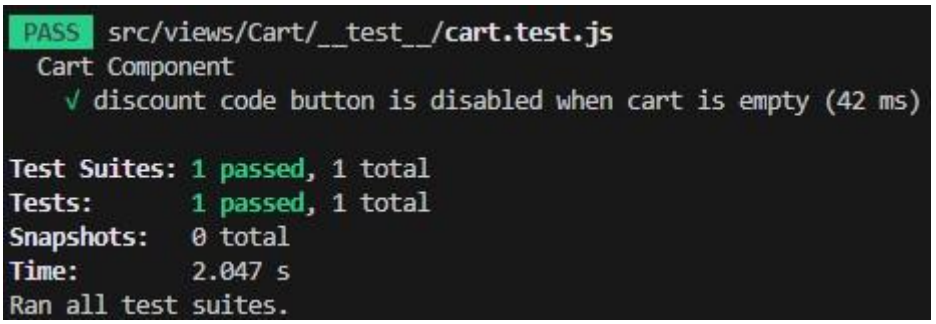
### 5.1.2. Test dezaktywacji przycisku koszyka

Jeżeli użytkownik nie dodał żadnych produktów do swojego koszyka to po wejściu do niego powinien mieć zablokowane niektóre funkcje. Dotyczy to przycisków umożliwiających przejście do strony płatności jak i przycisku do użycia kodu rabatowego. Odblokowane są one dopiero w momencie, w którym klient dodał chociaż jeden produkt do koszyka. Rysunek 5.3 przedstawia kod testu sprawdzającego, czy przycisk kodu rabatowego jest zablokowany przy braku artykułów w koszyku.

```
it('discount code button is disabled when cart is empty', () => {
    const emptyCart = [];

    render(<MemoryRouter><CartSummary cartProducts={emptyCart} setCartProducts={() => {}} setShowPaymentPopup={() => {}}/></MemoryRouter>);

    const discountButton = screen.getByTestId('discount-button');
    expect(discountButton).toBeDisabled();
})
```

**Rys. 5.3.** Test jednostkowy - dezaktywacja przycisku kodu rabatowego [opracowanie własne]

Wstępnie utworzona została wymagana stała **emptyCart**, która jest pustą tablicą. Oznacza to, że żaden produkt nie zostanie przekazany do komponentu koszyka. Następnie załączony zostaje komponent **CartSummary**. Jako propsy przekazana zostaje pusta tablica produktów oraz wymagane funkcje. Test sprawdza czy element z identyfikatorem testowym **discount-button** jest dezaktywowany. Rysunek 5.4 przedstawia wynik testu.




**Rys. 5.4.** Test jednostkowy - wynik testu dezaktywacji przycisku kodu rabatowego [opracowanie własne]

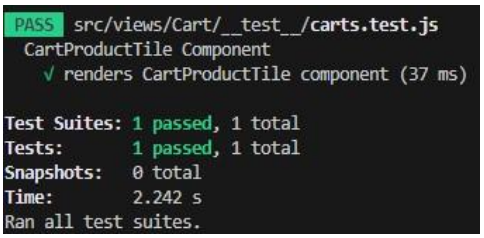
### 5.1.3. Test wyświetlania produktu w koszyku

Po dodaniu przez użytkownika produktu do koszyka, powinien on pojawić się w koszyku. Rysunek 5.5 przedstawia kod testu sprawdzającego, czy dodany produkt wyświetla się w koszyku.

```
describe('CartProductTile Component', () => {

  test('renders CartProductTile component', () => {
    const mockProduct = {
      id: '6505ee08dbe1e77026a9162c',
      product_name: 'Coca Cola',
      producer: 'Biedronka',
      price: 3.59,
      image_url: 'https://media.istockphoto.com/id/458464735/pl/zdj%C4%99cie/coca-cola.jpg?s=612x612&w=0&k=20&c=EGsWdrE-xvvBGNcAqYwOsvyFOCMKH_9TJ91X4nkwVVk=',
      quantity: 2
    };

    render(<MemoryRouter><CartProductTile product={mockProduct} setCartProducts={() => {}} /></MemoryRouter>);

    const productName = screen.getByText(/Coca Cola/i);
    expect(productName).toBeInTheDocument();
  });

});
```

**Rys. 5.5.** Test jednostkowy - wyświetlanie produktu w koszyku [opracowanie własne]

Na początku załadowany zostaje komponent **CartProductTile**, który zawiera propsy **product** z przykładowym produktem oraz wymaganą funkcją. Test sprawdza czy w komponencie **CartProductTile** wyświetla się przekazany produkt z nazwą **Coca Cola**. Rysunek 5.6 przedstawia wynik testu.




**Rys. 5.6.** Test jednostkowy - wynik testu wyświetlania produktu w koszyku [opracowanie własne]

## 5.2. Testy integracyjne

Rozdział 5.2 dotyczy testów integracyjnych, czyli sprawdzania poprawności działania współpracujących ze sobą części kodu w aplikacji. Do przeprowadzenia tych testów wykorzystane zostały narzędzia **JEST** i **Supertest**.

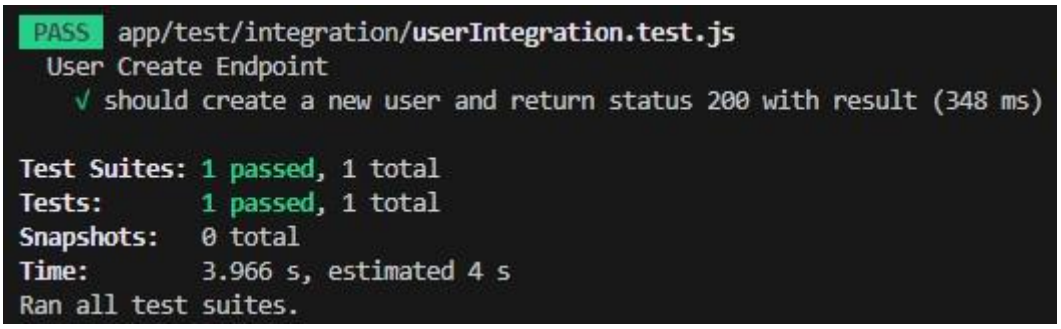
### 5.2.1. Utworzenie konta nowego użytkownika

Test przedstawiony na rysunku 5.7 łączy się z serwerem przy użyciu narzędzia **Supertest** i wysyła żądanie metodą **POST** do punktu końcowego **/api/user/create** przesyłając do niego dane nowego użytkownika do założenia mu konta w systemie. Następnie przy użyciu **expect** z technologii **JEST** sprawdzany jest rezultat żądania. Jeżeli status będzie równy 200 i zwrócony zostanie obiekt zawierający wartości identyfikatora, hasła i identyfikatora użytkownika to oznacza, że użytkownik został pomyślnie utworzony.

```
describe('User Create Endpoint', () => {
  it('should create a new user and return status 200 with result', async () => {

    const response = await request(server)
      .post('/api/user/create')
      .send({
        name: 'newuser25',
        email: 'newuser25@example.com',
        password: 'newpassword',
        confirmpassword: 'newpassword'
      });

    expect(response.status).toBe(200);
    expect(response.body).toEqual(expect.objectContaining({
      id: expect.any(String),
      password: expect.any(String),
      userId: expect.any(String),
    }));
  });
});
```

**Rys. 5.7.** Test integracyjny - utworzenie użytkownika [opracowanie własne]

Rysunek 5.8 przedstawia wynik testu dotyczącego tworzenia nowego użytkownika w systemie.




**Rys. 5.8.** Test integracyjny- wynik uruchomienia testu utworzenia użytkownika [opracowanie własne]

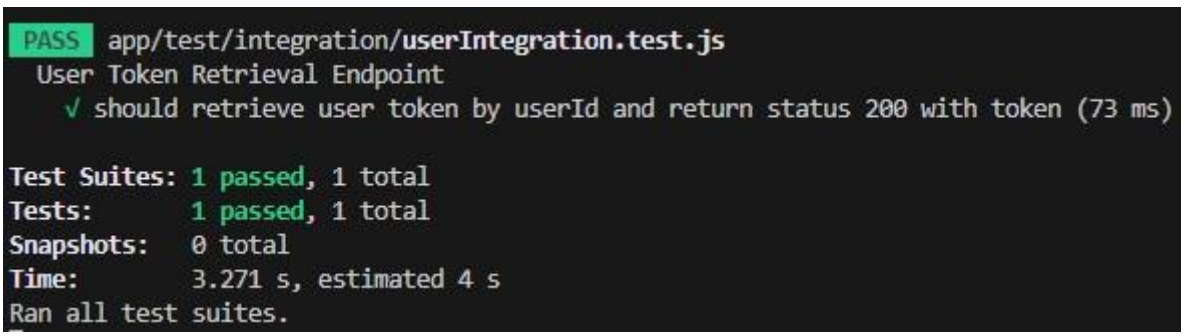
### 5.2.2. Pobranie tokenu JWT użytkownika

Test przedstawiony na rysunku 5.9 dotyczy pobrania tokenu **JWT** przy użyciu identyfikatora użytkownika za pomocą żądania **HTTP** metodą **GET** wysłanego do punktu końcowego **/api/user/token?userId=${userId}**. Rezultatem tego testu powinno być zwrócenie statusu o wartości 200 i danych, które zawierają token użytkownika.

```
describe('User Token Retrieval Endpoint', () => {
  it('should retrieve user token by userId and return status 200 with token', async () => {
    const userId = '64fcb422df2deec02079dfa5';

    const response = await request(server)
      .get(`/api/user/token?userId=${userId}`);

    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('value');
  });
});
```

**Rys. 5.9.** Test integracyjny- pobranie tokenu JWT użytkownika [opracowanie własne]

Wynik testu pobrania tokenu użytkownika przedstawiony jest na rysunku 5.10.




**Rys. 5.10.** Test integracyjny- wynik uruchomienia testu pobrania tokenu JWT użytkownika [opracowanie własne]

### 5.2.3. Wylogowanie użytkownika

Test przedstawiony na rysunku 5.11 dotyczy wylogowania użytkownika przy użyciu identyfikatora użytkownika za pomocą żądania **HTTP** metodą **POST** wysłanego do punktu końcowego **/api/user/logout**. Rezultatem tego testu powinno być zwrócenie statusu o wartości 200 i danych zawierających informację **deletedCount** równą 1.

```
describe('User Logout Endpoint', () => {
  it('should logout user and return status 200 with result', async () => {
    const userId = '64fcb422df2deec02079dfa5';

    const response = await request(server)
      .post('/api/user/logout')
      .send({ userId: userId });

    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('deletedCount', 1);
  });
});
```

**Rys. 5.11.** Test integracyjny - wylogowanie użytkownika [opracowanie własne]

Rysunek 5.12 przedstawia wynik testu wylogowania użytkownika o podanym identyfikatorze użytkownika.

```
PASS app/test/integration/userIntegration.test.js
  User Logout Endpoint
    √ should logout user and return status 200 with result (101 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        3.646 s, estimated 4 s
Ran all test suites.
```

**Rys. 5.12.** Test integracyjny- wynik uruchomienia testu wylogowania użytkownika [opracowanie własne]

# 6. Podsumowanie i wnioski

Celem pracy dyplomowej było utworzenie aplikacji webowej umożliwiającej zakupy artykułów spożywczych. W rozdziale 3.1 “Zakres funkcji” przedstawione były uprawnienia, które mają użytkownicy zalogowani i niezalogowani. Wszystkie z tych wymagań zostały spełnione:

- rejestracja,
- logowanie,
- przeglądanie produktów,
- filtrowanie produktów,
- dodawanie produktów do koszyka,
- dokonywanie płatności,
- przeglądanie zamówień,
- edycja danych osobowych na koncie.

Jest wiele możliwości rozwoju aplikacji. Świetnym pomysłem byłoby wprowadzenie programu lojalnościowego, który opierałby się na systemie punktów lojalnościowych. Dzięki temu klienci zbieraliby punkty, które później mogliby wymienić na zniżki w aplikacji. Kolejną sugestią odnośnie rozwoju strony jest aplikacja mobilna. Klienci mieliby możliwość użytkowania aplikacji na swoim telefonie bez użycia przeglądarki.

Aby usprawnić zaimplementowane już funkcje w aplikacji, dobrym pomysłem byłoby powiększenie zakresu metod płatności. Na ten moment płatności można dokonać przy użyciu karty debetowej lub kredytowej. Aby zmonetyzować aplikację można wprowadzić reklamy na stronie. Firmy mogłyby promować swoje produkty lub usługi na platformie. Kolejnym pomysłem, który byłby głównym dochodem z aplikacji to dodanie prowizji od każdego zamówienia. Użytkownik musiałby wtedy zapłacić za produkty, dostawę i dodatkową niewielką kwotę w postaci prowizji od zrealizowania usługi.

Utworzenie aplikacji do zakupów artykułów spożywczych pozwoliło rozwinąć umiejętności tworzenia aplikacji webowych oraz zapoznać się z nowymi technologiami.

# Bibliografia

[1] Adam Sosiński, *MongoDB – idealny system bazodanowy dla e-commerce?*
https://inetum.pl/mongodb-nosql-w-ecommerce/ (dostęp 20.10.2023)

[2] Eve Porcello, Alex Banks, *React od podstaw. Nowoczesne wzorce tworzenia aplikacji. Wydanie II*, Helion, 2021

[3] Jessica Young, *Global online sales reach nearly $4.29 trillion in 2020*
https://www.digitalcommerce360.com/article/global-ecommerce-sales/ (dostęp 29.10.2023)

[4] Marijn Haverbeke, *Zrozumieć JavaScript. Wprowadzenie do programowania.*, Helion, 2011

[5] Express.js
https://expressjs.com/ (dostęp 20.10.2023)

[6] HTML
https://developer.mozilla.org/en-US/docs/Web/HTML (dostęp 05.11.2023)

[7] HTTP
https://developer.mozilla.org/en-US/docs/Web/HTTP (dostęp 20.10.2023)

[8] JEST
https://jestjs.io/docs/getting-started (dostęp 20.10.2023)

[9] JSON
https://www.json.org/json-en.html (dostęp 20.10.2023)

[10] JWT
https://jwt.io/introduction (dostęp 20.10.2023)

[11] MongoDB
https://www.mongodb.com/docs/ (dostęp 20.10.2023)

[12] Node.js
https://nodejs.org/en/docs (dostęp 20.10.2023)

[13] React.js
https://react.dev/ (dostęp 20.10.2023)

[14] React Testing Library
https://testing-library.com/docs/react-testing-library/intro/ (dostęp 20.10.2023)

[15] REST API
https://www.ovhcloud.com/pl/learn/what-is-rest-api/ (dostęp 18.11.2023)

[16] Tailwind CSS
https://tailwindcss.com/docs/installation (dostęp 20.10.2023)

# Spis rysunków