

# **PROJEKTOWANIE APLIKACJI INTERNETOWYCH**

**SKLEP INTERNETOWY – DOKUMENTACJA**

**Zofia Lewkowicz 325297**

**Kacper Skalski 325320**

**Mateusz Bagieński 311499**

**Konrad Majewski 318804**

**Jakub Sulikowski 318839**

**Julian Uziembło 318853**

**PROWADZĄCY: dr inż. Piotr Bobiński**

**Styczeń 2025**

# Spis treści

<b>1</b>	<b>Zakres i cel projektu</b>	<b>1</b>
1.1	Praca w zespole . . . . .	2
1.2	Założenia funkcjonalności aplikacji . . . . .	3
<b>2</b>	<b>Projektowanie i implementacja bazy danych</b>	<b>3</b>
2.1	Wybrane narzędzia i technologie . . . . .	3
2.2	Koncept diagramu . . . . .	4
2.3	Definicja zbiorów encji określonych w projekcie . . . . .	5
2.4	Konfiguracja środowiska Dockerowego dla bazy danych . . . . .	7
2.5	Konfiguracja serwisu przekazywania zdjęć . . . . .	8
<b>3</b>	<b>Projektowanie i implementacja Backendu</b>	<b>8</b>
3.1	Wybrane narzędzia i technologie . . . . .	8
3.2	Implementacja REST API . . . . .	8
3.3	Endpointy . . . . .	9
3.4	Mapowanie tabel z bazy danych . . . . .	9
3.5	Napotkane problemy i rozwiązania . . . . .	11
3.5.1	Uwierzytelnianie . . . . .	11
3.5.2	Niezgodność typów w bazie danych Postgresql i Django . . . . .	11
3.5.3	Dockerfile - kopiowanie skryptów . . . . .	11
<b>4</b>	<b>Projektowanie i implementacja Frontendu</b>	<b>11</b>
4.1	Założenia . . . . .	11
4.2	Projektowanie . . . . .	12
4.3	Wybrane narzędzia i technologie . . . . .	12
4.4	Implementacja . . . . .	13
<b>5</b>	<b>Wnioski końcowe</b>	<b>24</b>

## 1 Zakres i cel projektu

Celem projektu było zaprojektowanie i wdrożenie aplikacji internetowej, która symuluje działanie sklepu online z asortymentem elementów elektronicznych. Inspiracją dla tego pomysłu była chęć połączenia teoretycznej prostoty – wynikającej z naszej codziennej styczności z tego rodzaju platformami – z wyzwaniem technicznym, jakie niesie za sobą implementacja działającego systemu e-commerce.

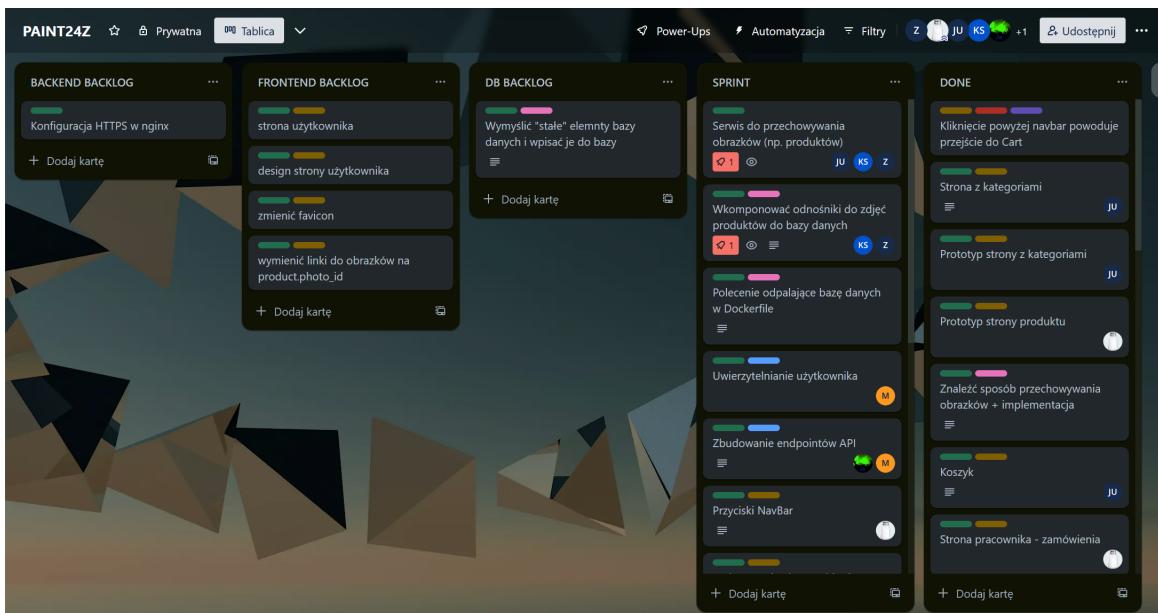
Sklep internetowy to dobry przykład aplikacji, która pozwala zgłębić różnorodne aspekty tworzenia systemów informatycznych. Projekt ten umożliwił nam eksplorację wszystkich kluczowych etapów budowy aplikacji – od projektowania bazy danych, przez tworzenie backendu, aż po implementację interfejsu użytkownika (frontend).

## 1.1 Praca w zespole

Aby zapewnić sprawność pracy zespołowej, postanowiliśmy podzielić się na grupy, dzięki czemu każdy członek zespołu miał jasno określone zadanie, na którym mógł się skoncentrować. Po wybraniu tematu projektu przeprowadziliśmy rozmowę na temat umiejętności i zainteresowań każdego z nas. Pozwoliło to na efektywne podzielenie zadań w taki sposób, aby były one interesujące i satysfakcjonujące dla wszystkich. W wyniku tego podziału przydzieliliśmy następujące funkcje:

- Baza danych: Zofia Lewkowicz, Kacper Skalski
- Frontend: Julian Uziembło, Jakub Sulikowski
- Backend: Konrad Majewski, Mateusz Bagieński

Praca w zespołach dwuosobowych była ustalana przez członków tych zespołów, a co tydzień, w niedzielę, odbywały się spotkania podsumowujące postęp całego projektu. Cotygodniowe spotkania projektowe były dla nas źródłem motywacji oraz zapewniały systematyczność w realizacji zadań. Korzystając z serwisu Trello, zapisywaliśmy kluczowe etapy rozwoju projektu, co pomagało nam monitorować postępy i lepiej planować kolejne działania.



Rysunek 1: Tablica w serwisie Trello - organizacja pracy zespołowej

Oczywiście, cały projekt wymagał ścisłej współpracy każdej z podgrup, aby osiągnąć ostateczny produkt. Oznaczało to, że każdy członek zespołu wiedział, czego potrzebuje i nad czym aktualnie pracują inni. Dzięki temu mogliśmy skupić się na swoich zadaniach, jednocześnie będąc świadomymi postępu prac w innych podgrupach i w razie potrzeby, gotowi do udzielenia wsparcia. Codzienna komunikacja odbywała się na specjalnym serwerze w usłudze Discord, natomiast aktualizacje architektury programu były przechowywane w serwisie GitLab-Elka.

The screenshot shows the GitLab interface for the 'PAINT24Z' project. At the top, there's a header with the project name 'PAINT24Z'. Below it is a navigation bar with tabs for 'Subgroups and projects', 'Shared projects', and 'Inactive'. A search bar is followed by filters for 'Created date' and sorting options. The main area displays a list of projects:

Project	Rating	Last Update
E-commerce-app	★ 0	2 hours ago
Database	★ 0	27 minutes ago
Backend	★ 0	41 minutes ago
Frontend	★ 0	54 minutes ago

Rysunek 2: Projekt w serwisie GitLab - organizacja pracy zespołowej

## 1.2 Założenia funkcjonalności aplikacji

Nasza aplikacja internetowa oferuje szeroki zakres funkcjonalności, które mają na celu zapewnienie użytkownikom jak najlepszego doświadczenia zakupowego. Klienci będą mogli przeglądać sklep oraz uzyskiwać szczegółowe informacje o produktach, takie jak cena, opis, producent, waga czy skład. Dodatkowo aplikacja umożliwia zakładanie kont użytkowników, do których przypisane będą spersonalizowane koszyki oraz historia zamówień.

System zapewni również funkcje filtrowania produktów według kategorii, co pozwoli użytkownikom szybko znaleźć interesujące ich przedmioty.

Dla pracowników sklepu przewidziano możliwość przeglądania szczegółowych informacji, takich jak lokalizacja produktu w magazynie oraz ilość dostępnych sztuk. Dzięki temu proces zarządzania zapasami stanie się bardziej efektywny. Pracownik ma możliwość zalogowania się do systemu i przeglądania produktów z poziomu dedykowanego panelu administracyjnego, zwane go dashboardem. Panel ten umożliwia zarówno edycję istniejących produktów, jak i dodawanie nowych pozycji do oferty.

## 2 Projektowanie i implementacja bazy danych

### 2.1 Wybrane narzędzia i technologie

Jako platformę bazodanową wybraliśmy PostgreSQL, który jest uznawany za jeden z najbardziej zaawansowanych systemów relacyjnych baz danych typu open source. Schemat bazy danych zaprojektowaliśmy za pomocą narzędzia Luna Modeler w wersji 8.5.1. To oprogramowanie znacznie ułatwia tworzenie diagramów oraz generuje kod SQL na podstawie opracowanego modelu, co znacząco przyspiesza pracę. Do uruchamiania i zarządzania bazą danych w środowisku backendowym wykorzystamy Docker Desktop. Dzięki niemu możemy stworzyć dedykowany kontener dla bazy danych, co zapewnia dużą elastyczność i umożliwia łatwe zarządzanie środowiskiem projektu.

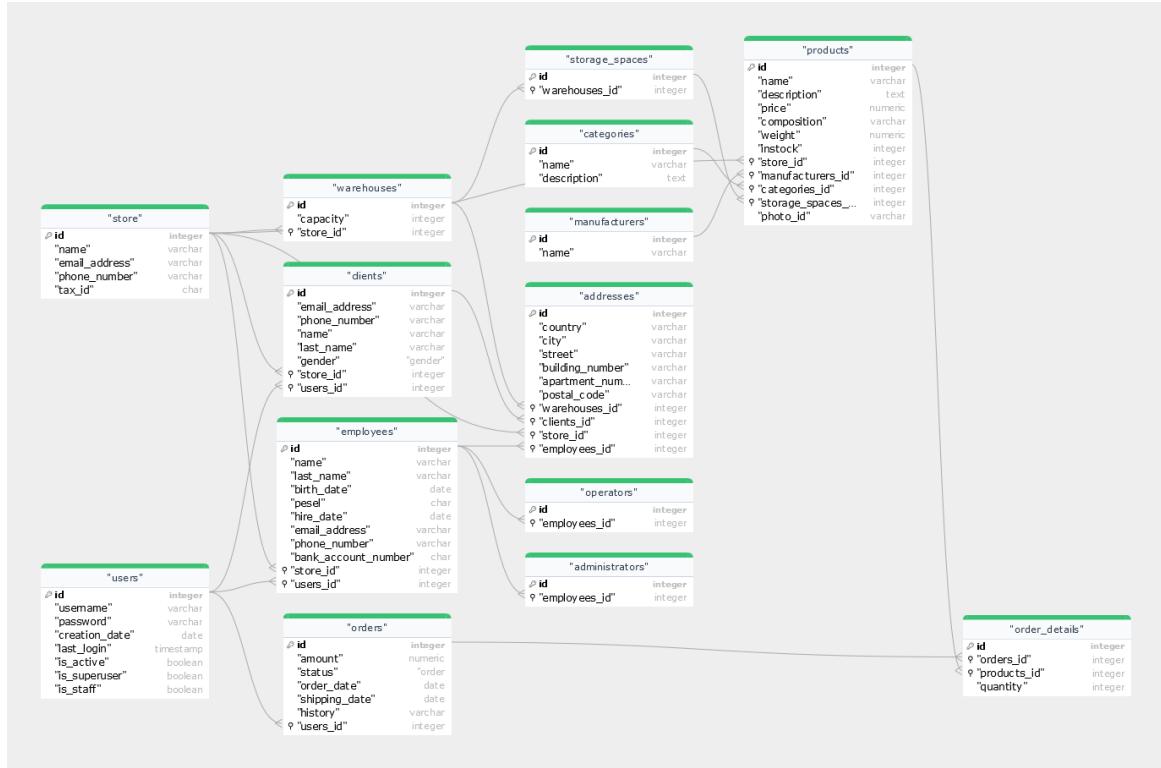
Dodatkowo, jednym z serwisów w docker-compose jest nginx - jest to oprogramowanie, które umożliwia hostowanie i udostępnianie stron internetowych na serwerach hostingodawcy.

NGINX posiada wiele ciekawych cech i poza serwerem WWW oferuje dodatkowe funkcje, takie jak serwer proxy, serwer http, czy load balancer. U nas posłużył on jako tzw. *reverse-proxy*: to on odbierał zapytania http kierowane do aplikacji i rozdzielał je na poszczególne "kanały": frontend, backend, image\_service. Dzięki temu, prosto było złączenie naszych serwisów w jedną całość. Dodatkowo, w przyszłości prosto może być np. przejście na HTTPS.

## 2.2 Koncept diagramu

Istotnym elementem naszej aplikacji jest baza danych, która pełni kluczową rolę w zarządzaniu dynamicznie zmieniającymi się informacjami. Dzięki niej możliwe jest sprawne aktualizowanie oferty produktowej, obsługa zamówień oraz przechowywanie danych o klientach, magazynach i pracownikach.

Prace nad bazą danych rozpoczęliśmy od zaplanowania wymaganych encji, ich atrybutów oraz relacji między nimi. W procesie modelowania pierwszej koncepcji schematu bazy danych wykorzystaliśmy naszą wiedzę oraz umiejętności zdobyte podczas przedmiotu Bazy Danych i Big Data. Po zaprezentowaniu początkowego projektu zespołowi i wdrożeniu poprawek, przystąpiliśmy do pracy w programie Luna Modeler. Aplikacja ta okazała się bardzo pomocna – oferowała wygodny interfejs i umożliwiała generowanie kodu SQL na podstawie utworzonego schematu graficznego. Niestety, program Luna Modeler jest dostępny bez ograniczeń jedynie w wersji premium, więc po upłynięciu 14-dniowego okresu próbnego, resztę poprawek musieliśmy dokonywać na surowym kodzie.



Rysunek 3: Diagram ER

### 2.3 Definicja zbiorów encji określonych w projekcie

- **Store:** Encja **store** definiuje podstawowe informacje o sklepie, takie jak jego nazwa, dane kontaktowe oraz numer identyfikacji podatkowej. Jest to centralny punkt odniesienia dla pozostałych encji, takich jak klienci, pracownicy czy magazyny.
- **Clients:** Encja **clients** opisuje klientów sklepu. Przechowuje ich dane kontaktowe, takie jak adres e-mail i numer telefonu, oraz dodatkowe informacje, takie jak płeć czy przypisanie do konkretnego sklepu.
- **Addresses:** Encja **addresses** odpowiada za zarządzanie adresami powiązanymi z różnymi podmiotami, takimi jak klienci, magazyny czy pracownicy. Dzięki niej można przechowywać szczegółowe dane adresowe, obejmujące kraj, miasto, ulicę, numer budynku oraz kod pocztowy.
- **Employees:** Encja **employees** opisuje pracowników sklepu. Zawiera informacje personalne, takie jak imię, nazwisko, data urodzenia, PESEL, oraz dane kontaktowe. Dzięki powiązaniu z encją **store** pracownicy są przypisani do konkretnych lokalizacji.
- **Administrators i Operators:** Te encje są podzbiorom encji **employees**. Administratorzy posiadają dodatkowe uprawnienia, takie jak zarządzanie użytkownikami i ofertą sklepu, natomiast operatorzy są odpowiedzialni za bardziej techniczne aspekty działania sklepu.

- **Warehouses i Storage Spaces:** Encja `warehouses` definiuje magazyny sklepu, a `storage_spaces` opisuje szczegółowe miejsca składowania produktów w magazynach. Relacja pomiędzy nimi umożliwia precyzyjne zarządzanie przestrzenią magazynową.
- **Products:** Encja `products` reprezentuje ofertę sklepu. Zawiera szczegółowe informacje o produktach, takie jak nazwa, opis, cena, waga, ilość na stanie oraz kategoria. Produkty są powiązane z magazynami, producentami oraz kategoriami.
- **Categories:** Encja `categories` pozwala grupować produkty według logicznych kategorii. Dzięki temu klienci mogą łatwo przeszukiwać ofertę sklepu.
- **Manufacturers:** Encja `manufacturers` opisuje producentów dostarczających towary do sklepu. Zawiera jedynie podstawowe informacje, takie jak nazwa producenta.
- **Orders i Order Details:** Encja `orders` przechowuje informacje o zamówieniach składanych przez użytkowników, w tym kwotę zamówienia, status, datę realizacji oraz historię zmian. Szczegóły zamówienia, takie jak produkty i ich ilości, są zapisane w encji `order_details`.
- **Users:** Encja `users` opisuje użytkowników systemu, zarówno klientów, jak i pracowników. Przechowuje dane logowania, datę utworzenia konta oraz ostatniego logowania, a także status aktywności i uprawnienia (np. `superuser`, `staff`).

## Najważniejsze relacje między encjami

- **Store - Employees:** Każdy sklep (`store`) może mieć wielu pracowników (`employees`). Relacja jeden-do-wielu, realizowana przez pole `store_id` w tabeli `employees`.
- **Store - Clients:** Każdy sklep może obsługiwać wielu klientów (`clients`). Relacja jeden-do-wielu, realizowana przez pole `store_id` w tabeli `clients`.
- **Store - Warehouses:** Sklep (`store`) jest powiązany z magazynami (`warehouses`), co pozwala na zarządzanie zapasami. Relacja jeden-do-wielu.
- **Users - Clients/Employees:** Użytkownik systemu (`users`) może być przypisany do jednego klienta (`clients`) lub jednego pracownika (`employees`).
- **Warehouses - Products:** Produkty (`products`) są przechowywane w przestrzeniach magazynowych (`storage_spaces`), które należą do magazynów (`warehouses`). Relacja wiele-do-jeden między `products` a `storage_spaces`.
- **Orders - Users:** Każde zamówienie (`orders`) jest powiązane z użytkownikiem (`users`), który je złożył. Relacja jeden-do-wielu, realizowana przez pole `users_id`.
- **Products - Categories/Manufacturers:** Produkty (`products`) należą do określonej kategorii (`categories`) i są przypisane do producenta (`manufacturers`). Relacje wiele-do-jeden.

- **Orders - Order Details:** Zamówienia (`orders`) są szczegółowo opisane w encji `order_details`, co pozwala na śledzenie ilości i produktów. Relacja jeden-do-wielu.

## 2.4 Konfiguracja środowiska Dockerowego dla bazy danych

W celu ułatwienia współpracy z backendem aplikacji i zapewnienia łatwego zarządzania bazą danych, zdecydowaliśmy się na wykorzystanie narzędzi Docker oraz Docker Compose. Dzięki nim nasz zespół był w stanie stworzyć kontenery, które przechowują bazę danych i umożliwiają prostą komunikację pomiędzy różnymi usługami w aplikacji.

### Dockerfile i Konfiguracja Kontenera z Bazą Danych

Pierwszym krokiem w procesie była konfiguracja Dockerfile, który pozwala na stworzenie obrazu kontenera z bazą danych PostgreSQL. Wybór obrazu bazowego `postgres:16-alpine` pozwolił nam na użycie zoptymalizowanej wersji PostgreSQL, dostosowanej do środowiska produkcyjnego. W pliku Dockerfile ustawiliśmy niezbędne zmienne środowiskowe, takie jak:

- `POSTGRES_PASSWORD` – hasło do bazy danych,
- `POSTGRES_USER` – użytkownik bazy danych,
- `POSTGRES_DB` – nazwa bazy danych.

Dodatkowo, za pomocą komendy `COPY`, przenieśliśmy nasz skrypt SQL do odpowiedniego katalogu w kontenerze, co pozwoliło na automatyczne wykonanie tego skryptu przy pierwszym uruchomieniu bazy danych w kontenerze. Dzięki temu, proces tworzenia i inicjowania bazy danych stał się zautomatyzowany.

### Konfiguracja Docker Compose

Aby uruchomić naszą aplikację wraz z bazą danych, użyliśmy narzędzia `docker-compose`, które pozwoliło na konfigurację i zarządzanie wieloma usługami w jednym pliku YAML. W pliku `docker-compose.yml` zdefiniowaliśmy wszystkie niezbędne usługi, w tym kontener bazy danych, backend w Django, frontend w React oraz serwer Nginx.

Kontener bazy danych został skonfigurowany tak, aby przechowywał dane na zewnętrznym dysku (`volume`), co zapewniało trwałość danych, nawet po ponownym uruchomieniu kontenera. Dzięki ustawieniu opcji `depends_on`, kontener bazy danych uruchamiał się po innych usługach, zapewniając, że aplikacja miała dostęp do bazy danych już przy jej starcie.

Zdefiniowaliśmy również sieć `dev`, w ramach której wszystkie usługi mogły łatwo się komunikować.

## 2.5 Konfiguracja serwisu przekazywania zdjęć

Aby zrealizować optymalny system przekazywania zdjęć produktów obecnych w naszej bazie danych, uznaliśmy, że wykorzystamy do tego Pythonowy framework **FastAPI** pozwalający na tworzenie aplikacji opartych na mikroserwisach. Stworzone przez nas API posiada dwa główne endpointy:

### Endpoint do przesyłania zdjęć (POST /images/):

Dzięki niemu producent, który chciałby dodać zdjęcie swoich produktów, może je przesłać w jednym żądaniu na podany adres. Każde zdjęcie jest zapisywane w systemie plików na serwerze z unikalnym identyfikatorem (UUID).

### Endpoint do pobierania zdjęć (GET /images/image\_id):

Podając odpowiedni identyfikator naszego obrazu, możemy pobierać i wyświetlać jego treść. Pozwala to na integrację z frontendem aplikacji oraz umożliwia nam lokalną edycję przechowywanych zdjęć w razie wystąpienia niepożądanych zmian.

## 3 Projektowanie i implementacja Backendu

### 3.1 Wybrane narzędzia i technologie

Zdecydowaliśmy się na skorzystanie z frameworku **Django** oraz **Django Rest API**. Do wyboru tego rozwiązania przyczyniła się dobra znajomość Pythona przez wszystkich członków zespołu. Ze względu na różne środowiska deweloperskie skorzystaliśmy z **Dockera**. Jako bazę do kontenera posłużył nam obraz z Pythonem w wersji **3.13** działającym na systemie **Alpine Linux**.

Jako narzędzie do testowania skorzystaliśmy z narzędzia **Postman**. Pozwoliło ono na proste wysyłanie zapytań do API.

### 3.2 Implementacja REST API

Proces rozpoczyna się od stworzenia modeli w Django, które reprezentują dane przechowywane w bazie danych. Następnie, za pomocą serializerów w DRF, mapuje się te modele na formaty JSON, co umożliwia łatwą wymianę danych między backendem a frontendem. Kluczową rolę odgrywają widoki i endpointy – widoki, takie jak APIView czy bardziej abstrakcyjne ViewSet (w naszym przypadku każdy endpoint został zaimplementowany z wykorzystaniem widoków API Django Rest Framework), obsługują logikę API, natomiast endpointy (tworzone za pomocą routerów) stanowią punkty dostępu do zasobów aplikacji. Dzięki temu front-end może komunikować się z backendem za pomocą żądań HTTP (np. GET, POST, PUT, DELETE), a odpowiedzi w formacie JSON są łatwo przetwarzane przez interfejs użytkownika. Django Rest Framework automatycznie wspiera również funkcjonalności takie jak paginacja, filtrowanie czy uwierzytelnianie.

### 3.3 Endpointy

W pliku *urls.py* dodaliśmy ustalone ścieżki. Zmapowaliśmy do nich funkcje/metody, które zaimplementowaliśmy w pliku *views.py*

- **/products/** - Zwraca listę wszystkich produktów z opcjonalnym filtrowaniem i sortowaniem.
- **/product/<int:id>/** - Zwraca szczegóły produktu o podanym *id*.
- **/categories/** - Zwraca listę kategorii produktów.
- **/about/** - Zwraca informacje o sklepie, w tym dane adresowe.
- **/manufacturers/** - Zwraca listę producentów.
- **/store/** - Zwraca informacje o dostępnych sklepach.
- **/orders/** - Zwraca listę wszystkich zamówień (dostępne tylko dla administratorów).
- **/client/<int:user\_id>/orders/** - Zwraca zamówienia dla określonego użytkownika (dostęp wymaga uwierzytelnienia i odpowiednich uprawnień).
- **/makeorder/** - Umożliwia złożenie zamówienia (dostęp wymaga uwierzytelnienia).
- **/users/** - Zwraca listę wszystkich użytkowników.
- **/register/** - Umożliwia rejestrację nowego użytkownika oraz klienta.
- **/token/** - Generuje token uwierzytelniający dla użytkownika.
- **/admin/** - Panel administracyjny Django.

### 3.4 Mapowanie tabel z bazy danych

W celu łatwego korzystania z obiektów znajdujących się w bazie danych, skorzystaliśmy z wbudowanego w Django ORM. W pliku *models.py* zdefiniowaliśmy klasy, w które odpowiadały tabelom w bazie danych.

Napisane przez nas serializatory (w pliku *serializers.py*) pozwoliły na łatwe zwracanie danych z bazy danych, jak również walidowanie poprawności przychodzących danych.

Modele danych w Django służą jako reprezentacja tabel w bazie danych, pozwalając na łatwe operacje CRUD. Oto kluczowe modele:

#### Model Products

Reprezentuje produkty dostępne w sklepie. Główne pola:

- **name** - nazwa produktu (**CharField**),
- **description** - opis produktu (**TextField**),

- `price`, `weight` - cena i waga (`DecimalField`),
- `manufacturer` - powiązanie z producentem (`ForeignKey` do `Manufacturers`),
- `category` - przypisana kategoria (`ForeignKey` do `Categories`).

### Model Orders

Obsługuje zamówienia. Pola:

- `amount` - kwota zamówienia (`DecimalField`),
- `status` - status zamówienia (`CharField`),
- `order_date`, `shipping_date` - daty zamówienia i wysyłki (`DateField`).

### Model OrderDetails

Przechowuje szczegóły zamówień, takie jak:

- `order` - powiązane zamówienie (`ForeignKey` do `Orders`),
- `product` - powiązany produkt (`ForeignKey` do `Products`),
- `quantity` - ilość zamówionego produktu (`IntegerField`).

### Model Clients

Reprezentuje klientów, w tym dane osobowe i kontaktowe:

- `name`, `last_name` - imię i nazwisko,
- `email_address`, `phone_number` - dane kontaktowe,
- `gender` - płeć (wybór z `TextChoices`),
- `store` - przypisany sklep.

### Relacje między modelami

Dzięki mechanizmom Django zaimplementowano relacje:

- `OneToOneField` - np. między `Employees` a `Administrators`,
- `ForeignKey` - np. między `Products` a `Categories`,
- `TextChoices` - obsługa enumeracji dla pól takich jak status zamówienia.

## Serializacja danych

Dla API zaimplementowano serializatory z wykorzystaniem `rest_framework`, które umożliwiają konwersję obiektów modelowych do formatu JSON. Przykłady:

- `ProductsSerializer` - serializuje dane o produktach,
- `OrdersSerializer` - obsługuje zamówienia,
- `CustomTokenObtainPairSerializer` - obsługa JWT do uwierzytelniania.

## 3.5 Napotkane problemy i rozwiązania

### 3.5.1 Uwierzytelnianie

W przypadku uwierzytelniania problemem okazało się wykorzystanie biblioteki *simplejwt*. Z tego powodu pierwotnie utworzony model *Users* nie był kompatybilny. Aby rozwiązać ten problem, trzeba było przedefiniować *Users* w naszej bazie danych. Na przykład dodać pola takie jak `is_superuser` lub zmienić wymagania dotyczące długości hasła. Następnie trzeba było odpowiednio poprawić mapowanie i inne funkcje uwzględniające te pola.

### 3.5.2 Niezgodność typów w bazie danych Postgresql i Django

Jednym z większych problemów, które napotkaliśmy, była niezgodność niektórych typów danych, które osoby odpowiedzialne za bazę danych wybrały z ORMMem Django. Jednym z nich jest *money*. Użycie go skutkowało błędem konwersji danych. Dużo czasu zajęło nam zrozumienie, co było problemem. Rozwiązaniem okazała się zmiana typu danych w bazie danych z *money* na *numeric* z odpowiednią liczbą miejsc po przecinku.

### 3.5.3 Dockerfile - kopiowanie skryptów

Korzystaliśmy z systemów operacyjnych Arch Linux oraz Windows. Windows korzystał z innych znaków końca linii, a system plików NTFS nie trzymał Unix-owych atrybutów plików. Plik *entrypoint.sh*, który był kopowany do kontenera, miał problem z działaniem na komputerze z Windowsem. Żeby to naprawić i zapobiec temu problemowi w przyszłości, dodaliśmy automatyczną konwersję znaków końca linii na te używane przez Linux (narzędzie *dos2unix*) oraz ustalialiśmy atrybuty plików po przekopiowaniu do kontenera.

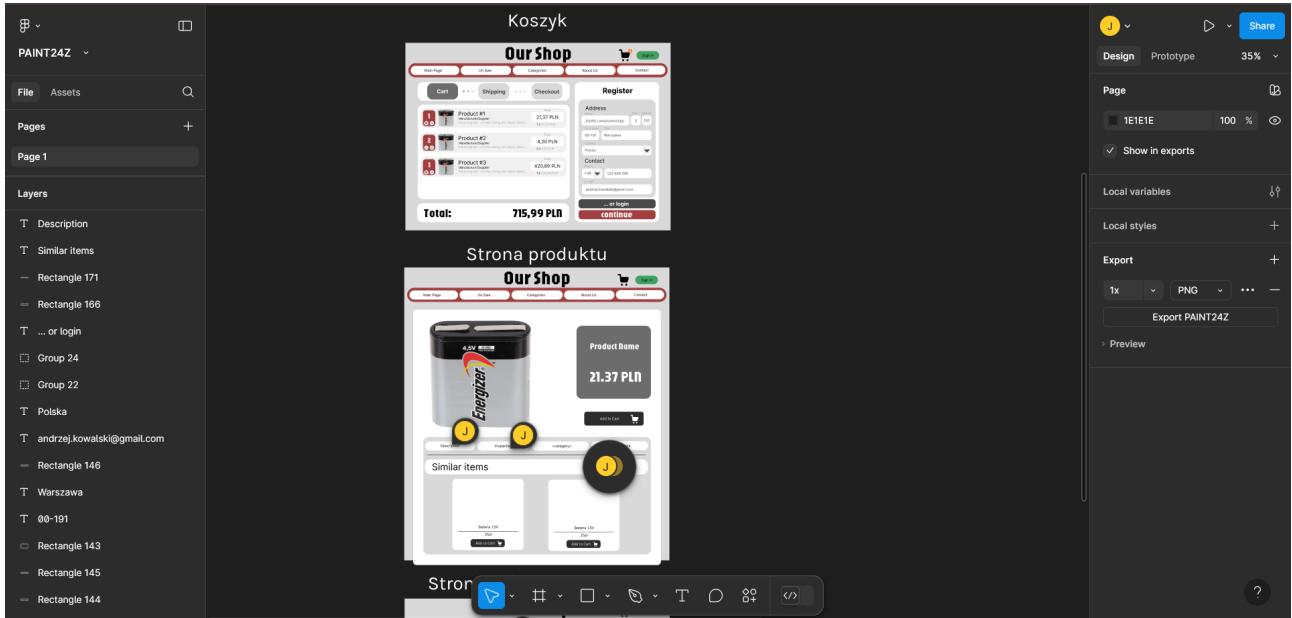
## 4 Projektowanie i implementacja Frontendu

### 4.1 Założenia

Strona miała być łatwa w obsłudze dla klienta jak i pracownika sklepu. Powinna spełniać oczekiwania klienta, pozwalać mu na zakupy przez internet, przeglądanie oferty sklepu oraz logowanie do serwisu. Z perspektywy pracownika, powinien on mieć możliwość zarządzania asortymentem sklepu oraz zamówieniami klientów.

## 4.2 Projektowanie

Do projektowania aplikacji użyliśmy narzędzia internetowego o nazwie Figma. Daje ona możliwość prostego tworzenia kształtów, dodawania tekstu, ikon. Dzięki temu mogliśmy w łatwy sposób zaplanować, jak ma wyglądać docelowa strona, co przyczyniło się do ułatwienia przyszłej pracy.



Rysunek 4: Zrzut ekranu z narzędzia Figma z zaprojektowanymi podstronami: koszyka oraz produktu.

## 4.3 Wybrane narzędzia i technologie

### React

React to otwarto źródłowa biblioteka JavaScript stworzona przez Facebooka. Pozwala ona developerom na budowanie dużych aplikacji internetowych, które mogą zmieniać dane, nie przeładowując przy tym strony. Kluczową cechą Reacta jest oparcie na komponentach, które umożliwiają tworzenie izolowanych i ponownie wykorzystywanych bloków kodu, co przekłada się na szybszy i bardziej efektywny rozwój aplikacji. Posłużył nam on do napisania front-endu aplikacji. Wchodząc trochę głębiej w tą technologię: użyliśmy komponentów funkcyjnych (definiowanych jako funkcje, w przeciwieństwie do klas, jako które definiowane były komponenty w poprzednich wersjach React). Do zarządzania stanem użyliśmy hooks API, które w prosty sposób pozwala zarządzać reaktywnymi wartościami, reagować na zmiany stanu i renderować od nowa tylko tą część UI, która tego wymaga.

## **Next.js**

Next.js to także otwartoźródłowy projekt na bazie Node.js. W naszym projekcie służy on za serwer, który serwuje użytkownikom strony HTML utworzone z komponentów React.

## **TypeScript**

TypeScript to rozszerzenie języka Javascript, zawierające typy - każda dana musi określać swój typ. Jest on bardzo przydatny do pisania niezawodnych aplikacji, ponieważ pozwala wyłapać dużo błędów powodowanych niedokładnym pisaniem kodu, które wyszły by na światło dzienne dopiero na etapie produkcji w Javascript. Dzięki typom dodatkowo użytkownik ma dostęp do funkcji intellisense, która podpowiada pisany kod w zależności od typu danych lub typu obiektu, co jeszcze bardziej usprawnia pracę.

## **Tailwind**

Tailwind to prosta biblioteka CSS, eliminująca potrzebę pisania plików .css. Działa ona w ten sposób, że zawiera wiele predefiniowanych, często przydatnych opcji stylizowania komponentów, np. ich szerokość, parametr układu flex bądź kolor (tła, tekstu itd.). Posługiwać się nimi można, definiując parametr `class` (lub `className` w React) komponentu HTML, dodając do niego odzielone spacjami nazwy klas stylizujących pojedyncze jego atrybuty, np.: `<div class="flex flex-row w-3 p-5 m-8 bg-black"> ... </div>` definiuje, że "dzieci" komponentu div będą układane w rzędzie, a sam komponent będzie miał wielkość 3 (w tym przypadku: jednostki rem), padding 5 rem oraz margines 8 rem, a także kolor tła czarny.

## **Docker**

Docker to wszechstronne narzędzie do konteneryzacji aplikacji. I w naszym przypadku się ono przydało - pozwala na działanie aplikacji w taki sam sposób, niezależnie od środowiska. Pozwala także w prostszy sposób udostępnić aplikację, umieszczając ją na hostingu na zewnętrznym numerze IP. Pozwala także spiąć całą aplikację w całość, wrzucając wszystkie kontenery - frontend, backend, bazę danych - do tej samej podsieci.

### **4.4 Implementacja**

Aplikacja składa się z wielu stron. Niektóre strony składają się z wielu podstron w ramach tego samego adresu URL - jedna z wielu możliwości oferowanych przez React.

## **Strona produktu**

Na stronie produktu, oprócz paska nawigacji, możemy zauważyc panel, na którym znajdują się zdjęcie, nazwa oraz cena produktu. Dodatkowo widoczny jest przycisk dodający produkt do koszyka. Poniżej znajduje się panel z opisem, który posiada następujące sekcje:

- Description
- Properties
- Category
- Similar Items

### **Sekcja "Description"**

Zawiera opis produktu.

### **Sekcja "Properties"**

Zawiera właściwości produktu takie jak waga, czy producent

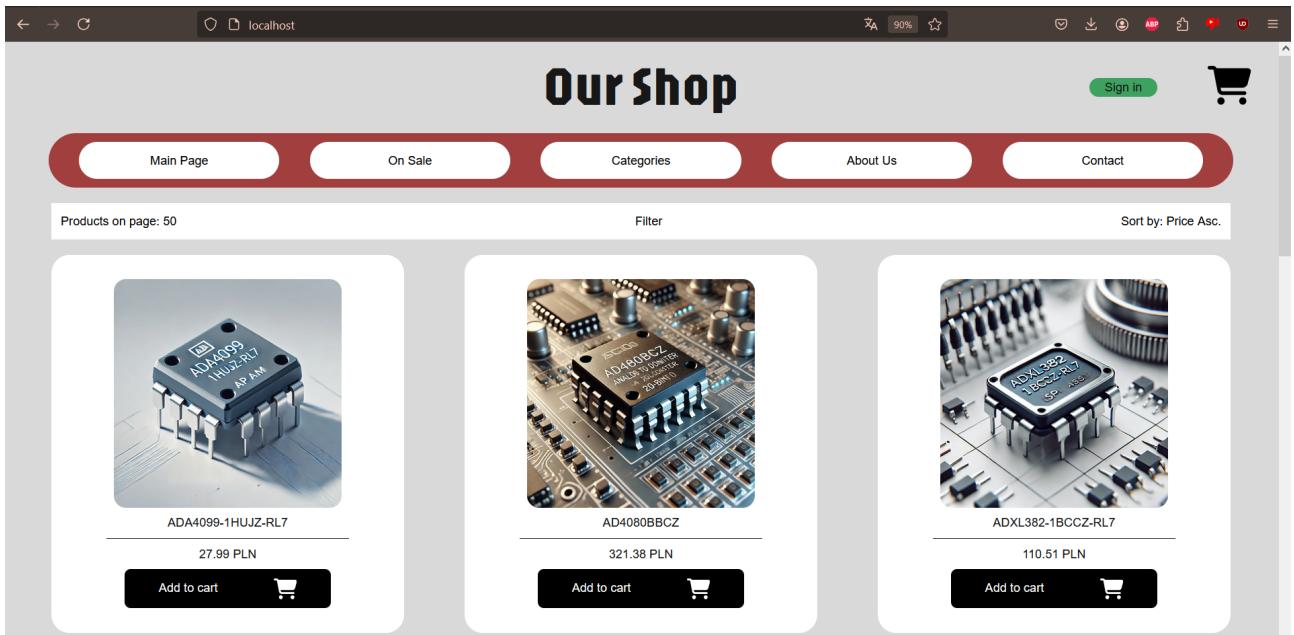
### **Sekcja "Category"**

Zawiera opis kategorii

### **Sekcja "Similar Items"**

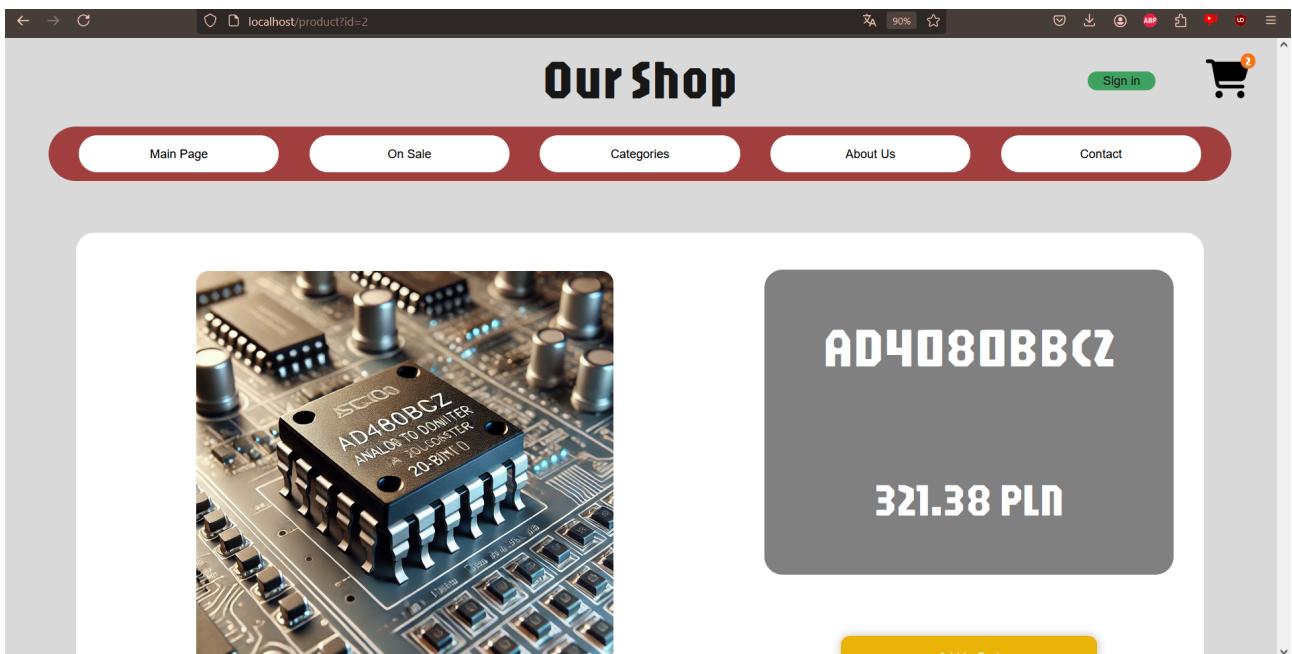
Zawiera inne produkty o tej samej kategorii

Po wejściu na stronę (adres "/"), użytkownik witany jest przez stronę z produktami. Może on od razu przeglądać je, czytać ich opisy oraz dodawać do koszyka.



Rysunek 5: Strona produktów - ekran tytułowy aplikacji.

Następnie, może wejść na stronę produktu (adres "/product?id=<id>"). Tam może zobaczyć wszystkie najważniejsze informacje o produkcie: cenę, opis, wytwórcę itd., oraz dodać produkt do koszyka.



Rysunek 6: Strona produktu - góra.

A screenshot of a product page showing the same blue integrated circuit and its price of '321.38 PLN'. Below the image and price are four tabs: 'Description', 'Properties', 'Category', and 'Similar Items'. The 'Description' tab is currently active, displaying the text 'Analog to Digital Converters - ADC 20-Bit, 40 MSPS SAR ADC'.

Rysunek 7: Strona produktu - dół.

Description

Properties

Category

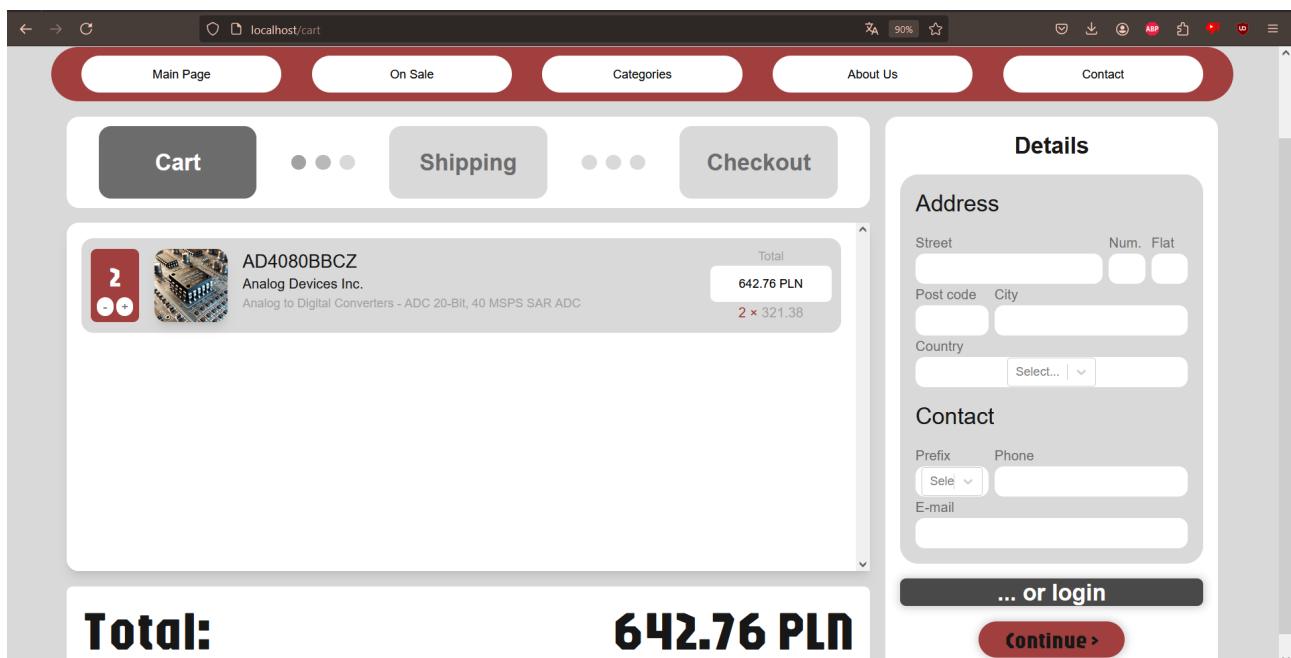
Similar Items

## Properties

- Manufacturer: Analog Devices Inc.
- Weight: 0.02 kg
- Composition: Advanced analog components

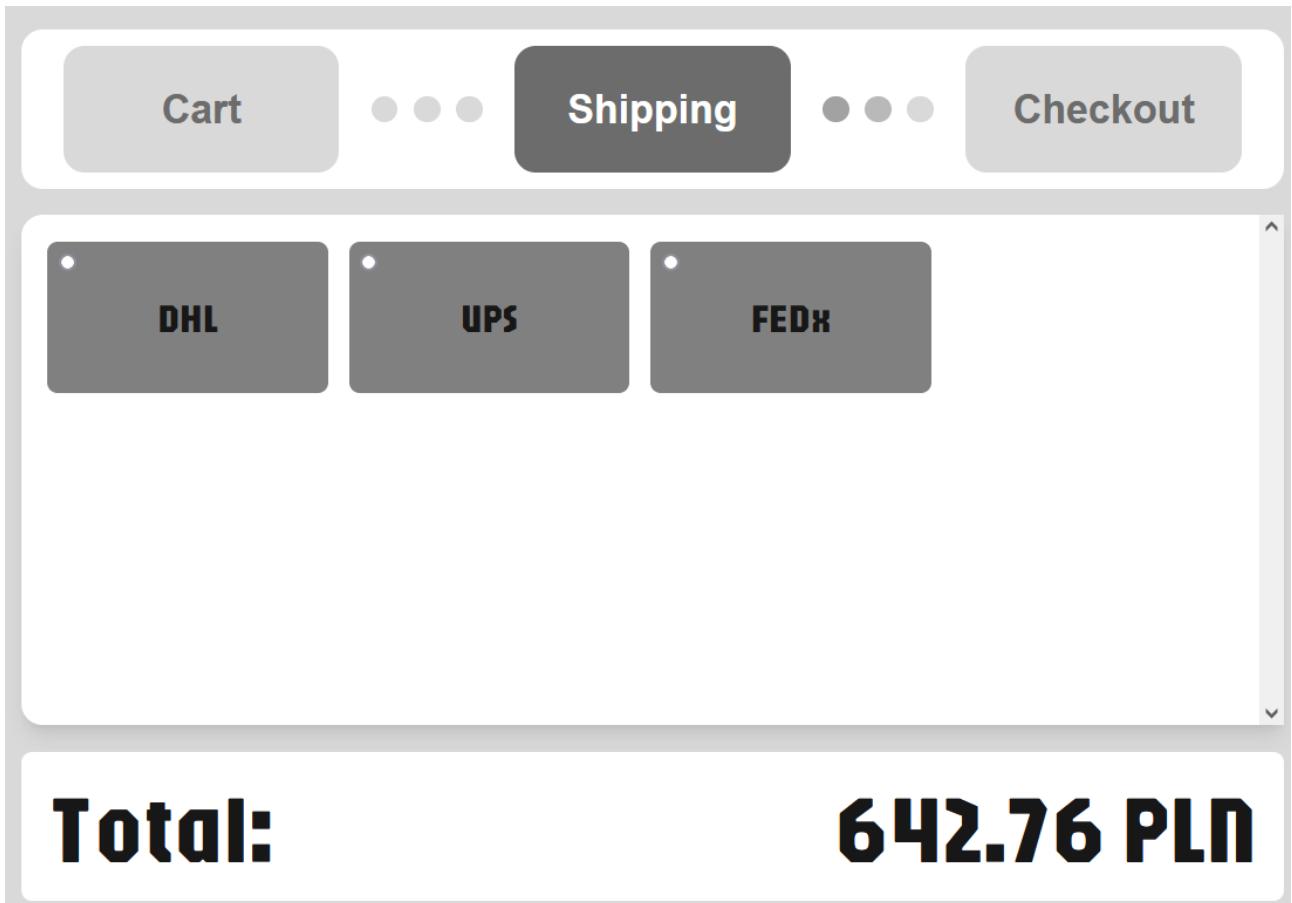
Rysunek 8: Strona produktu - dół (wartian z opisem atrybutów produktu).

Użytkownik może chcieć obejrzeć swój koszyk - wystarczy kliknąć na ikonę w prawym górnym rogu. Strona koszyka zawiera wszystkie przydatne informacje, ilość danego produktu w koszyku, oraz podliczoną wartość zamówienia. Zawartość koszyka zapisywana jest w tzw. `localStorage` - zmiennej przechowującej dane w formacie klucz-wartość. Po podaniu swoich danych (bądź uprzednim zalogowaniem), użytkownik może przejść dalej - klikając "continue".



Rysunek 9: Koszyk - rzeczy w koszyku.

Po przejściu dalej, jego oczom ukazuje się opcja wybrania dostawcy produktu ze sprededfiniowanej listy.



Rysunek 10: Koszyk - wybór dostawy.

Po wybraniu "continue", użytkownik kierowany jest na ostatnią z podstron - "Checkout". Może tam zobaczyć swoje dane, wybrane produkty, metodę dostawy itd. Po wybraniu "Finish", zostaje w systemie złożone zamówienie.

The screenshot shows a user interface for a shopping cart summary. At the top, there are three tabs: "Cart", "Shipping", and "Checkout". The "Checkout" tab is highlighted. Below the tabs, there is a table with three columns: "Product", "Amount", and "Price". One item is listed: "AD4080BBCZ" with an amount of "2" and a price of "642.76". A section titled "Chosen shipping method:" shows "UPS". At the bottom left, it says "Total: 642.76 PLN". On the right, there is a "Details" section with two tabs: "Address" and "Contact". The "Address" tab is selected, showing fields for Street, Post code, City, and Country, all containing placeholder text "Abacka". The "Contact" tab shows fields for Prefix, Phone, and E-mail, also with placeholder text "Abacka". At the bottom right are "Previous" and "Finish" buttons.

Product	Amount	Price
AD4080BBCZ	2	642.76

**Chosen shipping method:**

**UPS**

**Total:** **642.76 PLN**

**Details**

**Address**

Street: Abacka Num. Flat: 1 2

Post code: Abacka City: 1

Country: Abacka

**Contact**

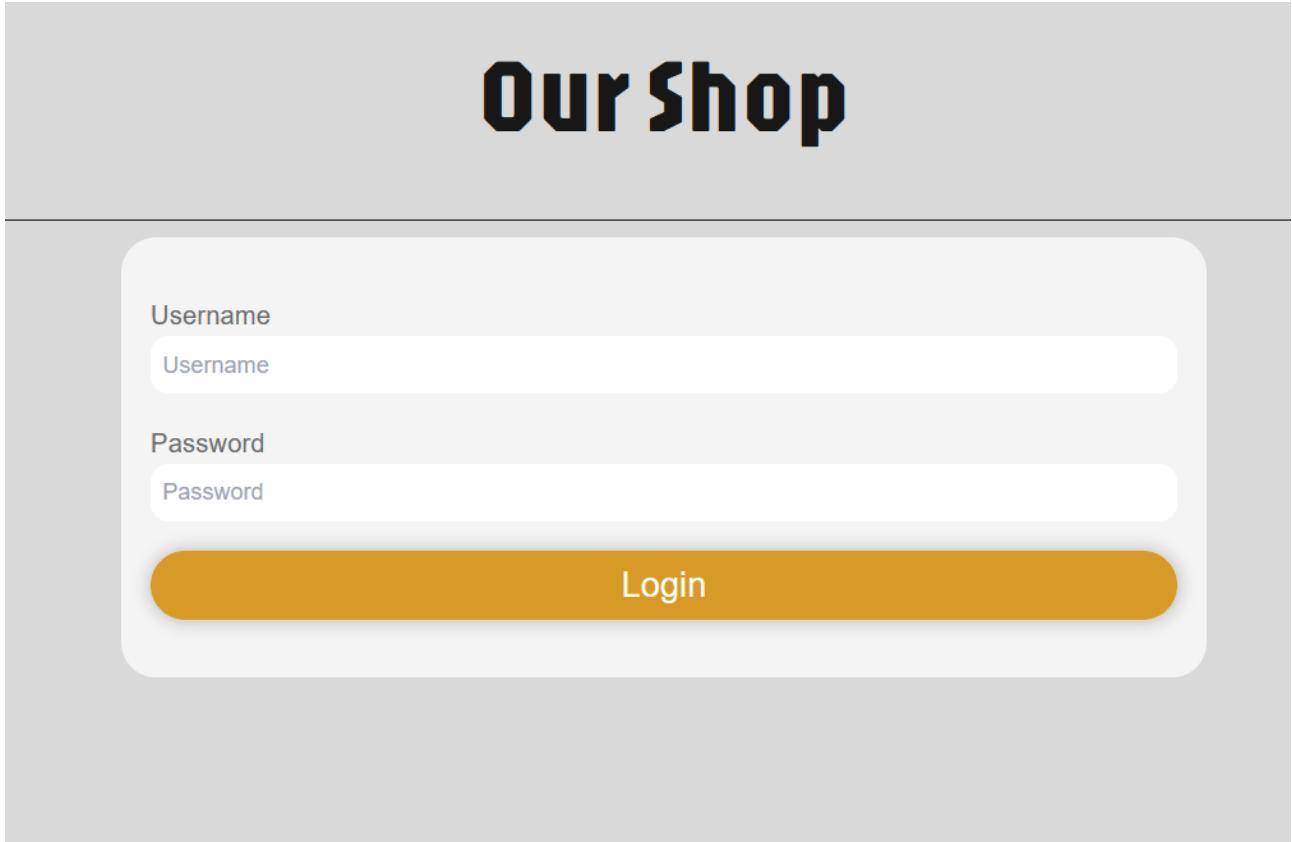
Prefix: Abacka Phone: 1

E-mail: Abacka

**< Previous** **Finish**

Rysunek 11: Koszyk - podsumowanie.

Użytkownik może zalogować się do serwisu, klikając ikonę "Sign in" w prawym górnym rogu. Jego oczom ukaże się panel logowania:



Rysunek 12: Panel logowania

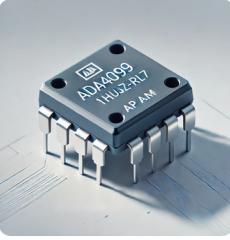
Po podaniu poprawnych danych, użytkownik zostaje zalogowany, a ikona "Sign in" zmienia się w żółtą ikonę z jego nazwą użytkownika oraz krótkim przywitaniem.

# Our Shop

Welcome, elektronicznypan 

Main Page On Sale Categories About Us Contact

Products on page: 50 Filter Sort by: Price Asc.



ADA4099-1HUIZ-RL7

27.99 PLN

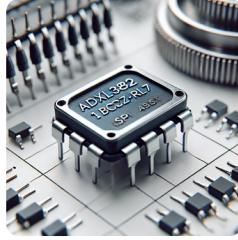
Add to cart 



AD4080BBCZ

321.38 PLN

Add to cart 



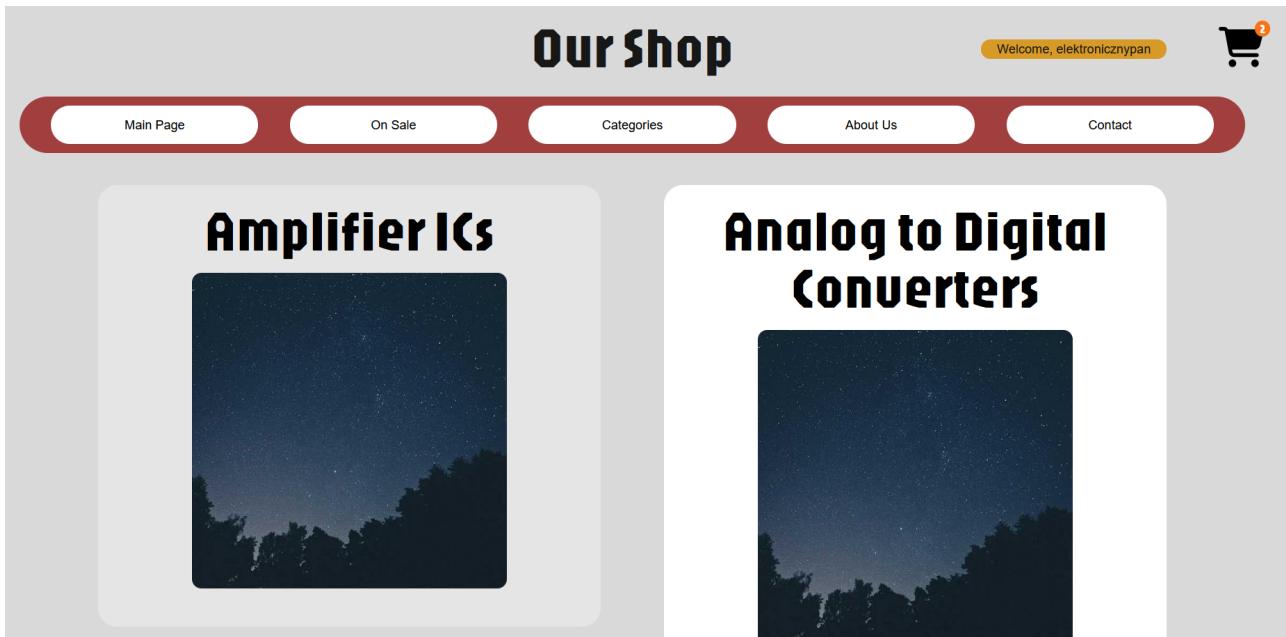
ADXL382-1BCCZ-RL7

110.51 PLN

Add to cart 

Rysunek 13: Użytkownik zalogowany.

Użytkownik może zamiast produktów przeglądać kategorie produktów. Po wybraniu kategorii, zostaje przeniesiony do ekranu głównego z produktami tylko z wybranej kategorii.



Rysunek 14: Kategorie.

Pracownik może zalogować się i przeglądać produkty z innego poziomu - tzw. dashboard'u. Daje on możliwość edycji poszczególnych produktów, jak i też dodawania nowych:

# Our Shop

**Orders**

**Add new product**

**Products**

	ADA4099-1HUUZ-RL7	Operational Amplifiers - Op Amps Precision, 10MHz, 7V/us OTT RRIO Amp	27.99 PLN	in stock: 12
	AD4080BBCZ	Analog to Digital Converters - ADC 20-Bit, 40 MSPS SAR ADC	321.38 PLN	in stock: 70
	ADXL382-1BCCZ-RL7	Accelerometers ADXL382 SPI	110.51 PLN	in stock: 23
	STM32F407VG Microcontroller	ARM Cortex-M4 32-bit Microcontroller with FPU, 1 MB Flash, 168 MHz	60.50 PLN	in stock: 99
	BC547 NPN Transistor	General-purpose NPN transistor for low-power amplification and switching	1.20 PLN	in stock: 200
	ESP32-WROOM-32 Wi-Fi Module	Low-power dual-core Wi-Fi and Bluetooth module with integrated antenna	25.00 PLN	in stock: 85
	LTC3643 Backup Power Controller	High-efficiency step-up DC/DC converter with integrated buckboost	85.00 PLN	in stock: 33

Rysunek 15: Dashboard pracownika.

**Orders**

**Add new product**

**Products**

Name: ADA4099-1HUUZ-RL7  
Price: 27.99

Description: Operational Amplifiers - Op Amps Precision, 10MHz, 7V/us OTT RRIO Amp

Weight: 0.010  
Composition: Standard electronic components  
In stock: 12

**Submit**

Rysunek 16: Dashboard pracownika - edycja produktu.

# Our Shop

---

## New product

Name

Price

Description

Weight  Composition  In stock

Image:  Nie wybrano pliku.

Manufacturer

Category

**Submit**

**Abort**

Rysunek 17: Dashboard pracownika - dodawanie nowego produktu.

## 5 Wnioski końcowe

Praca nad projektem pozwoliła nam zgłębić wiedzę na temat projektowania i implementacji aplikacji internetowych. Dzięki dobrze zorganizowanej współpracy zespołowej mogliśmy skutecznie realizować szczegółowe zadania oraz sprawnie rozwiązywać pojawiające się problemy. Wykorzystanie różnorodnych narzędzi umożliwiło nam zapoznanie się z nowymi programami i ich funkcjonalnościami, co z pewnością będzie cennym doświadczeniem zarówno w dalszej edukacji, jak i na kolejnych etapach kariery zawodowej.