POLSKO-JAPOŃSKA AKADEMIA
TECHNIK KOMPUTEROWYCH

WYDZIAŁ INFORMATYKI

KATEDRA INŻYNIERII OPROGRAMOWANIA

INŻYNIERIA OPROGRAMOWANIA I BAZ DANYCH

Wojciech Kania
s10451

Elastyczny system sprzedaży dla osób bez doświadczenia informatycznego

Praca magisterska napisana pod kierunkiem:
dr inż. Mariusz Trzaska

Warszawa, luty 2018

Spis treści

Streszczenie

Niniejsza praca omawia zagadnienia związane ze sprzedażą w internecie. Autor przeanalizował obecnie istniejące na rynku systemy sprzedażowe pod kątem ich obsługi przez osobę bez doświadczenia informatycznego.

Zostały zaprezentowane podstawowe funkcje, które musi spełniać aplikacja, aby umożliwiała prowadzenie handlu w Internecie. Przedstawiono koncepcje rozwiązań, które sprawią, że system będzie intuicyjny i prosty w obsłudze oraz nie będzie wymagał umiejętności programowania.

Efektem końcowym pracy jest działający prototyp, który umożliwia przedsiębiorcy, który ma niewielką wiedzę informatyczną, prowadzenie handlu w Internecie.

Rozdział 1

Wprowadzenie

Rynek ecommerce, czyli handlu w Internecie, wciąż rozwija się w bardzo szybkim tempie. Liczba sklepów internetowych w Polsce będzie rosła o wiele szybciej niż rozwinięte już rynki amerykańskie, czy niemieckie[4]. Wraz ze wzrostem wartości rynku ecommerce rośnie zapotrzebowanie na systemy pozwalające na sprzedaż produktów w Internecie.

Dostępne rozwiązania sprawdzają się dobrze w wielu firmach. Jest wiele rozwiązań, które z perspektywy klienta sklepu są intuicyjne. Systemy sprzedażowe oferują zaawansowane funkcje, a także są udostępnione na zasadzie Open Source, tak więc firmy mogą dostosować je do swoich potrzeb.

Jednak zarządzanie sklepami internetowymi wymaga dużej wiedzę informatycznej, a wprowadzanie zmian na sklepie wymaga często umiejętności programowania. Systemy te nie są przystosowane dla rzeczywistych odbiorców - dla przedsiębiorców.

1.1 Cel pracy

Celem pracy jest opracowanie koncepcji narzędzia, które będzie umożliwiało sprzedaż w Internecie i będzie skierowane do osób, które nie mają dużej wiedzy informatycznej.

1.2 Zarys koncepcji

System powinien posiadać następujące funkcjonalności, aby umożliwiał sprzedaż produktów w Internecie:

- dodawanie produktów,

- zarządzanie cechami produktów,
- katalogowanie produktów,
- zarządzanie nawigacją strony,
- zarządzanie wersjami językowymi strony,
- zarządzania treścią strony (CMS),

1.3 Rezultaty pracy

Rezultatem pracy jest koncepcja systemu umożliwiającego sprzedaż w internecie, którego obsługa nie będzie wymagała wiedzy informatycznej. Koncepcja zostanie przedstawiona w wersji demonstracyjnej aplikacji. W skład aplikacji demonstracyjnej będą wchodzić:

- Aplikacja Vue ze sklepem internetowym
- Aplikacja Vue z panelem administracyjnym
- Aplikacja RESTful API do sklepu internetowego stworzona w Node.js
- Aplikacja RESTful API do panelu administracyjnego stworzona w Node.js

1.4 Organizacja pracy

Na początku omówiono rynek branży ecommerce oraz funkcje, które system powinien realizować, aby umożliwiał sprzedaż w Internecie.

W dalszej części pracy zostały omówiono systemy sprzedażowe na których oparte jest ponad połowa sklepów internetowych, które korzystają z rozwiązań Open Source. Przedstawiona została analiza implementacji funkcjonalności przedstawionych w punkcie 1.2. Analiza zawiera scenariusze przypadków użycia każdej z wymienionych funkcjonalności. Wykazano wady, które mają obecne rozwiązania.

W kolejnej części pracy przedstawiono technologie, które wykorzystano do stworzenia koncepcji, która wyeliminuje mankamenty omówionych systemów.

Następne dwa rozdziały omawiają koncepcje rozwiązania problemu oraz przedstawiają prototyp koncepcji. Zaprezentowano również przykład użycia prototypu.

W ostatniej części podsumowano całą prace. Przedstawiono wady i zalety zaproponowanego rozwiązania i omówiono możliwości rozwoju prototypu.

Rozdział 2

Analiza istniejących rozwiązań

Na rynku istnieje wiele rozwiązań systemów sprzedaży w internecie. Spośród darmowych rozwiązań największą popularnością cieszą się [17]:

1. Magento - otwarte oprogramowanie napisane w języku PHP. System bazuje na frameworku Zend. Dane mogą być przechowywane w bazach: MySQL oraz MariaDB. Magento to najpopularniejsze darmowe oprogramowanie. Co drugi sklep działający w internecie i korzystający z darmowych rozwiązań jest zbudowany właśnie na tej platformie.
2. WooCommerce - wtyczka do systemu blogowego Wordpress. Woocommerce dodaje do Wordpressa możliwość dodawania produktów, podobnie jak posty na bloga. Rozszerza bloga o moduł koszyka oraz formularz zamówienia. 20% sklepów działających na darmowym oprogramowaniu zbudowana jest w oparciu o Wordpressa i WooCommerce. Są to głównie niewielkie sklepy, które decydują się na to rozwiązanie, ponieważ instalacja, a później dalsza obsługa jest bardzo prosta. Woocommerce jest jednak tylko wtyczką i rozwiązuje jedynie trywialne problemy.
3. osCommerce - jeszcze pare lat temu najpopularniejszy darmowy system sprzedaży. Obecnie posiada 5% rynku darmowych systemów sklepowych. System jest bardzo rzadko aktualizowany. Ostatnia wersja 2.3.4 została zaprezentowana 5 czerwca 2014 roku [14]. W wyniku tego osCommerce nie jest przystosowany do obecnych trendów, a jego znaczenie najprawdopodobniej będzie maleć.

Wymienione wyżej trzy systemy bardzo dobrze pokazują obecny rynek darmowych systemów sprzedaży. Na rynku istnieje jedno rozwiązanie, które jest wykorzystywane przez małe, średnie i duże sklepy - Magento. Mikro sklepy decydują się na WooCommerce. Pozostałe 30% rynku stanowią pomniejsze rozwiązania, które są albo wtyczkami do istniejących systemów CMS (np. Drupal), albo niszowymi systemami. Większość rozwiązań zostało napisane w języku PHP i wykorzystują do przechowywania danych relacyjną bazę SQL - najczęściej MySQL.

W dalszej części pracy zostaną przeanalizowane Magento w wersji 1.9.2.1 oraz osCommerce w wersji 2.3.4. Te dwa rozwiązania są dedykowanymi systemami sprzedażowymi, a nie dodatkiem do systemu blogowego, jak WooCommerce. Cieszą się one największą popularnością. Ponad połowa sklepów internetowych, która korzysta z darmowych rozwiązań, zbudowana jest właśnie na tych dwóch systemach.

2.1 Najważniejsze funkcjonalności

Podczas analizy skupię się na pięciu czynnościach, które są kluczowe dla każdego systemu sprzedażowego:

2.1.1 Dodawanie produktów

Podstawową funkcjonalnością każdego systemu sprzedaży jest dodawanie produktów. Atrybutami produktu, bez których nie jest możliwe prowadzenie sprzedaży są: nazwa produktu, cena produktu, ilość produktu w magazynie, stan produktu (włączony/wyłączony). W większości przypadków oprócz wspomnianych kluczowych atrybutów właściciel sklepu potrzebuje dodać opis oraz zdjęcia do produktu.

2.1.2 Zarządzanie cechami produktów

Każdy produkt można opisać stałym zestawem zmiennych: nazwa produktu, opis produktu, cena produktu, ilość w magazynie, status produktu (włączony/wyłączony). Z punktu widzenia relacyjnej bazy danych mamy prostą tabelę: name VARCHAR(80), description TEXT, price DECIMAL(4,2), count INT(11), isActive (BOOLEAN). W Panelu Administracyjnym również otrzymamy prosty formularz z polami tekstowymi (<input type='text'>) na nazwę, cenę oraz ilość, polem na wieloliniowy tekst (<textarea>), oraz radio buttony odpowiadające za status produktu.

Problemy zaczynają się, kiedy chcemy bardziej opisać produkt. Załóżmy, że prowadzimy sklep internatowy z odzieżą. Mamy w ofercie spodnie. Spodnie mają różne kolory, rozmiary oraz tkaninę. Do każdych spodni proponujemy pasek za dodatkową opłatą. Cena danego modelu spodni mogą różnić się w zależności od koloru.

Efekt końcowy, który powinien zobaczyć klient można łatwo sobie wyobrazić. Mamy nazwę, opis i cenę spodni. Kolory są prezentowane w postaci radio buttonów. Po wyborze danego koloru powinna zaktualizować się jego cena. Rozmiary zostaną wyświetlone użytkownikowi

jako lista rozwijana. Użytkownik powinien móc zadecydować, czy chciałby otrzymać dodatkowo pasek klikając w checkbox.

Każdy system internetowy powinien oferować możliwość zarządzania cechami, gdyż praktycznie w każdej branży istnieją produkty, które mają swoje unikalne cechy - może być to rozmiar spodni, pamięć masowa komputera, czy pojemność perfum.

2.1.3 Katalogowanie produktów

Całą ofertę produktową warto skatalogować. Dzięki temu użytkownik szybciej znajdzie produkt, który go interesuje. Dokładne skatalogowanie produktów pozwala na zastosowanie skutecznej kampanii marketingowych skoncentrowanych na konkretnych produktach, które mogą być dopasowane do płci, grupy wiekowej, czy nawet zainteresowań klienta.

2.1.4 Zarządzanie nawigacją

Nawigacja to kluczowy element każdej strony internetowej. Dzięki dobrej i intuicyjnej nawigacji użytkownik będzie miał pozytywne doświadczenie ze sklepem. Tworzenie odpowiednich przekierowań jest ważnym elementem marketingu i kluczowe dla pozycjonowania strony. Dobrze stworzona nawigacja i odpowiednie skatalogowanie zmniejszy czas poszukiwania produktu przez użytkownika.

2.1.5 Tłumaczenie

Ponieważ Internet to sieć globalna, sklep ma łatwą drogę rozwoju i wejścia na nowe, zagraniczne rynki. Wiąże się to z koniecznością utworzenia nowej wersji językowej strony. Tłumaczenie na inne języki jest ważną funkcją i występuje w większości systemów umożliwiających sprzedaż.

2.1.6 Zarządzanie pozostałymi rodzajami treści

Katalog produktów to podstawowa rzecz w sklepie internetowym. Ale jest jeszcze wiele rzeczy które musi posiadać sklep internetowy, aby mógł funkcjonować. Każdy sklep internetowy powinien mieć regulamin (wymagany przez polskie prawo), miejsce na informacje o przesyłce, polityce prywatności, możliwościach i sposobach zwrotu produktu. Oprócz tego właściciel sklepu powinien mieć możliwość tworzenia treści promocyjnych na sklepie (może być to slider z banerami promocyjnymi, bądź lista czterech najlepiej sprzedających się produktów).

2.2 Magento

Pierwszym omówionym systemem będzie Magento. Jest to obecnie najpopularniejsze rozwiązanie spośród darmowych oprogramowań. Duża liczba sklepów opartych o to oprogramowanie jest wynikiem m.in. dużej możliwości konfiguracji oraz ilości istniejących integracji z systemami płatności, czy firmami kurierskimi.

2.2.1 Dodawanie produktów

Formularz dodawania produktu jest podzielony na kilka sekcji (Rysunek 2.1). Najważniejsze z nich to 'General', 'Prices', 'Images' oraz 'Inventory'. W tych sekcjach znajdują się pola, które wymagaja wprowadzenia danych. Pierwszą nieintuicyjną rzeczą jest to, że użytkownik zostaje poinformowany o tym, że nie uzupełnił wymaganego pola dopiero po próbie zapisu produktu. Walidacja wartości odbywa się również dopiero po próbie zapisu.

W sekcji 'General' wprowadza się nazwę produktu, opis produktu, SKU produktu, wagę oraz status produktu. Wszystkie pola domyślnie są podstawowymi polami formularzu (input oraz textarea). Opis produktu w tym oknie powinien być wprowadzony jako kod HTML. Dopiero po kliknięciu przycisku 'WYSIWYG Editor' pojawia się okienko z edytorem, który jest już bardziej zrozumiały dla zwykłego użytkownika.

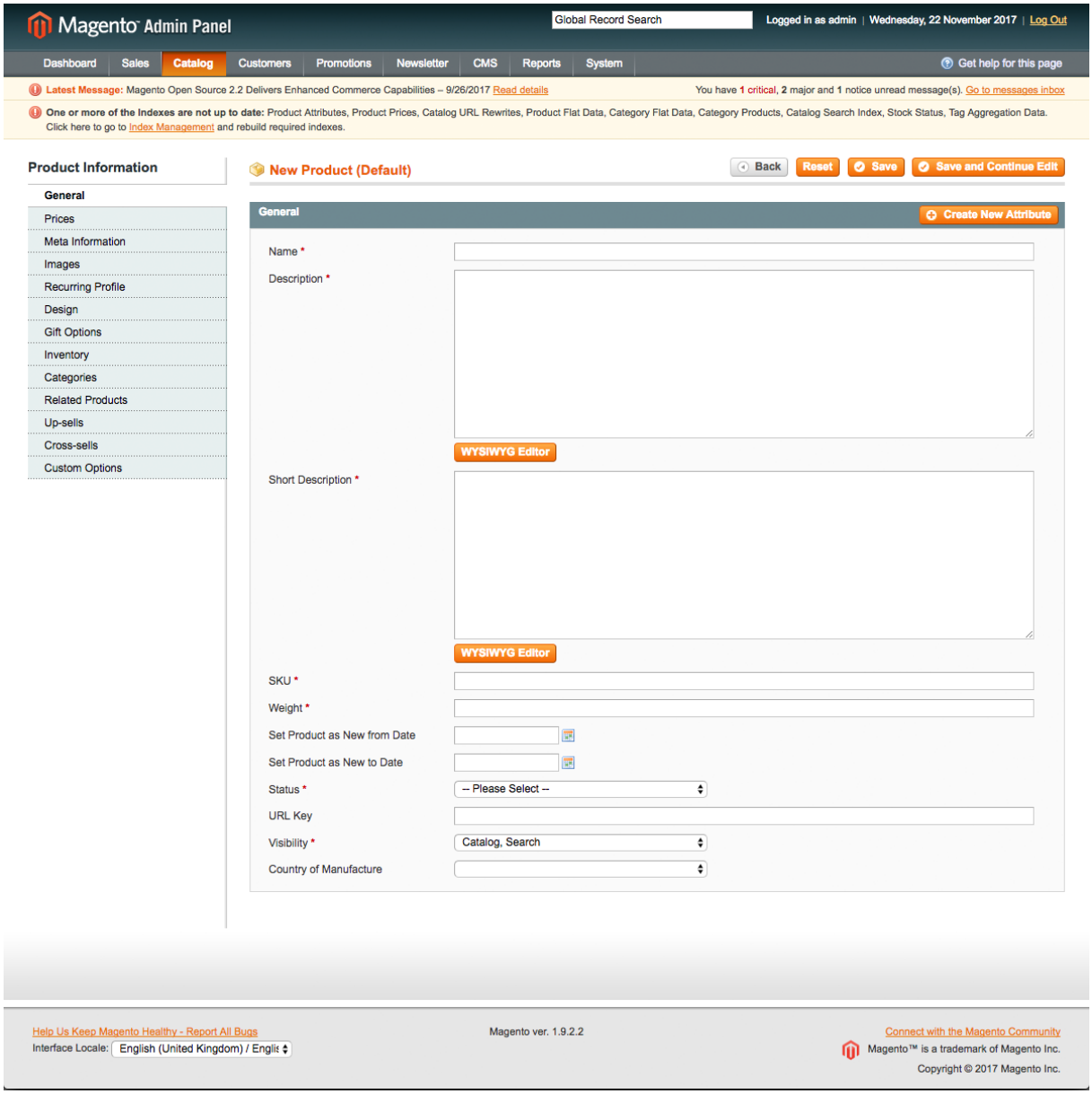
W sekcji 'Images' dodawane są zdjęcia oraz ustawiana jest ich kolejność oraz rola zdjęcia (podstawowe zdjęcie, małe zdjęcie, miniaturka). Problemem w tej sekcji jest moduł obsługujący wysyłanie zdjęć na serwer. Stworzony jest on we Flashu. Powoduje to problemy w wielu nowych przeglądarkach, gdzie Flash jest z automatycznie blokowany. Przed dodaniem zdjęć należy w preferencjach przeglądarki dodać wyjątek, aby zawsze pozwalać Flashowi na działanie na stronie sklepu.

2.2.2 Zarządzanie cechami produktów

Stworzenie produktu z cechami w systemie Magento jest możliwe, ale dla przedsiębiorcy, który nie ma doświadczenia informatycznego może być to bardzo trudne.

Aby stworzyć taki produkt należy najpierw dodać cechy, które występują w produkcie. Przykładowym produktem będą spodnie, o których wspomniano już we wcześniejszej części pracy. Cechami produktu będzie: kolor, rozmiar, tkanina, producent.

W Panelu Administracyjnym dodajemy cechy produktu: kolor (typ dla właściciela sklepu: lista rozwijana), rozmiar (typ pola dla właściciela sklepu: lista rozwijana), tkanina (typ pola dla właściciela: lista rozwijana). Rysunek 2.2 przedstawia ekran dodawania cechy produktu.

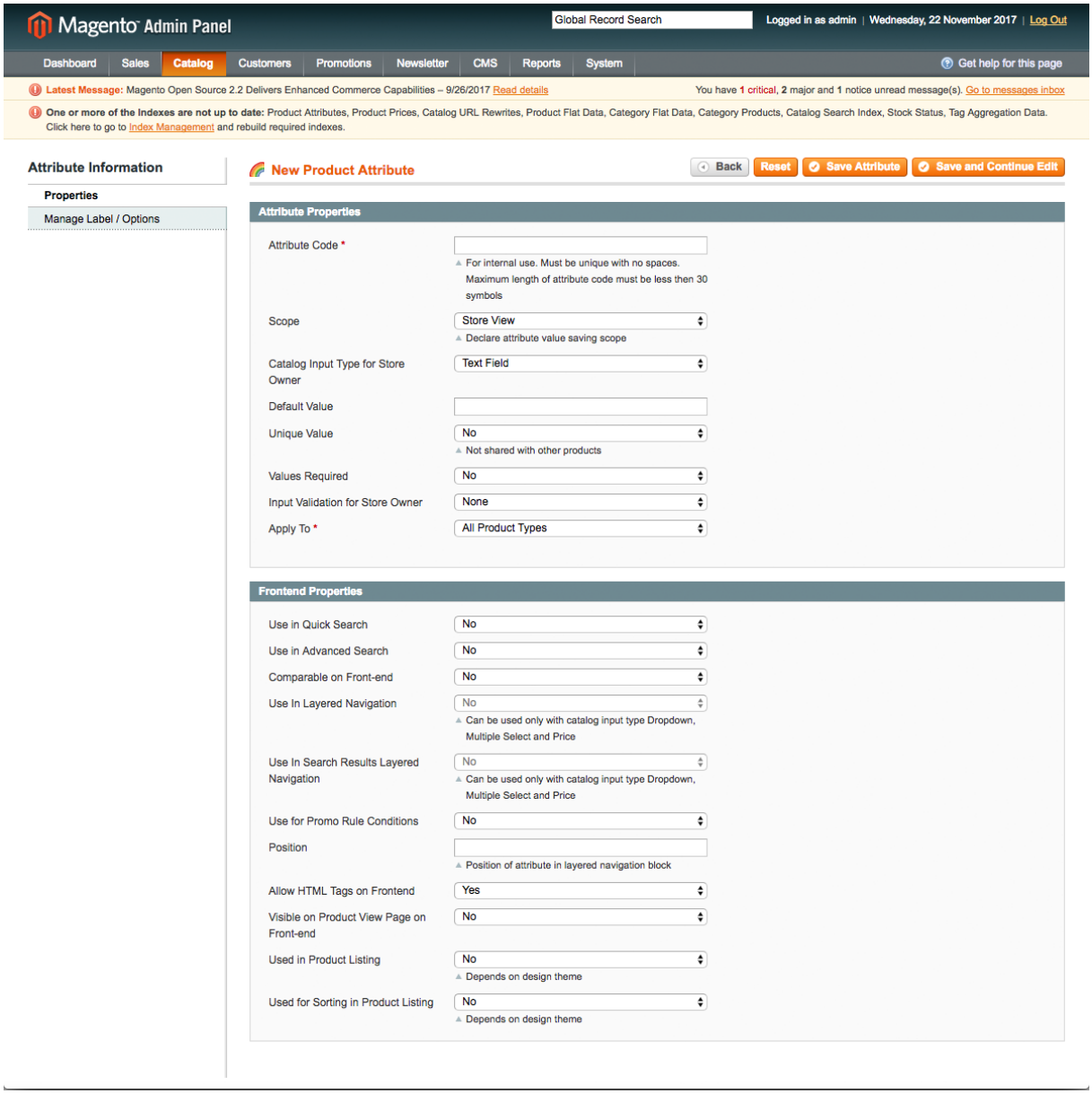


Rysunek 2.1: Magento: Ekran dodawania nowego produktu. Sekcja 'General'

Po skonfigurowaniu cech należy utworzyć zestaw cech produktu i przypisać do tego zestawu utworzone wcześniej cechy).

Kolejnym krokiem jest utworzenie produktu konfigurowalnego i wybranie zestawu cech, a także zestawu cech konfigurowalnych (w naszym przypadku cechami konfigurowalnymi będą kolor i rozmiar).

Wypełniamy standardowe pola w Magento, tak jak przy dodawaniu produktu (o tym procesie wspomina poprzedni podrozdział). Aby dodać teraz wyżej wymienione cechy wybieramy "Produkty powiązane"i dodajemy kolejne produkty powiązane, czyli tak naprawdę atrybuty, które mają wpływ na stan magazynowy.



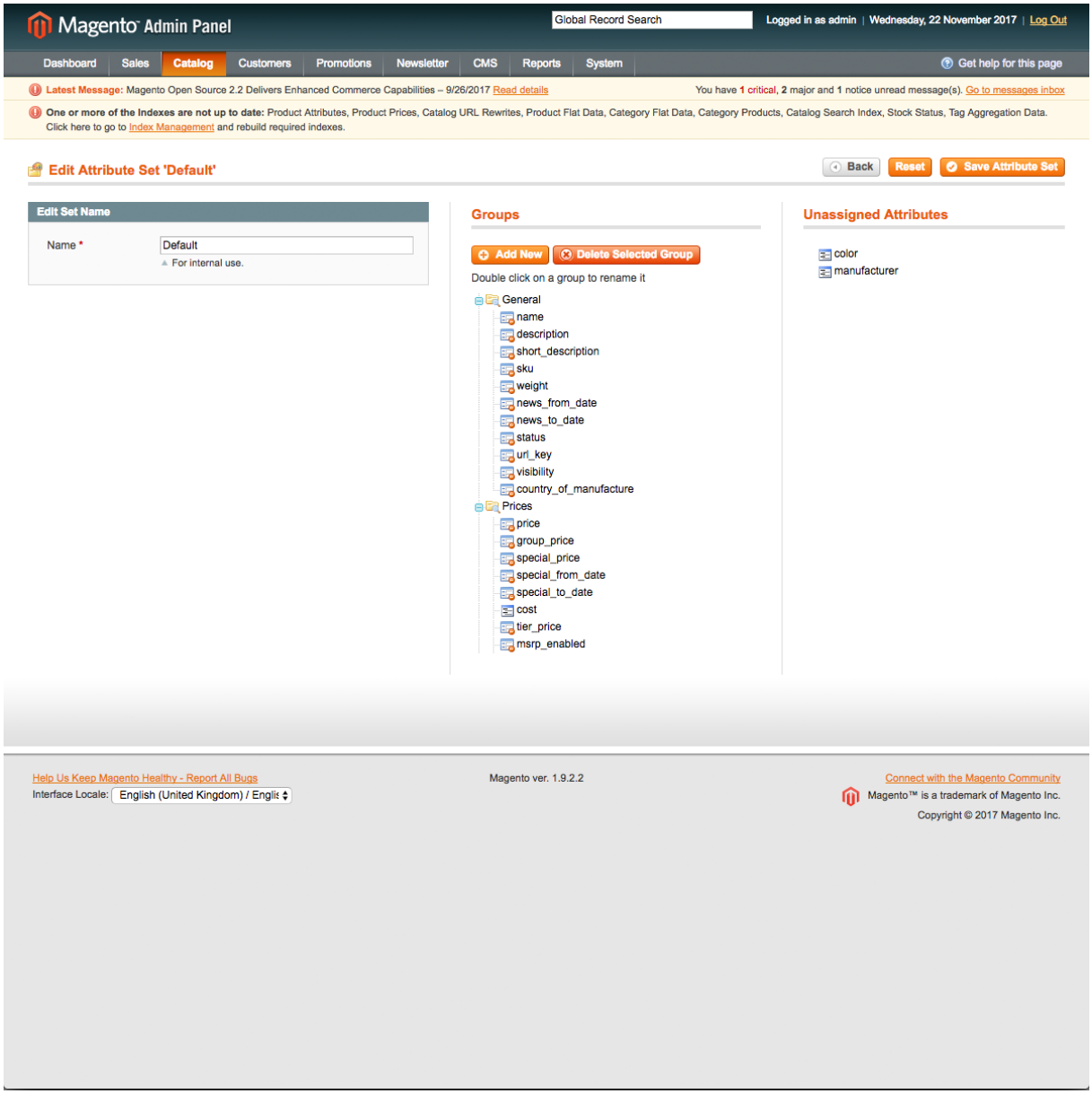
Rysunek 2.2: Magento: Ekran dodawania cechy produktu

Dla osoby, która nie ma doświadczenia informatycznego etap dodawania produktów jest z całą pewnością nieintuicyjny i skomplikowany.

2.2.3 Katalogowanie produktów

W Magento katalogujemy produkty po kategoriach. Kategorie mogą mieć swoje podkategorie. Produkt może należeć do dowolnej liczby kategorii. Istnieje też możliwość, że produkt należy do podkategorii, ale już nie należy do kategorii nadrzędnej.

W Magento każda kategoria tworzona jest jako niezależny zbiór produktów. Nie jest moż-



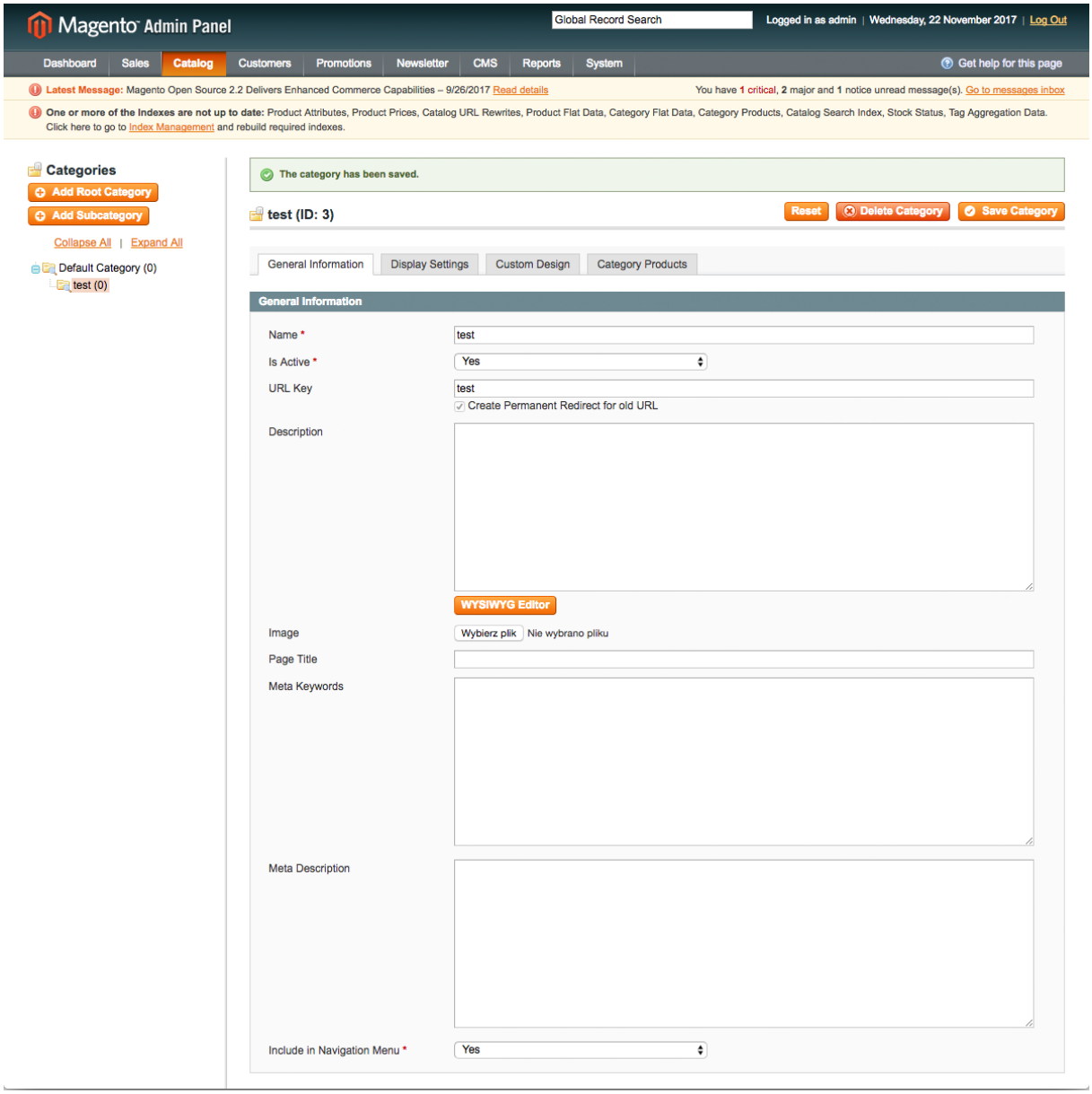
Rysunek 2.3: Magento: Ekran edycji zestawu cech produktu

liwa operacja na zbiorach.

2.2.4 Zarządzanie nawigacją

W Panelu Administracyjnym zarządzanie nawigacją praktycznie nie istnieje. Menu główne sklepu generowane jest na podstawie kategorii (Listing 2.1). Menu generowane jest w postaci listy <ul>. Podkategorie danej kategorii są zagnieżdżanie jako kolejne listy. Użytkownik decyduje o tym, czy kategoria ma wchodzić do menu.

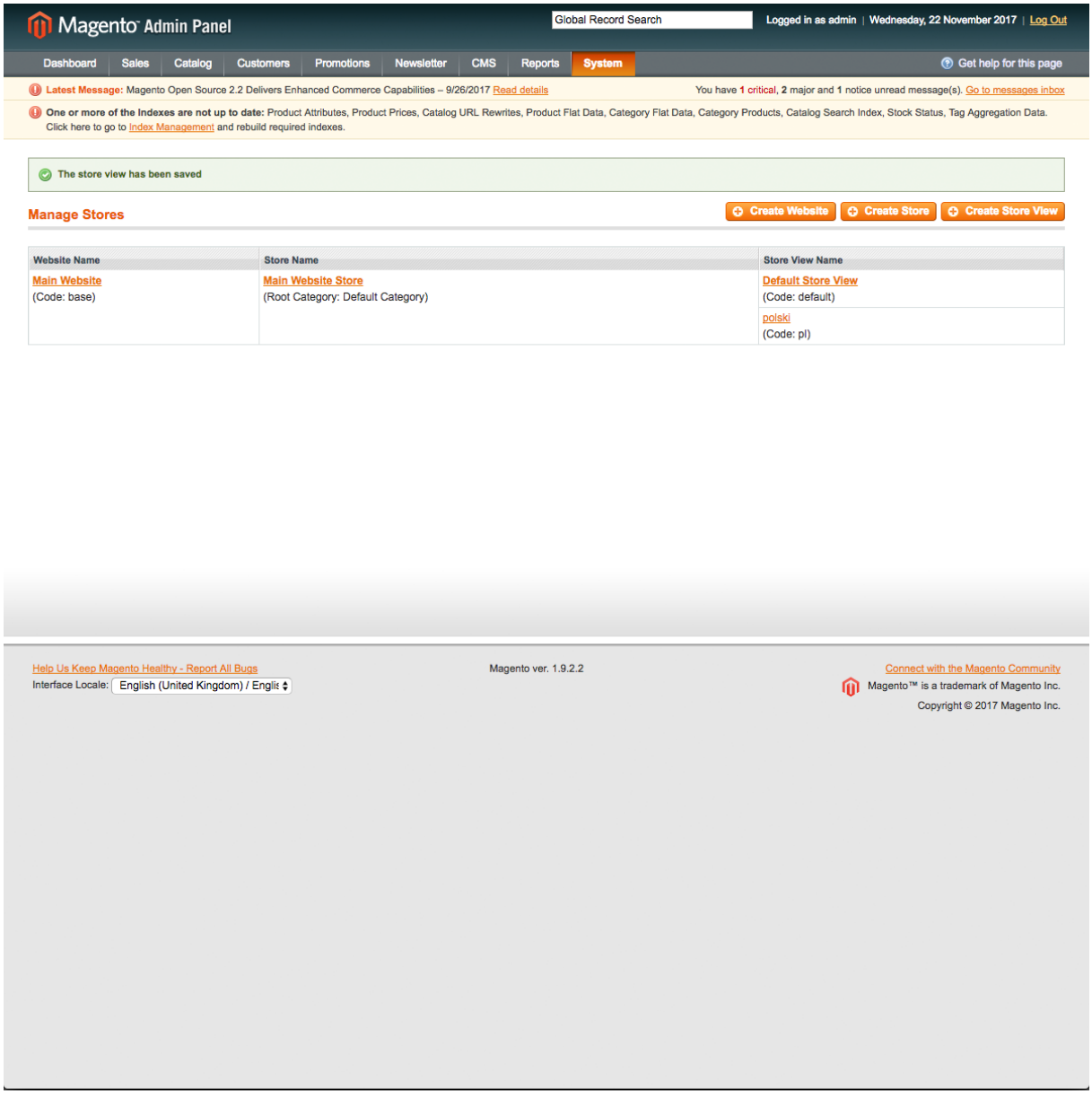
Listing 2.1: Funkcja generująca menu z kategorii



Rysunek 2.4: Magento: Ekran tworzenia kategorii

```
<?php
$_menu = $this->renderCategoriesMenuHtml(0,'level-top');
echo $_menu;
?>
```

Dodawanie do menu przekierowań na stronę statyczną, czy na zewnętrzną stronę jest niemożliwe. Dodatkowe linki, czy zmiana architektury menu jest możliwa tylko wyłącznie przez edycję kodu. Osoba nie potrafiąca programować nie może zarządzać nawigacją na sklepie. Jest to duża wada Magento.



Rysunek 2.5: Magento: Ekran dostępnych wersji językowych

2.2.5 Zarządzanie treścią

W Magento wszystkie ciągi znaków, które pojawiają się w szablonie przechowywane są w plikach CSV. Pliki CSV mają określone nazwy, tak aby system wczytywał tylko jeden plik. Aby dodać nowe tłumaczenie należy utworzyć katalog o skrócie języka, a następnie dodać pliki CSV, które mają strukturę "klucz - wartość". Pełna wersja językowa powinna zawierać 76 pliki CSV.

Listing 2.2: Fragment pliku `Mage_Adminhtml.csv` w wersji angielskiej

```
"Add Exception","Add Exception"
"Add Field Mapping","Add Field Mapping"
```

```
"Add Field with URL:","Add Field with URL:"
"Add New","Add New"
"Add New Block","Add New Block"
"Add New Image","Add New Image"
"Add New Profile","Add New Profile"
"Add New Role","Add New Role"
"Add New Template","Add New Template"
"Add New URL Rewrite","Add New URL Rewrite"
"Add New User","Add New User"
"Add New Variable","Add New Variable"
"Add Products","Add Products"
"Add URL Rewrite","Add URL Rewrite"
```

Ciągi znaków w szablonie zwracane są przez specjalną funkcję, która przyjmuję jako argument klucz. Funkcja sprawdza jaką wersję językową ma ustawioną użytkownik i sprawdza czy we właściwym pliku CSV istnieje odpowiedni klucz. Jeżeli tak to zwracana jest wartość z pliku CSV. Jeżeli nie istnieje, funkcja zwracana wartość z domyślnej wersji językowej sklepu. Jeżeli klucz nie występuje w domyślnej wersji sklepu jako tekst zwracana jest wartość klucza.

Listing 2.3: Użycie funckji wyświetlającej tekst.

```
<p class="note-msg">
<?php echo $this->__('There are no products matching
the selection.') ?>
</p>
```

Użytkownik w Panelu Administratora może utworzyć nową wersję językową strony. Jednak tłumaczenie, czyli pliki CSV musi sam utworzyć, a później wgrać je przez klienta FTP.

2.2.6 Zarządzanie pozostałymi rodzajami treści

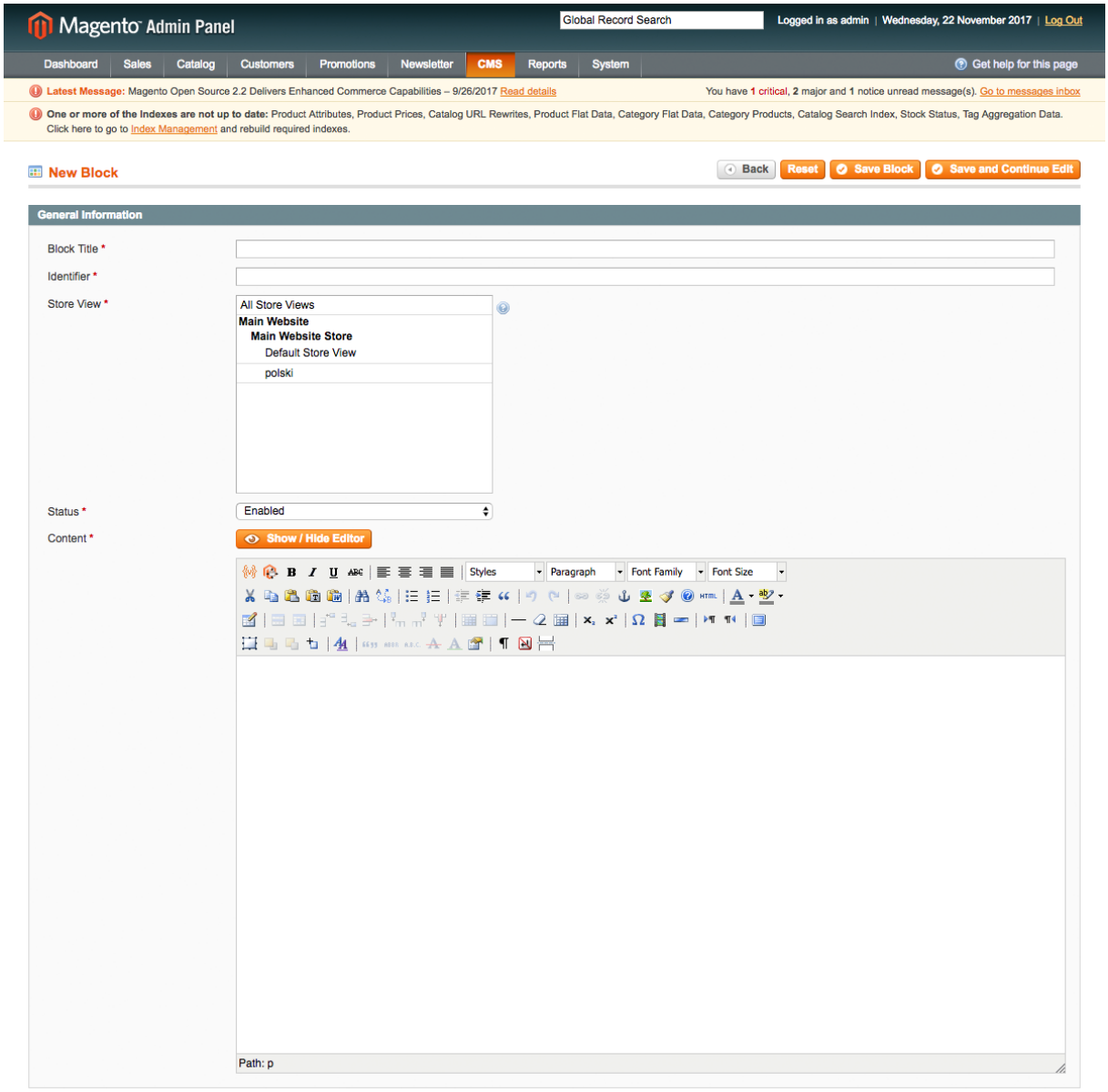
Magento oferuje system CMS, który umożliwia tworzenie modułów statycznych (fragmentów HTML) oraz stron.

Moduły statyczne

W modułach statycznych możemy dodać dowolną treść, dzięki edytorowi WYSIWYG (Rysunek 2.6). Jednak treści tworzone przez osobę bez doświadczenia informatycznego w tym edytorze będą proste (treścią może być pole tekstowe, obraz, tabela). Istnieje możliwość stworzenia skomplikowanych modułów, lecz wymaga to znajomości HTML i JavaScript.

Tak przygotowany moduł można umieścić do szablonu Magento. Jednak aby to zrobić potrzebny jest dostęp do FTP i dodanie w odpowiednim pliku fragmentu kodu z Listingu 2.4.

Listing 2.4: Dodanie bloku statycznego 'bloczek' do szablonu



Rysunek 2.6: Magento: Ekran tworzenia modułu statycznego

```
<?php
echo $this->getLayout()->createBlock('cms/block')
        ->setBlockId('bloczek')->toHtml()
?>
```

Istnieje też możliwość dodania bloku statycznego do konkretnej strony z poziomu Panelu Administracyjnego. W tym wypadku nie potrzebujemy dostępu do FTP, ale nie obędzie się bez kodu. Aby dodać blok statyczny do strony należy przejść do zakładki 'Design' i wpisać kod przedstawiony Listningu 2.5. Wspomniany kod dodaje do sekcji zdefiniowanej w szablonie jako 'left' (użytkownik musi mieć wiedzę, jak skonsturowany jest szablon jego strony), bloku statycznego o identyfikatorze 'bloczek'.

Listing 2.5: Dodanie bloku statycznego ’bloczek’ do strony w Magento

```
<reference name="left">
<block type="cms/block" name="bloczek" before="-">
    <action method="setBlockId">
        <block_id>bloczek</block_id>
    </action>
</block>
</reference>
```

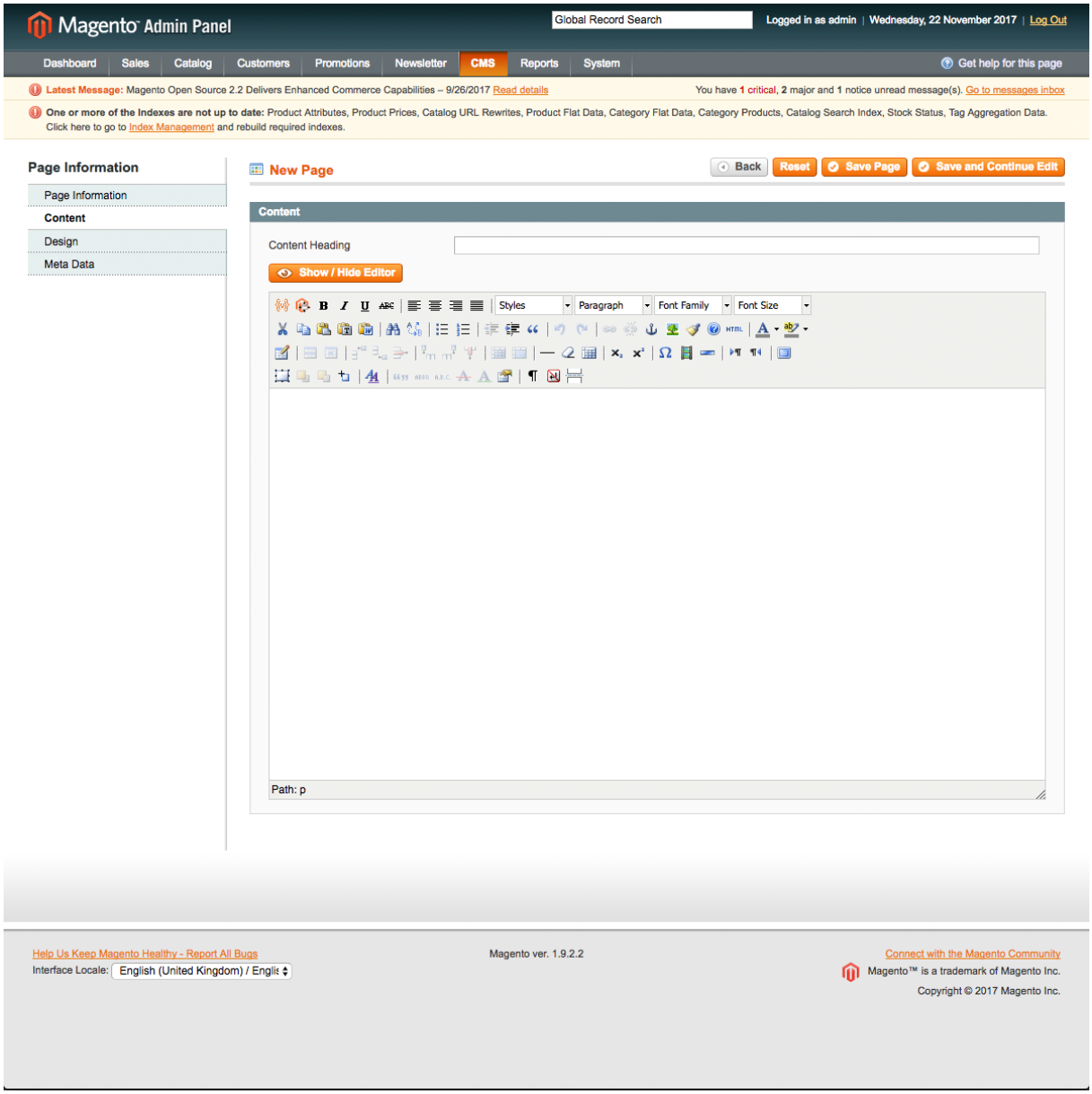
Ostatnią metodą umieszczenia bloku statycznego do konkretnej strony jest najprostsza. Blok statyczny może być umieszczony jako element treści wybranej strony statycznej przez fragment kodu z Listingu 2.6. Jest to najprostsze rozwiązanie, ale wciąż użytkownik może mieć problemy z jego dodaniem. Kodu tego nie wolno dodać w trybie WYSIWYG, ponieważ zostanie potraktowany on jako tekst.

Listing 2.6: Dodanie bloku statycznego ’bloczek’ do treści strony w Magento

```
{{block type="cms/block" block_id="bloczek"}}
```

Strony

Tworzenie nowej strony statycznej jest bardzo podobne do tworzenia modułu statycznego. Również mamy edytor WYSIWYG. Możemy do strony dodawać moduły statyczne, tak jak to pokazano w poprzedniej sekcji.



Rysunek 2.7: Magento: Ekran tworzenia strony

2.3 osCommerce

osCommerce jest jednym z pierwszych systemów sprzedażowych, który został upubliczniony na licencji Open Source. Premiera tego systemu nastąpiła w 2000 roku. W latach 2000-2008 system ten był bardzo popularnym rozwiązaniem, jednak w 2008 roku pojawił się system Magento. Ponieważ osCommerce był rzadko aktualizowany tracił popularność na rzecz regularnie rozwijanego konkurenta. Obecnie osCommerce to drugi system dedykowany sprzedaży w Internecie. Jego popularność waha się w okolicach 5%.

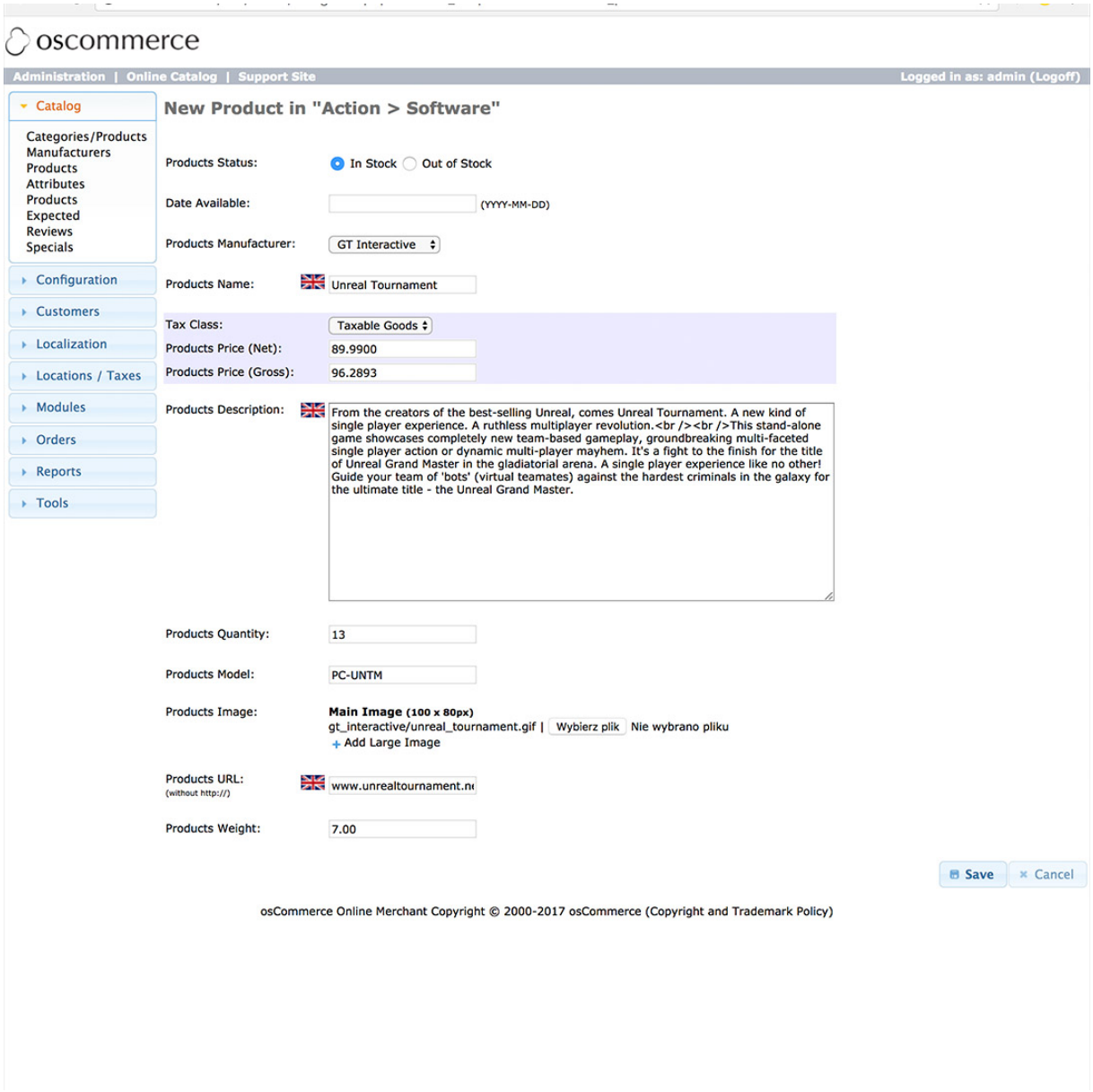
2.3.1 Dodawanie produktów

Formularz dodawania produktu jest bardzo prosty w osCommerce. Ma on o wiele mniej pól niż Magento. Produktowi można przypisać takie wartości jak: nazwa produktu, status, data dostępności, typ podatku, cenę, producenta, opis, liczbę w magazynie, model produktu, duże zdjęcie i miniaturkę oraz adres URL. Rysunek 2.8 przedstawia formularz dodawania produktów.

Produkt można dodać bardzo szybko, ale w dzisiejszych czasach z pewnością te pola nie wystarczą by dobrze opisać produkt. Pole na opis produktu nie jest edytorem WYSIWYG. Aby sformatować opis produktu należy użyć kodu HTML. Brakuje możliwości dodania większej ilości zdjęć do produktu. Formularz nie sprawdza poprawności wprowadzonych danych.

2.3.2 Zarządzanie cechami produktów

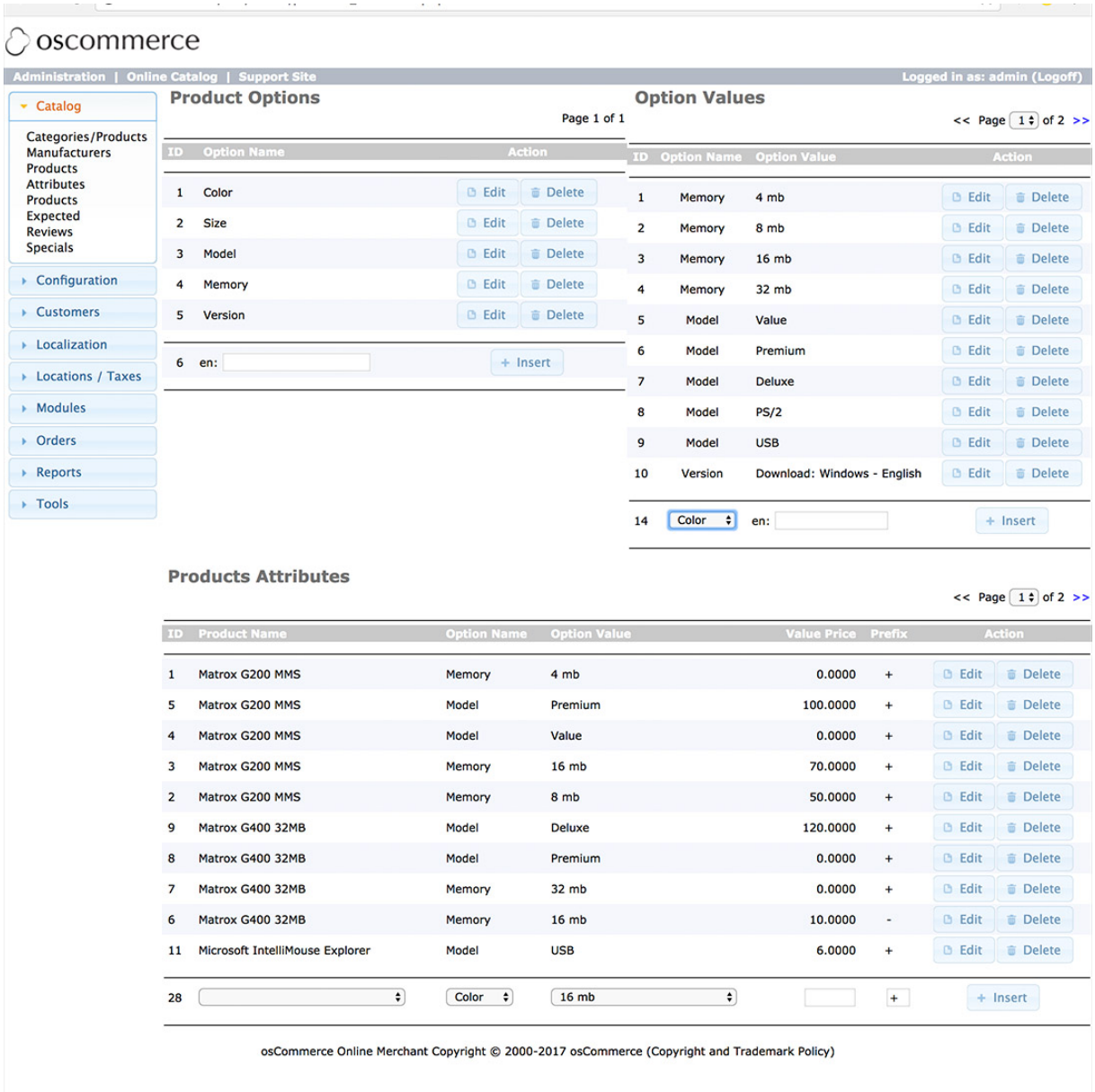
Produkty w osCommerce mogą mieć cechy, które użytkownik może wybrać z listy rozwijanej. Cechy te przypisuje się w sekcji "Product attributes". Ekran podzielony jest na 3 części, (rysunek 2.9). W lewym rogu znajduje się lista cech, w prawym rogu wartości jakie posiada cecha. Są to wszystkie wartości wszystkich cech. Po dodaniu cechy i przypisaniu cech produktów na dole strony można przypisać wartość cechy do produktu. Zarówno produkt, cechę oraz wartość cechy wybiera się z listy rozwijanej. Jednak listy rozwijane zawierają wszystkie produkty, wszystkie cechy i wszystkie wartości cech. Problemy jakie wynikają z takiego rozwiązania mogą pojawić się przy dużej ilości produktów, kiedy użytkownikowi będzie miał trudność ze znalezieniem odpowiedniego produktu. Kolejną wadą tego rozwiązania jest możliwość dodania do cechy wartości, która nie jest powiązana z cechą (np. można dodać do cechy kolor wartość 16 mb przypisaną do cechy pamięć). Walidacja, podobnie jak przy dodawaniu produktów, nie istnieje.



Rysunek 2.8: osCommerce: Ekran dodawania nowego produktu.

2.3.3 Katalogowanie produktów

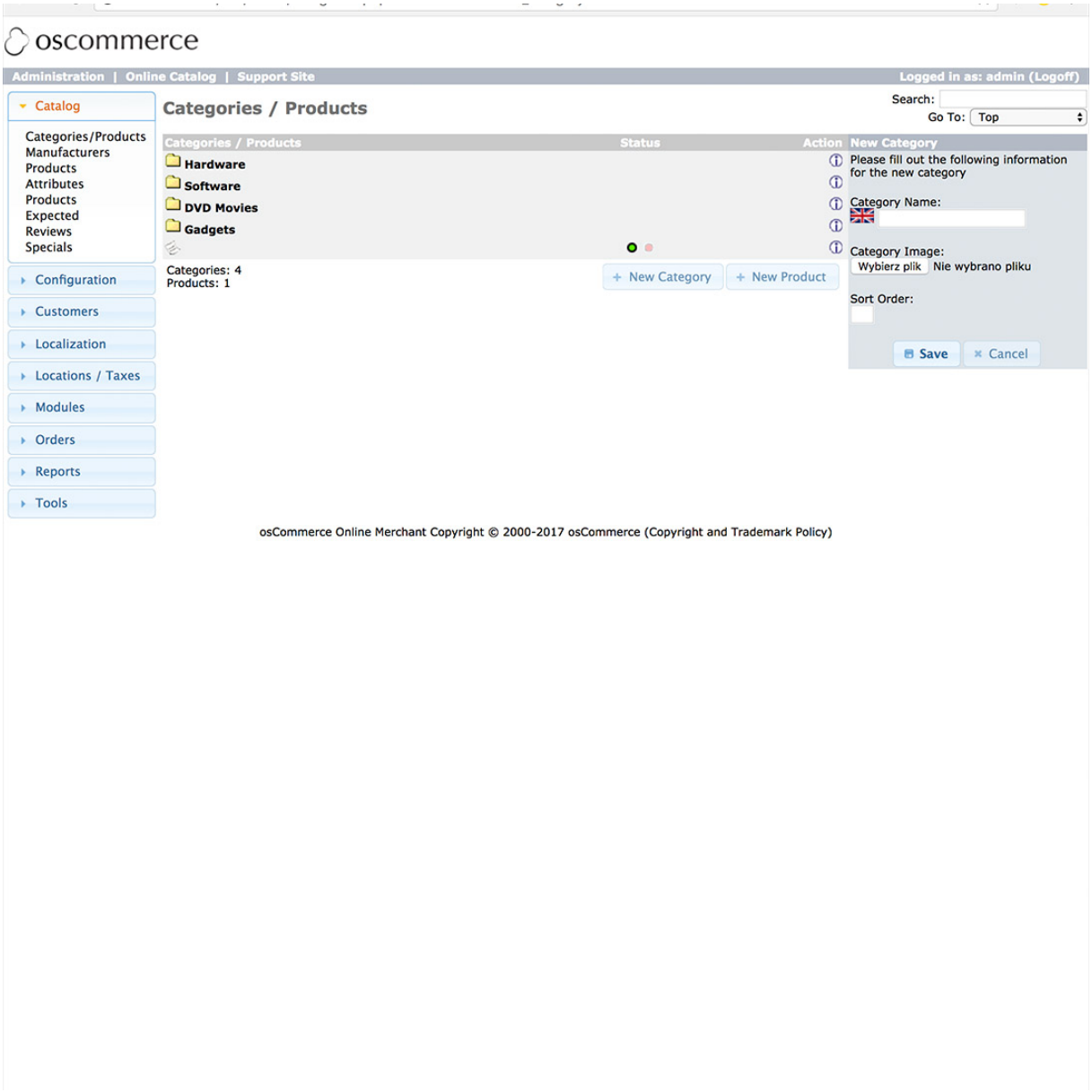
Katalogowanie produktów w osCommerce przypomina system plików systemów operacyjnych. Na Rysunku 2.10 przedstawione jest główne okno sekcji Kategorie. Kategoria przedstawiona jest tutaj jako folder. Folder może mieć swoje podfoldery. W folderach mogą znajdować się produkty. Nie ma możliwości aby produkt znajdował się w dwóch folderach jednocześnie.



Rysunek 2.9: osCommerce: Cechy produktów.

2.3.4 Zarządzanie nawigacją

W osCommerce zarządzanie nawigacją nie istnieje. Użytkownik musi stworzyć nawigację sam w szablonie strony. Aby to zrobić użytkownik musi umieć tworzyć kod HTML, CSS i PHP oraz posiadać wiedzę na temat działania skryptu osCommerce, by skorzystać z odpowiednich funkcji. Istnieje również ryzyko, że żadna z funkcji nie będzie spełniała oczekiwań użytkownika, wtedy potrzebna będzie mu wiedza na temat baz danych i znajomość podstaw języka SQL, aby pobrać dane z bazy danych i przetworzyć je w skrypcie PHP.



Rysunek 2.10: osCommerce: Katalog produktów.

2.3.5 Tłumaczenie

W osCommerce tłumaczenie przechowywane jest w stałych PHP. Stworzenie nowej wersji językowej wymaga utworzenia wielu plików (43 pliki w domyślnej wersji angielskiej) i przesłanie ich na serwer, a następnie dodanie wersji językowej z poziomu Panelu Administracyjnego i wskazanie ścieżki do katalogu z tłumaczeniami. Wymagana jest też podstawowa wiedza na temat języka PHP. Edycja tłumaczenia polega na edycji konkretnego pliku w katalogu z tłumaczeniami.

Listing 2.7: osCommerce: Plik account_history.php zawierający stałe z tłumaczeniem

```
<?php
```

```
/*
  $Id$

  osCommerce, Open Source E-Commerce Solutions
  http://www.oscommerce.com

  Copyright (c) 2003 osCommerce

  Released under the GNU General Public License
*/

define('NAVBAR_TITLE_1', 'My Account');
define('NAVBAR_TITLE_2', 'History');

define('HEADING_TITLE', 'My Order History');

define('TEXT_ORDER_NUMBER', 'Order Number:');
define('TEXT_ORDER_STATUS', 'Order Status:');
define('TEXT_ORDER_DATE', 'Order Date:');
define('TEXT_ORDER_SHIPPED_TO', 'Shipped To:');
define('TEXT_ORDER_BILLED_TO', 'Billed To:');
define('TEXT_ORDER_PRODUCTS', 'Products:');
define('TEXT_ORDER_COST', 'Order Cost:');
define('TEXT_VIEW_ORDER', 'View Order');

define('TEXT_NO_PURCHASES', 'You have not yet made any purchases.');
?>
```

2.3.6 Zarządzanie pozostałymi rodzajami treści

osCommerce nie ma żadnego systemu zarządzania treścią stron statycznych. Strony statyczne reprezentowane są jako pliki PHP. Listing 2.8 pokazuje przykładowy plik ze stroną statyczną. Na początku pobrane zostają stałe z plików językowych. W przypadku pliku shipping.php istnieją dwie stałe HEADING_TITLE oraz TEXT_INFORMATION. Oprócz tego ładowane są pliki szablonu takie jak template_top.php, czy application_bottom.php.

Listing 2.8: osCommerce: Plik shipping.php ze stroną statyczną.

```
<?php
/*
  $Id$

  osCommerce, Open Source E-Commerce Solutions
  http://www.oscommerce.com

  Copyright (c) 2010 osCommerce

  Released under the GNU General Public License
*/

  require('includes/application_top.php');
```

```
  require(DIR_WS_LANGUAGES . $language .
  '/' . FILENAME_SHIPPING);

  $breadcrumb->add(NAVBAR_TITLE,
  tep_href_link(FILENAME_SHIPPING));

  require(DIR_WS_INCLUDES . 'template_top.php');
?>

<h1><?php echo HEADING_TITLE; ?></h1>

<div class="contentContainer">
  <div class="contentText">
    <?php echo TEXT_INFORMATION; ?>
  </div>

  <div class="buttonSet">
    <span class="buttonAction">
    <?php
    echo tep_draw_button(IMAGE_BUTTON_CONTINUE,
    'triangle-1-e', tep_href_link(FILENAME_DEFAULT));
    ?>
    </span>
  </div>
</div>

<?php
  require(DIR_WS_INCLUDES . 'template_bottom.php');
  require(DIR_WS_INCLUDES . 'application_bottom.php');
?>
```

Listing 2.8 pokazuje co musi zrobić użytkownik, aby dodać stronę statyczną. Użytkownik musi utworzyć pliki z tłumaczeniem (ich struktura została opisana w punkcie 2.3.5). Następnie utworzyć stronę statyczną w głównym katalogu sklepu i stworzyć plik z szablonem, używając stałych i pobierając komponenty używane na reszcie strony. Aby użytkownik mógł sam dodać stronę statyczną musi posiadać wiedzę na temat PHP, HTML, CSS oraz być zaznajomiony z zasadami działania systemu osCommerce.

2.4 Podsumowanie

Opisane wyżej systemy działają od wielu lat na rynku i mają wiele zalet. Magento posiada bardzo zaawansowany system cech, który pozwala na stworzenie praktycznie każdej konfiguracji i dowolny produkt. osCommerce jest natomiast zbudowany z bardzo prostych modułów i programisty bardzo szybko jest w stanie edytować kod. Podstawową wadą opisanych wyżej systemów sprzedażowych, które zarazem są najpopularniejszymi systemami, jest to, że ich docelowym odbiorcą nie jest przedsiębiorca, których chce sprzedawać swoje produkty w Internecie. Aby stworzyć sklep internetowy na bazie tych systemów, który będzie zgodny prawem, będzie intuicyjny dla klientów, albo chociaż nawet będzie posiadał dobrze skatalogowane produkty wymaga konfiguracji przez osobę, która posiada wiedze informatyczną na wysokim poziomie. Systemy te cechuje nieintuicyjność. W związku z tym nawet obsługa sklepu dla osoby z małym doświadczeniem informatycznym może być uciążliwa.

Rozdział 3

Wykorzystane narzędzia i technologie

W tym rozdziale zostaną zaprezentowane technologie, które zostały wykorzystane do stworzenia systemu sprzedażowego. Omówione zostaną również narzędzia, które znacząco wspomogły etap tworzenia oprogramowania.

3.1 Technologie

Podstawowy celem, który ma realizować aplikacja to stworzenie systemu sprzedażowego, który będzie mógł być obsługiwany przez osobę bez doświadczenia informatycznego.

Docelowy użytkownik nie powinien być zmuszany do wprowadzania zmian w kodzie aplikacji, korzystania z dodatkowych programów poza przeglądarką, instalacji dodatkowych wtyczek, czy korzystaniu z protokołów FTP lub SSH do wprowadzania zmian. Dlatego prototyp powinien umożliwiać wprowadzanie zmian z poziomu Panelu Administracyjnego, a oprócz tego powinien zostać napisany w językach, które każda przeglądarka obsługuje, czyli HTML, CSS i JavaScript.

3.1.1 JavaScript

JavaScript (w skrócie JS) to dynamiczny język programowania, który przez użytkowników internetu kojarzony był negatywnie. Na początku rozwoju sieci WWW skrypty JavaScript były wykorzystywane głównie do wyświetlania reklam popup, czy alertów.

Wraz z rozwojem sieci i rewolucji Web 2.0 JavaScript coraz częściej był wykorzystywany. Kiedy w 2010 roku Steve Jobs ogłosił, że iPhone oraz inne urządzenia mobilne nie będą posiadały Flasha i jako alternatywę proponuje HTML 5 i JavaScript wywołało to wiele kontrowersji [8]. Jednak z biegiem czasu okazało się, że był to właściwy ruch. W 2020 roku firma Adobe

całkowicie zamknie projekt Flash Playera. Od listu napisanego przez Jobsa JavaScript coraz bardziej zyskiwał na znaczeniu.

Obecnie takie firmy jak PayPal [15], Netflix [12] chętnie korzystają z JavaScriptu. Dzięki językowi JavaScript możliwe jest tworzenie dynamicznych stron internetowych.

3.1.2 Node.js

Node.js to środowisko uruchomieniowe JavaScript służące do wykonywania kodu JavaScript po stronie serwera. Node oparty jest na Chrome V8, silniku opracowanego przez firmę Google. Wykorzystywany jest również w m.in. przeglądarce Google Chrome. Napisany został w języku C++. Silnik ten implementuje specyfikację ECMAScript[7]. Node.js udostępniany jest na licencji Open Source. Pierwsze wydanie pojawiło się w 2009 roku.

Node został stworzony w oparciu o metodologię programowania sterowanego zdarzeniami. Posiada asynchroniczne I/O, a znacząca większość jego funkcji, które wykorzystuje programista, nie odwołuje się bezpośrednio do I/O. Dzięki takim rozwiązaniom twórca oprogramowania nie musi przejmować się zakleszczeniami znanymi z programowania wielowątkowego. Developer musi jednak zwrócić uwagę na to, aby tworzone przez niego funkcję były, o ile jest to możliwe, asynchroniczne oraz aby wszystkie zdarzenia rozbijać na jak najmniejsze podzdarzenia[13].

3.1.3 MongoDB

MongoDB to nierelacyjny system zarządzania bazą danych napisany w języku C++. MongoDB udostępniany jest na licencji GNU Affero General Public License. MongoDB jest dokumentową bazą danych. Struktura danych w Mongo jest elastyczna. Dane przechowywane są w formacie BSON.

BSON jest to plik binarny, który przechowuje zserlializowane dokumenty w stylu plików JSON. BSON wykorzystuje minimalną przestrzeń dyskową do przechowywania danych, a jednocześnie jest bardzo łatwy do zarządzania. Cechuję go wydajność, ponieważ używa tylko formatów danych znanych z języka C (C data types). Dzięki temu bardzo szybko można kodować oraz dekodować dane[1].

Format JSON (JavaScript Object Notation) wykorzystywany jest do komunikacji z bazą danych. Struktura pliku JSON to zbiór atrybutów w postaci klucz-wartość[9].

MongoDB to rozproszona baza danych. W związku z tym cechuje ją wysoka dostępność, skalowalność i łatwość w użyciu[11].

3.1.4 Mongoose

MongoDB zwraca pliki w formacie JSON, czyli obiekty JavaScript. Dzięki temu dane zwrócone przez bazę mogą od razu być przetwarzane. Jednak w aplikacji skorzystano z mongoose. Mongoose nie jest systemem ORM, ponieważ nie ma potrzeby modelowania obiektów w bazie nierelacyjnej, gdyż dane przechowywane w bazie są obiektami. Mongoose dodaje do obiektów przechowywanych w Mongo dodatkowe funkcjonalności takie jak np. walidacja, konwersja danych, budowanie zapytań, itd.

3.1.5 Vue.js

Vue.js to framework do języka JavaScript do budowania interfejsów użytkownika. Konkurencją dla Vue.js może być React, framework opracowany przez Facebooka oraz Angular, framework stworzony przez Google. Frameworki te są do siebie bardzo podobne. Vue.js wyróżnia prostota oraz modułowość[19]. Spośród nich według niezależnego frameworku to właśnie Vue.js jest najszybszym frameworkiem[10].

Element UI

Biblioteka webcomponentów zawierająca rozszerzone elementy DOM (Document Object Model) oraz unikalne elementy GUI m.in.:

1. notification - komponent do wyświetlania powiadomień ,
2. upload - komponent do wgrywania zdjęć,
3. pagination - komponent zarządzania paginacją,
4. tabs - komponent wyświetlający treści w postaci kart znanych z przeglądarek.

Vuex

Biblioteka dodająca do Vue.js Centralized state management. Jest to miejsce w aplikacji w której przechowywane są zasoby do których dostęp powinno mieć wiele komponentów [20].

Zmiana zmiennej przechowywanej w State Managment można być dokonana przez komponent wywołując specjalnie przygotowaną akcję (action). Komponenty oczekują na eventy propagowane przez Vuex w których zawarta jest informacja o zmianie zmiennej. Kiedy event zostanie odebrany przez komponent, który wykorzystuję daną zmienną zostanie ona zaktualizowana.

Dzięki Vuex możliwe jest utworzenie aplikacji, która będzie odświeżała się w tle.

Vue Router

Typowe aplikacje webowe działają na zasadzie:

1. Klient wysyła żądanie do serwera o zasoby za pośrednictwem URL (Uniform Resource Locator)
2. Serwer przetwarza URL i wysyła zasób, który jest przypisany do strony głównej.

Większość aplikacji webowych jako zasób traktują całe strony. Kiedy użytkownik chce przejść do innej strony musi ponownie wykonać zapytanie do serwera, czyli przejść do innego adresu url przez hieperłącze, bądź wprowadzając adres URL do przeglądarki. Następnie przeglądarka musi wczytać ponownie całą stronę. A biorąc pod uwagę na doświadczenia użytkownika (user experience) nie jest to najlepsze rozwiązanie.

Vue Router to biblioteka, która 'blokuje' domyślne zachowanie przeglądarki. W aplikacji napisanej we Vue wydzielane jest miejsce na stronie w której ma wykonywać się komponent VueRouter. Kiedy użytkownik zmienia adres URL nie przeładowuje mu się cała strona. VueRouter mapuje, który komponent ma być wyświetlony w zdefiniowanym miejscu przy danym adresie URL.

Zapytania do serwera o nowe zasoby są wykonywane w tle. Komponenty przypisane do danego adresu URL mogą również w ogóle nie wysyłać żądania o zasoby.

3.1.6 Docker

Docker to platforma do uruchamiania aplikacji rozproszonych. Wyróżnia się prostotą i otwartością. Jest udostępniony na licencji Apache License 2.0 [3].

Docker pozwala umieścić aplikację w kontenerze. Kontener to lekkie i niezależne od siebie pakiety oprogramowania. Izolują one oprogramowanie od otoczenia, dzięki czemu niweluje on różnice między deweloperami, a także między różnymi systemami.

Własne obrazy Dockera tworzy się w pliku Dockerfile i ma on bardzo prosty schemat widoczny w listingiu 3.1. W pliku tym zawsze wybieramy postawowy obraz, który jest upubliczniony na Docker Hub (w przykładzie jest to obraz Nginx w wersji Alpine, czyli lekkiej dystrybucji Linuksa).

Listing 3.1: Plik Dockerfile tworzący kontener Dockera do aplikacji Vue.

```
FROM nginx:alpine
WORKDIR /app

RUN apk --no-cache add --virtual nodejs nodejs-npm

COPY package.json /app
RUN npm install
COPY . /app

RUN npm run build

RUN  rm -rf /usr/share/nginx/html
       && mv /app/dist/ /usr/share/nginx/html/

EXPOSE 80
```

3.2 Narzędzia i środowisko pracy

W tej sekcji zostaną opisane kluczowe narzędzia, które zostały użyte podczas tworzenia aplikacji.

3.2.1 Visual Studio Code

Zaawansowany edytor tekstowy stworzony przez Microsoft bazujący na edytorze Atom. Edytor posiada wiele przydatnych funkcji przy pisaniu kodu w JavaScript.

3.2.2 npm

Npm (node package manager) to menedżer pakietów dla języka JavaScript. Dzięki niemu w bardzo szybki sposób można zainstalować dodatkowe biblioteki. Listing 3.2 pokazuję jak prosto zainstalować przykładową bibliotekę.

Listing 3.2: Instalacja biblioteki Vuex przez npm.

```
npm install --save vuex
```

3.2.3 chai i mocha

Dwie biblioteki do testowania. Biblioteka mocha służy do tworzenia testów, a biblioteka chai do sprawdzania typów, wartości, długości zmiennych.

3.2.4 nodemon

Biblioteka, która monitoruje wszelkie zmiany w kodzie i automatycznie restartuje aplikację.

3.2.5 eslint

W języku JavaScript nie występuje kompilator. Eslint to narzędzie, które dba o jakość pisanego kodu. Sugeruję m.in. stosowanie funkcji asynchronicznych, stosowanie odpowiednich wcięć w kodzie itd. Eslint jest bardzo dobrze obsługiwany przez Visual Studio Code.

3.2.6 Git

Rozproszony system kontroli wersji. Git jest darmowym systemem udostępnionym na podstawie licencji GNU GPL w wersji 2. Cechuje go lekkość oraz szybkość [5]

3.2.7 Github

Hosting dla wszystkich projektów programistycznych, które działają w oparciu o system Git. Github posiada wiele integracji, które zostały wykorzystsane podczas pisania pracy[6].

3.2.8 Docker Hub

Docker Hub serwis w chmurze, który przechowuje wszystkie obrazy utworzone przez użytkowników. Dzięki ogólnej dostępności obrazy mogą być bardzo szybko instalowane na różnych maszynach. Wykorzystując obrazy Dockera programista ma pewność, że niezależnie od maszyny na jakiej będzie działał kontener, środowisko w którym będzie działać aplikacja będzie zawsze takie same[2].

3.2.9 Watchtower

Obraz Dockera, który obserwuje serwis Docker Hub i sprawdza czy wybrane obrazy są aktualne. Jeżeli w serwisie Docker Hub pojawi się nowa wersja obrazu zostanie ona pobrana, skompilowana, a następnie wszystkie kontenery, które bazują na tym obrazie zostają zaktualizowane. Dzięki obrazowi aktualizacja aplikacji na testowym serwerze następuję automatycznie w momencie 'wypchnięcia' (git push) nowej wersji na serwer źródłowy (serwis GitHub)[18].

3.2.10 Robo 3T

Graficzny klient do bazy danych MongoDB. Zawiera wiele przydatnych narzędzi, które przyspieszają pracę nad bazą Mongo. To rozwiązanie Open Source działa na systemy operacyjne: Windows, Mac OS X oraz Linux[16].

Rozdział 4

Propozycja nowych rozwiązań

Celem pracy magisterskiej jest stworzenie systemu sprzedażowego, którym będzie mogła zarządzać osoba bez doświadczenia informatycznego. W rozdziale 2 zostały opisane dostępne na rynku systemy sprzedażowe. Wykazano wady zastosowanych rozwiązań, które sprawiają, że wprowadzanie zmian w systemie wymaga od użytkownika wiedzy informatycznej.

Aby zrealizować cel i jednocześnie wyeliminować wady obecnych rozwiązań należy zbudować system sprzedażowy w oparciu o trzy zasady:

1. Zero kodu - zmiany w sklepie powinny być możliwe bez znajomości jakiegokolwiek języka programowania. Obecne rozwiązania wymagają znajomości HTML nawet przy dodawaniu opisu produktu.
2. Tylko przeglądarka - jedynym narzędziem potrzebnym do wprowadzenia zmian jest przeglądarka internetowa w domyślnej konfiguracji. Klienci FTP, edytory tekstowe, wtyczki do przeglądarek, czyli narzędzia, które są niezbędne do wprowadzania zmian w Magento czy osCommerce, dla użytkownika bez doświadczenia informatycznego mogą być trudne do konfiguracji i obsługi.
3. Intuicyjne GUI - należy stworzyć czytelny i intuicyjny interfejs użytkownika, który będzie reagował na zachowania użytkownika.

Koncepcja będzie umożliwiała wprowadzanie zmian bez znajomości HTML przede wszystkim dzięki użyciu Web Componentów, które ukrywają od HTML, CSS i JavaScript. Web Componenty będą konfigurowane przez parametry, których edycja będzie możliwa z poziomu Panelu Administracyjnego. Oprócz tego tam gdzie umieszczanie treści na stronę przez użytkownika, np. przy tworzeniu opisów produktów, będzie obsłużone przez edytor WYSIWYG.

System sprzedażowy będzie mógł być zarządzany jedynie przez przeglądarkę, dzięki zastosowaniu języków i rozwiązań, które są przez nią domyślnie wgrane. Panel Administracyjny

będzie stworzony tylko w językach HTML, CSS oraz JavaScript. Wprowadzanie zmian na stronie, edycja podstron, zarządzanie nawigacją, tworzenie modułów będzie możliwe z poziomu Panelu Administracyjnego.

Intuicyjne GUI zostanie zrealizowane w prototypie dzięki stworzeniu Panelu Administracyjnego w formie Single Page Application. Jest to koncepcja, która staje się coraz bardziej popularna dzięki rozwojowi aplikacji mobilnych. Aplikacje webowe, które mają charakter Single Page Application wyróżnia dynamiczna interakcja z użytkownikiem.

Poniżej zostaną omówione szerzej koncepcje Web Componentów oraz Single Page Application.

4.1 Web Components

Web Components to zestaw specyfikacji, które pozwalają na stworzenie własnych elemntów DOM (Document Object Model).

4.1.1 Specyfikacje

Web Components opiera się na czterech specyfikacjach, które pojawiły się w HTML5.

Custom elements

HTML5 daje możliwość twórcy oprogramowania tworzenia swoich własnych znaczników. Znaczniki tym można zmieniać wygląd przez CSS oraz kontrolować ich zachowanie przez skrypty JavaScript, tak jak inne standardowe znaczniki HTML [21].

Shadow DOM

Standardowy plik HTML zawiera wszystkie znaczniki w jednym pliku. Architektura pliku HTML opiera się na zasadzie drzewa. Shadow DOM daje możliwość enkapsulacji znaczników HTML. Własne znaczniki opisane wyżej mogą mieć swój shadow root (inne znaczniki HTML), który jest rzeczywiście wyświetlany w miejscu znacznika [23].

HTML imports

Możliwość wczytywania i ponownego użycia dokumentów HTML w innych dokumentach HTML [22].

HTML template

Znacznik <template> to znacznik, który jest pomijany podczas wczytywania strony, ale może być zainicjowany później [24].

4.1.2 Zastosowanie

Dzięki Web Componentom można zbudować wiele różnych modułów jak np. slider czy menu. Cała struktura HTML, wygląd CSS oraz obsługa zachowań opisana przez skrypt JavaScript będzie ukryta w Web Componencie. W szablonie głównym pojawi się jedynie znacznik np. <slider>.

Znacznik ten będzie przyjmować różne parametry podobnie jak znaczniki HTML. Parametry te będą definiowały zachowanie componentu. Przykładowy komponent <slider> może mieć parametr speed, który będzie określał szybkość zmiany banerów. Parametry te będą przechowywane w bazie danych. Użytkownik będzie mógł je edytować z poziomu Panelu Administracyjnego.

Dzięki Web Componentom użytkownik nie musi znać języka HMTL, CSS, czy JavaScript. Kod jest ukryty. Web Componenty rozwiązują problem braku znajomości przez użytkownika języków programowania.

4.1.3 Przykład Web Componentu

W tej sekcji zostanie zaprezentowany prosty Web Component stworzony w oparciu o framework Vue.js. Web Component będzie wyświetał aktualną datę w różnych wersjach językowych.

W Web Componencie można wyróżnić trzy części dokumentu:

1. część HTML,
2. część JavaScript,
3. część CSS.

HTML

Oznaczona znacznikiem <template> część HTML odpowiada za strukturę, która jest dodawana do drzewa DOM głównego dokumentu HTML. W listingu 4.1 szablon jest bardzo prosty. Jest to znacznik <p>, znany z HTML jako znacznik akapitu. W środku znacznika znajduje w podwójnym nawiasie klamrowym nazwa zmiennej date. Framework Vue.js wyświetla w tym miejscu zmienną, która jest zadeklarowana w części JavaScript.

JavaScript

W Web Componentach najważniejszym fragmentem kodu jest skrypt JavaScript. Fragment ten oznaczony jest znacznikiem <script>. W pierwszej części importujemy biblioteki, komponenty oraz inne pliki JavaScript, które wykorzystujemy w obrębie komponentu. W przykładzie komponentu z listingu 4.1 korzystamy z biblioteki Moment.js.

Następnie pojawia się eksportowany obiekt JavaScript. Obiekt ten jest odpowiedzialny za całą obsługę komponentu. W tablicy props znajdują się atrybuty jakie przyjmuje obiekt. W przykładowym komponencie jedynym obsługiwanym atrybutem jest atrybut lang, który odpowiada za wersję językowa w jakim wyświetlany jest data. Kolejną pozycją obiektu jest funkcja data. Funkcja ta zwraca obiekt ze wszystkimi zmiennymi, które mogą być używane we wszystkich innych funkcjach oraz w części HTML. Zwracany przez funkcję data obiekt posiada trzy zmienne:

1. date - zmienna w której przechowywana będzie aktualna data,

2. format - zmienna odpowiedzialna za format daty przechowywanej w zmiennej date,

3. l - zmienna do której będzie przypisany język.

Funkcja created wywoływana jest po utworzeniu komponentu. Można ją przyrównać do konstruktora z wielu języków obiektówych np. Java. W tej funkcji sprawdzamy, czy język został podany i jeżeli tak to zostanie przypisany do zmiennej l, w przeciwnym wypadku język zostanie ustawiony na polski. Następnie tworzymy interwał, który będzie wywoływał funkcję getDate co sekundę.

W zmiennej methods przechowywany jest obiekt z funkcjami. Wywołanie funkcji z metody methods jest możliwe przez referencję this.

CSS

Ostatnią sekcją komponentu odpowiedzialna jest za styl. Opisana jest przez znacznik <style>. Parametr scoped dodany do tego znacznika powoduje, że styl będzie dotyczył tylko tego komponentu i nie będzie on wpływał na inne komponenty.

Listing 4.1: Instalacja biblioteki Vuex przez npm.

```
<template>
  <p>{{ date }}</p>
</template>
<script>
import moment from 'moment'
export default {
  props: ['lang'],
  data () {
    return {
      date: '',
      format: 'dddd D MMMM YYYY HH:mm:ss',
      l: 'pl'
    }
  },
  created () {
    if (typeof this.lang === 'undefined') {
      this.l = 'pl'
    } else {
      this.l = this.lang
    }
    this.date = this.getDate()
    setInterval(this.getDate, 1000)
  },
  methods: {
    getDate () {
      this.date = moment().locale(this.l)
                          .format(this.format)
    }
  }
}
</script>
<style scoped>
p {
  color: blue;
  font-size: 20px;
  line-height: 30px;
  font-family:'Gill Sans', 'Gill Sans MT',
              Calibri, 'Trebuchet MS', sans-serif
}
</style>
```

Dodanie WebComponentu do dokumentu HTML

Tak przygotowany komponent bardzo łatwo można dodać do dokumentu HTML. W listingu 4.2 pokazano fragment kodu HTML inicjujący cztery instacje komponentu.

Listing 4.2: Instalacja biblioteki Vuex przez npm.

```
<timer />
<timer lang="en" />
<timer lang="fr" />
<timer lang="de" />
```

Rezultat widoczny jest na obrazku 4.1

4.2 Single Page Application

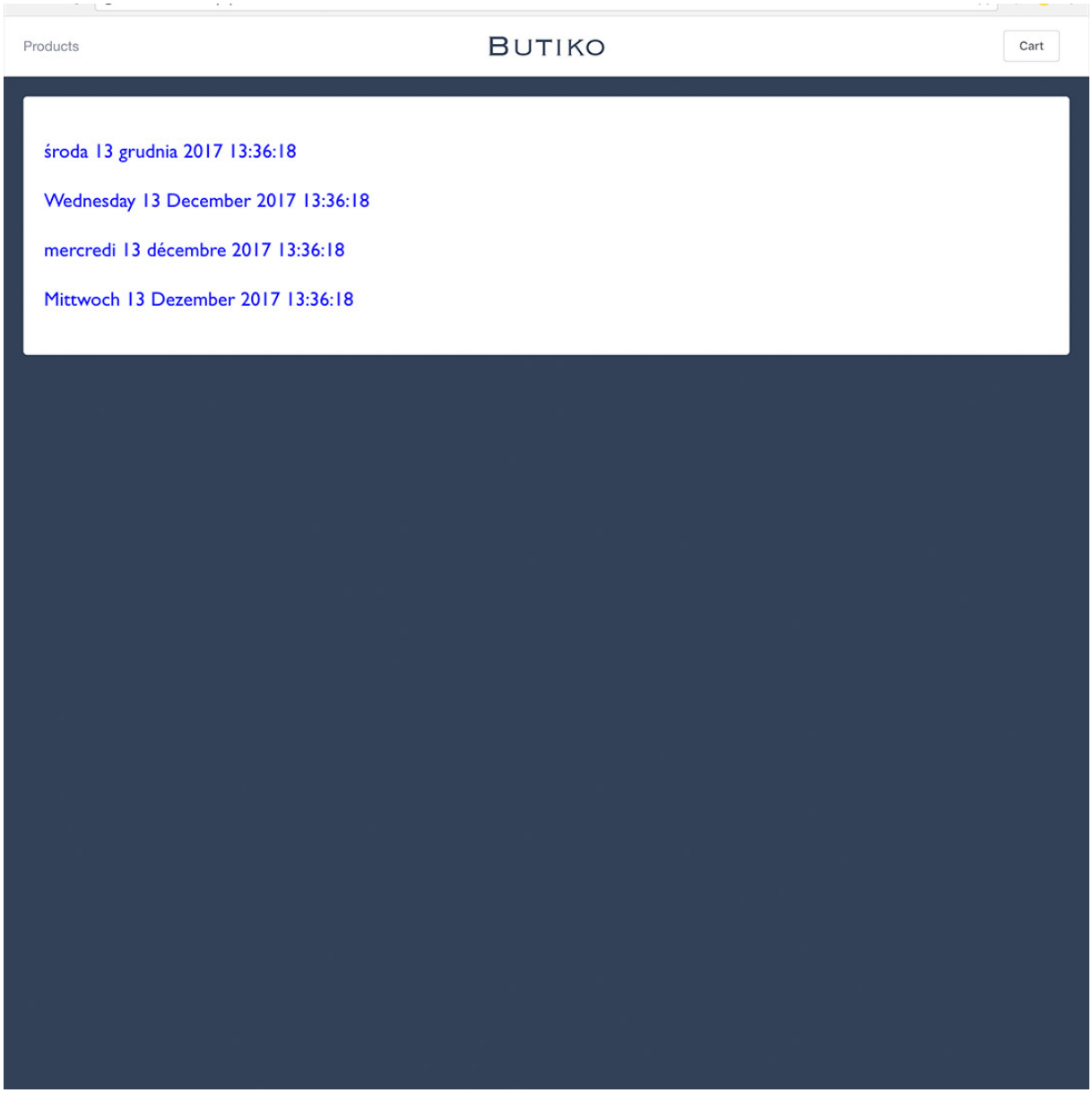
Koncepcja projektowania stron internetowych na zasadzie Single Page Application staje się coraz bardziej popularna.

Koncepcja ta polega na wczytaniu całej strony tylko raz. Przejście na inną podstronę polega na dynamicznej zmianie komponentu. W przeglądarce zmienia się adres URL, ale użytkownik wciąż pozostaje na tej samej stronie. Zmiana adresu URL skutkuje asynchronicznym wywołaniem skryptu JavaScript. Skrypt ten może m.in. zmienić treść komponentu, zmienić wyświetlany komponent, załadować dodatkowy komponent, zmienić układ strony, czy też wysłać zapytanie do serwera, a następnie przetworzyć odpowiedź i wyświetlić ją w komponencie.

Rozwiązanie tradycyjne, gdzie odpowiedzią serwera jest nowy dokument HTML reaguje na działania użytkownika tylko wtedy kiedy użyje on hiperłącza, bądź wyśle formularz. Dołączając do tradycyjnego rozwiązania bibliotekę JavaScript np. JQuery strona może reagować na dużo więcej działań użytkownika.

Jednak aplikacja webowa zbudowana na zasadach Single Page Application w całości wykorzystuje możliwości języka JavaScript. Aplikacja może reagować na takie samo zachowanie co tradycyjne rozwiązanie z użyciem JavaScript, jednak rekcje te są o wiele bardziej przyjazne dla użytkownika.

Zachowanie strony zbudowanej jako Single Page Application przypomina w działaniu aplikacje mobilne i rozwiązanie Single View Application. Urządzenia mobilne cieszą się coraz większą popularnością. Użytkownicy używają wielu aplikacji i ich doświadczenie z technologiami wzrasta właśnie dzięki urządzeniom mobilnym. Dlatego strona, która będzie zbudowana podobnie jak aplikacja mobilna oraz będzie reagowała podobnie jak aplikacja mobilna będzie bardziej przyjazna dla użytkownika niż tradycyjne rozwiązanie.



Rysunek 4.1: Cztery instancje Web Componentu Timer.

Użycie koncepcji Single Page Application sprawi, że zostanie zrealizowana trzecia zasada, która wyeliminuje mankamenty obencie istniejących systemów, czyli intiucyjne GUI.

Rozdział 5

Prototyp

W tym rozdziale zostanie zaprezentowany propozycję systemu sprzedażowego skierowanego do osób bez doświadczenia informatycznego. Prototyp nazwa się Butiko co w języku Esperanto, stworzonego przez Ludwika Zamenhofa, oznacza sklep.

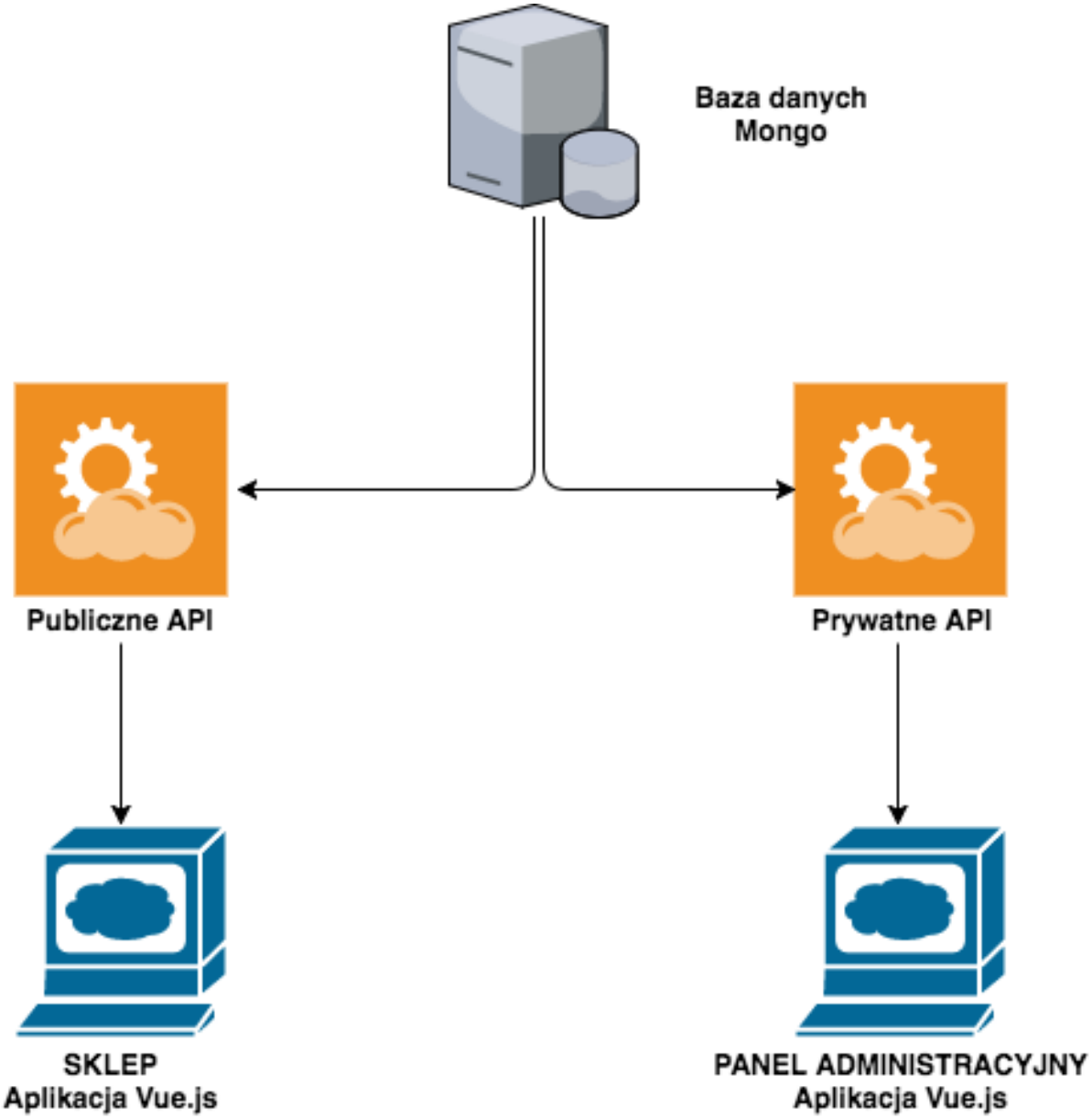
5.1 Architektura aplikacji

Aplikacja została zaprojektowana w oparciu o architekturę RESTful API. Architektura taka pozwala stworzyć system rozproszony. W architekturze prototypu można wyróżnić trzy warstwy (Rysunek 5.1):

- bazę danych
- serwis API
- aplikacja kliencka

Baza danych

Bazą danych wykorzystana w prototypie jest MongoDB. Jest to nierelacyjna i dokumentowa baza danych. Została ona wybrana, ponieważ bardzo szybko tworzy się do niej API, a komunikacja z bazą odbywa się przez pliki JSON (JavaScript Object Notation).



Rysunek 5.1: Architektura prototypu Butiko.

Serwis API

Serwis API jest pomostem między bazą danych, a aplikacją kliencką. API zostało napisane w Node.js. Komunikacja między aplikacją kliencką, a serwisem API odbywa się przez protokół HTTP. RESTful API wyróżnia się tym, że oprócz standardowych metod protokołu HTTP, czyli GET i POST wykorzystuje również inne metody m.in. PUT i DELETE. Dzięki temu, że aplikacja może być rozproszona istnieje możliwość stworzenia wielu serwisów API, które będą odpowiadały za różne aspekty aplikacji. W prototypie istnieją dwa serwisy API:

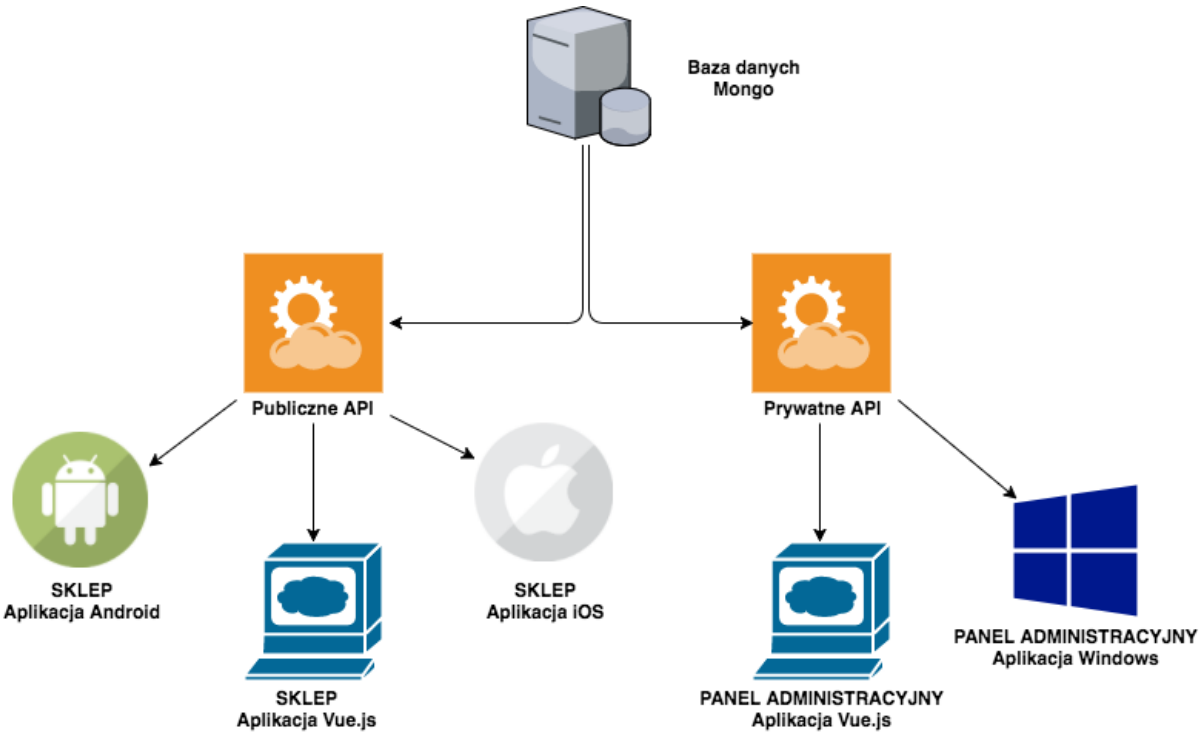
- Publiczny oferujący wyświetlanie produktów wraz ze stanem magazynowym oraz skła-

danie zamówień. Publiczne API może być wykorzystywane do aplikacji dla klientów sklepu.

- Prywatny oferujący możliwość edycji i tworzenia treści, obsługi zamówień. Wszystkie operacje wymagają dodania tokenu JWT Auth, który uwierzytelnia administratorów. API prywatne jest stworzone do komunikacji serwera z panelem administracyjnym.

Aplikacja kliencka

Jest to aplikacja, której odbiorcą jest człowiek, może być to administrator, albo klient sklepu. W architekturze RESTful API aplikacją kliencką może być aplikacja webowa, tak jak jest to w przypadku prototypu. W prototypie istnieją dwie aplikacje klienckie: sklep internetowy oraz panel administracyjny. Ponieważ architektura RESTful API pozwala na utworzenie aplikacji rozproszonej bardzo łatwo można dodać kolejne aplikacje klienckie jak np. aplikacja mobilna dla systemu iOS, aplikacja mobilna dla systemu Android, czy aplikacja desktopowa dla administratorów na system Windows (Rysunek 5.2).



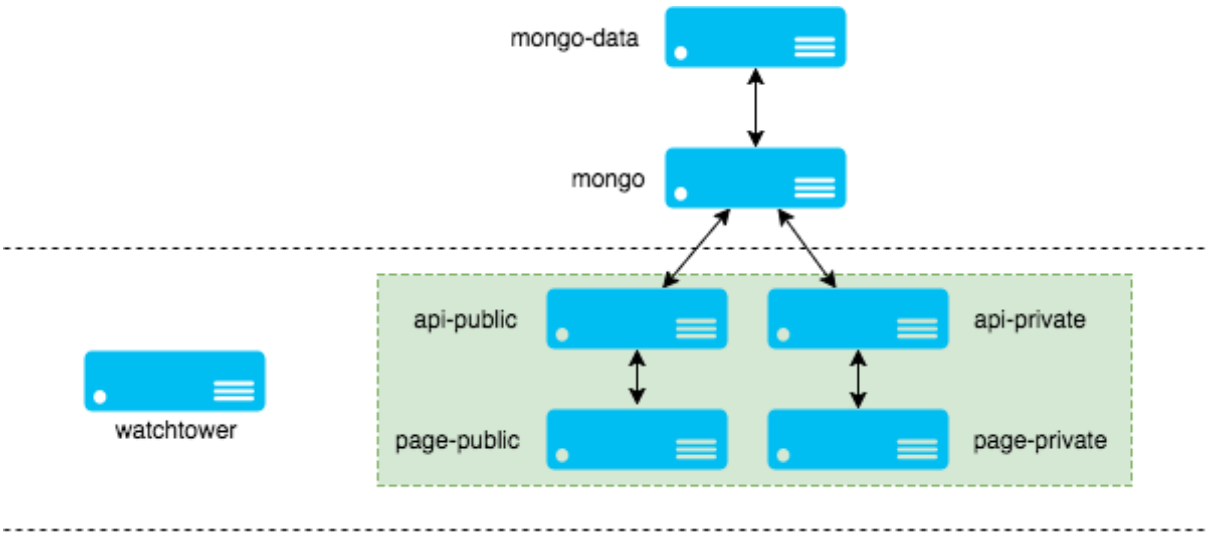
Rysunek 5.2: Możliwość rozbudowy w architekturze RESTful API.

5.2 Docker

Prototyp został przygotowany do uruchomienia go na plaftormę Docker. Aplikacja składa się z siedmiu kontenerów. Dodatkowo wykorzystano narzędzie Docker Compose, który zawiera całą konfigurację w jednym pliku.

5.2.1 Kontenery

Rysunek 5.3 przedstawia kontenery, które wchodzą w skład prototypu Butiko. Kontenery te można podzielić na prywatne i publiczne (na rysunku 5.3 publiczne kontenery znajdują się zielonym polu). W dalszej części podrozdziału opisane zostały kontenery składające się na aplikację.



Rysunek 5.3: Kontenery Dockera prototypu Butiko.

Kontener mongo-data

Jest to kontener w którym przechowywane są wszystkie dane bazy Mongo.

Kontener mongo

Kontener dla systemu bazy danych Mongo DB. Oddzielenie danych od samego systemu bazy danych zmniejsza ryzyko uszkodzenia danych przy aktualizacji kontenera. Poniżej zostały opisane wszystkie kontenery, które składają się na aplikację.

Kontener watchtower

Kontener, który obserwuje czy obrazy kontenerów publicznych są aktualne. Obrazy kontenerów przechowywane są w serwisie Docker Hub. Jeżeli pojawi się aktualizacja obrazu kontenera, kontener watchtower wykryję te zmianę, pobierze obraz kontenera, skompiluje go i uruchomi, a następnie podmieni istniejący kontener z nowo utworzonym.

Kontenery api-public oraz api-private

Kontenery, które przechowują serwisy API. Kontenery są zbudowane z oficjalnych obrazów Node.js. Kontenery te oparte są na Linuksie Apline, którego cechą jest niewielkie zapotrzebowanie na przestrzeń dyskową.

Kontenery page-public oraz page-private

Kontenery przechowują aplikacje klienckie. Kontener page-public przechowuje sklep internetowy, a page-private - panel administracyjny.

Kontenery te oparte są na tym samym obrazie, który został utworzony na potrzeby tej pracy. Listing 5.1 przedstawia plik Dockerfile, który jest plikiem tworzącym obraz kontenera. Obraz ten oparty jest na nginx, który pełni rolę serwera reverse proxy.

Aplikacje klienta zostały stworzone we Vue.js. Aby uruchomić aplikację w trybie produkcyjnym wymagany jest npm (node package manager). Kontener więc najpierw pobiera npm, następnie instaluje wszystkie biblioteki wykorzysywane w aplikacji, a następnie uruchamiany jest skrypt build. Efektem końcowym skryptu build jest folder dist, który zawiera plik index.html oraz skompresowane pliki JavaScript.

Folder dist przenoszony jest do folderu na który serwer proxy przekierowuje połączenie z portu 80.

Listing 5.1: Plik Dockerfile kontenerów page-public i page-private

```
FROM nginx:alpine
WORKDIR /app

RUN apk --no-cache add --virtual nodejs nodejs-npm

COPY package.json /app
RUN npm install
COPY . /app

RUN npm run build
```

```
RUN  rm -rf /usr/share/nginx/html
     && mv /app/dist/ /usr/share/nginx/html/

EXPOSE 80
```

5.2.2 Docker Compose

Docker Compose to narzędzie, które pozawala na skonfigurowanie aplikacji, która korzysta z wielu kontenerów. Konfiguracja ta może sprowadzać się nawet do jednego pliku YML. Tak też jest w przypadku prototypu Butiko. Listing 5.2 przedstawia plik docker-compose.yml, który uruchamia wszystkie wyżej opisane kontenery. Tak więc, aby zainstalować aplikację Butiko na serwerze, który posiada platformę Docker wystarczy pobranie jednego pliku konfiguracyjnego i uruchomiene komendy 'docker-compose up'. Nie potrzebna jest żadna dodatkowa konfiguracja.

Listing 5.2: Plik docker-compose.yml prototypu Butiko

```
version: "2"
services:
  api-admin:
    container_name: api-admin
    image: wk93/butiko-api-admin
    ports:
      - "3000:3000"
    links:
      - mongo
  api-public:
    container_name: api-public
    image: wk93/butiko-api-public
    ports:
      - "4000:4000"
    links:
      - mongo
  page-admin:
    container_name: page-admin
    image: wk93/butiko-page-admin
    ports:
      - "8080:80"
    links:
      - api-admin
  page-public:
    container_name: page-public
    image: wk93/butiko-page-public
    ports:
      - "80:80"
    links:
      - api-public
  mongo:
    image: mongo
    ports:
      - "27017:27017"
    volumes_from:
```

```
    - mongodata
mongodata:
  image: tianon/true
  volumes:
    - /data/mongo
watchtower:
  image: v2tec/watchtower
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  command: api-admin api-public
           page-admin page-public --cleanup
  restart: always
```

5.3 Aplikacja serwerowa

Serwisy API wspomniane w części na temat architektury aplikacji to część serwerowa prototypu. W prototypie, jak pokazano na Rysunku 5.1, istnieją dwa serwisy API - publiczny i prywatny. Serwisy API zostały napisane w języku JavaScript i działają w środowisku Node.js.

5.3.1 Publiczne API

Podstawowym zadaniem API jest obsługa żądań, które pochodzą z aplikacji klienckiej i wysyłanie odpowiedzi. Wszystkie dostępne akcje wykonywane na żądania użytkownika wykonywane są na bazie danych. Można więc wyróżnić trzy elementy serwisu API:

- schemat,
- kontroler,
- router.

Schemat

Schemat jest to opis struktury dokumentu. Pełni tę samą rolę co Model w architekturze MVC, czyli jest reprezentacją problemu bądź logiki biznesowej.

Schemat jest ważnym elementem w API. Baza danych Mongo daję możliwość utworzenia dowolnej struktury dokumentów w ramach jednej kolekcji. Jest to bardzo przydatna rzecz, ale należy tę wolność ograniczyć, tak aby dane przechowywane w kolekcji miały jakąkolwiek wartość i mogły być przetwarzane przez aplikację. Właśnie taką rolę pełni schemat. W tym

obiekcie opisywane są pola jakie powinny występować w dokumencie, jakiego powinny być typu. Określane też są walidatory.

Listing 5.3 zawiera kluczowy schemat w prototypie Butiko - schemat zamówień. Schemat ten zawiera informację, że w dokumencie w kolekcji Order powinny mieć pola

- address - wymagane pole typu AddressSchema, czyli innego schematu. W bazie w tym miejscu bedzie znajdował się obiekt, który jest opisany w schemacie AddressSchema. Nie jest to referencja do dokumentu z innej kolekcji, jak jest to w bazie danych.
- email - wymagane pole typu String, czyli ciąg znaków
- phone - wymagane pole typu String
- comment - niewymagane pole typu String
- delivery - wymagane pole typu String
- payment - wymagane pole typu String
- items - tablica obiektów. Obiekty w tej tablicy mają pola:
  - product - wymagane pole tybu ObjectID. Jest to referencja do dokumentu z kolekcji Products
  - count - wymagane pole typu Number. Typem Number może być każda liczba rzeczywista
  - price - wymagane pole typ Number
- statuses - tablica obiektów, które mają pola:
  - type - wymagane pole typu String,
  - date - wymagane pole typu Date. Typ Date w bazie Mongo jest to data zapisana w standardzie ISO.

Listing 5.3: Schemat zamówień w serwisie API

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema
var AddressSchema = require('./address')

var OrderSchema = new Schema({
  address: {
    type: [AddressSchema],
    required: [true, 'Address is required']
  },
  email: {
    type: String,
    required: [true, 'Email is required']
```

```
  },
  phone: {
    type: String,
    required: [true, 'Phone is required']
  },
  comment: {
    type: String,
    required: false
  },
  delivery: {
    type: String,
    required: [true, 'Delivery is required']
  },
  payment: {
    type: String,
    required: [true, 'Payment is required']
  },
  items: [
    {
      product: {
        type: Schema.Types.ObjectId,
        ref: 'Product',
        required: true
      },
      count: {
        type: Number,
        required: true
      },
      price: {
        type: Number,
        required: true
      }
    }
  ],
  statuses: [
    {
      type: {
        type: String,
        required: true
      },
      date: {
        type: Date,
        required: true
      }
    }
  ]
})

module.exports = mongoose.model('Order', OrderSchema)
```

Tak utworzony obiekt zawiera wsztystkie informacje o zamówieniu, które złożył użytkownik. Przykładowy dokument z bazy danych przedstawiony jest na Listingu 5.4

Listing 5.4: Dokument przechowywany w Mongo

```
{
  "_id" : ObjectId("5a214faae890d30373bdfc48"),
  "email" : "s10451@pjwstk.edu.pl",
  "phone" : "818 642 221",
  "delivery" : "kurier",
  "payment" : "przelew",
  "statuses" : [
    {
      "type" : "New",
      "date" : ISODate("2017-12-01T12:48:42.802Z"),
      "_id" : ObjectId("5a214faae890d30373bdfc49")
    },
    {
      "date" : ISODate("2017-12-01T12:49:07.442Z"),
      "type" : "Completed",
      "_id" : ObjectId("5a214fc3de96cb17d17c9a88")
    }
  ],
  "items" : [
    {
      "product" : ObjectId("5a0ef29699e7e1fc8b262a77"),
      "price" : 10,
      "count" : 1,
      "_id" : ObjectId("5a214faae890d30373bdfc4a")
    }
  ],
  "address" : [
    {
      "name" : "Wojciech",
      "surname" : "Kania",
      "street" : "Klonowa",
      "zipcode" : "00-105",
      "city" : "Warszawa",
      "_id" : ObjectId("5a214faae890d30373bdfc4b"),
      "number" : [
        "10",
        "12"
      ]
    }
  ],
  "__v" : 0
}
```

Kontroler

Kontroler jest to obiekt, który posiada zbiór funkcji, które działają na danym schemacie. W Listingu 5.5 przedstawiono jedną funkcji z kontrolera, który działa na schemacie Product. Funkcja readOne otrzymuje trzy parametry:

- req - obiekt w którym znajdują się m.in. parametry przesłane w żądaniu użytkownika,

- res - obiekt który posiada m.in. funkcję odopwiadającą za wysłanie odpowiedzi do użytkownika
- next - funkcja, która przechodzi do następnej funkcji stworzonej w routerze.

W funkcji tej pobieramy z obiektu req parametr id, który powinien podać użytkownik, a następnie wywołujemy zapytanie do bazy danych, czy istnieje taki produkt. W zapytaniu tym podajemy funkcję, która ma się wykonać po wykonaniu zapytania. Funkcja ta albo wywoła funkcję, która wysyła odpowiedź do użytkownika z produkt, jeżeli będzie produkt w bazie o takim id, albo z błędem.

Listing 5.5: Funkcja readOne z kontrolera ProductController

```
var ProductController = {
  readOne(req, res, next) {
    const id = req.params.id
    ProductSchema.findById({_id: id},
    function (error, product) {
      if (error) {
        res.status(400).send({error: error.message})
      } else {
        res.send({product})
      }
    })
  }
}
```

Funkcje w Node.js tworzone są w technice callback. Ta technika programowania polega na tworzeniu funkcji, które zostaną wywołane po wykonaniu funkcji, które pochodzą z zewnętrznych bibliotek. Tworząc zapytanie do bazy danych, czyli odwołując się do funkcji findById potrzebna jest wiedza o tym jak przekazać parametr do funkcji i co funkcja ma zwrócić. Nie trzeba wiedzieć co dokładnie ta funkcja robi. Z taką wiedzą możemy stworzyć funkcję callback, która przyjmie dwa parametry error i product.

Również sama funkcja readOne jest funkcją callback. Otrzymuje ona parametry z routera, który w dalszej części pracy zostanie opisany, a następnie wywołuje funkcje, które zostały jej wcześniej podane.

Router

Router przyporządkowuje metody oraz URL żądania HTTP do funkcji kontrolerów. Listing 5.6 przedstawia fragment funkcji pełniącą rolę routera. Funkcja przyjmuję jako parametr obiekt app. Jest to obiekt express pochodzący z biblioteki o takiej samej nazwie. Obiekt ten odbiera wszystkie żądania HTTP. Aby wywołać daną funkcję na żądanie HTTP należy wywołać funkcję obiektu app o nazwie odpowiadającej metodzie żądania (GET, POST, PUT, DELETE), a

jako parametry podać URL oraz referencje do odpowiedniej funkcji. Przykładowo w funkcji z Listingu 5.6 wywołanie funkcji app.get('/products/:id', ProductController.readOne), powoduje przypisanie do żądania HTTP typu GET o URL '/products/:id' (gdzie id to parametr żądania) funkcji readOne pochądzącej z kontrolera ProductController.

Listing 5.6: Fragment routera publicznego API

```
var router = function (app) {
  app.get('/products/:id', ProductController.readOne)
  app.get('/products/', ProductController.readAll)

  app.post('/orders/', OrderController.create)
}

module.exports = router
```

5.3.2 Prywatne API

Prywatne API ma taką samą budowę jak publiczne API. API prywatne różni się od publicznego tym, że każde żądanie musi posiadać token. Token ten użytkownik może otrzymać podając:

- e-mail - pole unikalne,
- hasło - w bazie danych przechowywane po wcześniejszym zaszyfrowaniu algorytmem bcrypt.

Listing 5.7 przedstawia funkcję login, która sprawdza czy użytkownik wprowadził poprawne dane, jeżeli tak zwrócony zostanie token ważny 24 godziny. W przeciwnym wypadku zostanie zwrócony odpowiedni błąd.

Listing 5.7: Funkcja logująca w prototypie Butiko

```
login (req, res, next) {
  var credentials = {
    email: req.body.email,
    password: req.body.password
  }

  if (!credentials.email || !credentials.password) {
    return res.status(401).send(
      {error: 'Email and password is required'}
    )
  }

  console.log(credentials.error)

  UserSchema.findOne({email: credentials.email},
  (error, user) => {
```

```
    if (error) return next(error)

    if (user) {
      if (user.comparePassword(credentials.password)) {
        const payload =
          {id: user._id, email: user.email}

        const token = jwt.sign(payload, this.getKey(),
        {expiresIn: '24 hours'})

        return res.send({token})
      }

      return res.status(401).json(
        {error: 'Wrong password'})
    }

    return res.status(401).json(
      {error: 'User not found'})
  })
}
```

Listining 5.8 przedstawia funkcję verify. Funkcja ta sprawdza, czy token który wprowadził użytkownik jest poprawny. Funkcja ta jest funkcją middleware (funkcją pośredniczącą). Wykonywana jest przed każdym żądaniem (z wyjątkiem funkcji logującej) HTTP. Jeżeli token nie jest poprawny kolejne funkcje nie zostaną wykonane. Takie zachowanie routera jest możliwe dzięki jego strukturze, która została przedstawiona w listingu 5.9

Listing 5.8: Funkcja sprawdzająca poprawność tokenu w prototypie Butiko

```
verify (req, res, next) {
    var token = req.headers.authorization
    if (token) {
      token = token.split(' ')[1]
      jwt.verify(token, this.getKey(),
      function (error) {
        if (error) {
          return res.status(401)
            .json({error: 'Token is not valid'})
        }

        return next()
      })
    } else {
      return res.status(403)
      .send({error: 'No token provided.'});
    }
  }
```

Listing 5.9: Początek funkcji router w prywatnym API

```
var router = function (app) {
  app.post('/login', JwtController.login)
```

```
app.all('*', JwtController.verify)

...
```

5.4 Aplikacja kliencka

Aplikacja kliencka składa się z dwóch aplikacji webowych - sklepu internetowego oraz panelu administracyjnego.

5.4.1 Sklep internetowy

Sklep internetowy został napisany we frameworku Vue.js. Jest to Single Page Application, czyli aplikacja, która ma tylko jedną stronę, jest wczytywana tylko raz. Posiada ona jednak różne widoki. W aplikacji został wykorzystany router, który przypisuje URL do odpowiedniego komponentu. W wyniku tego komponenty są wczytywane dynamicznie, odświeżana jest tylko część strony, web component <router-view>. Aplikacja składa się z następujących widoków:

- widok katalogu produktów,
- widok koszyka,
- widok formularza zamówienia,
- widok stron statycznych.

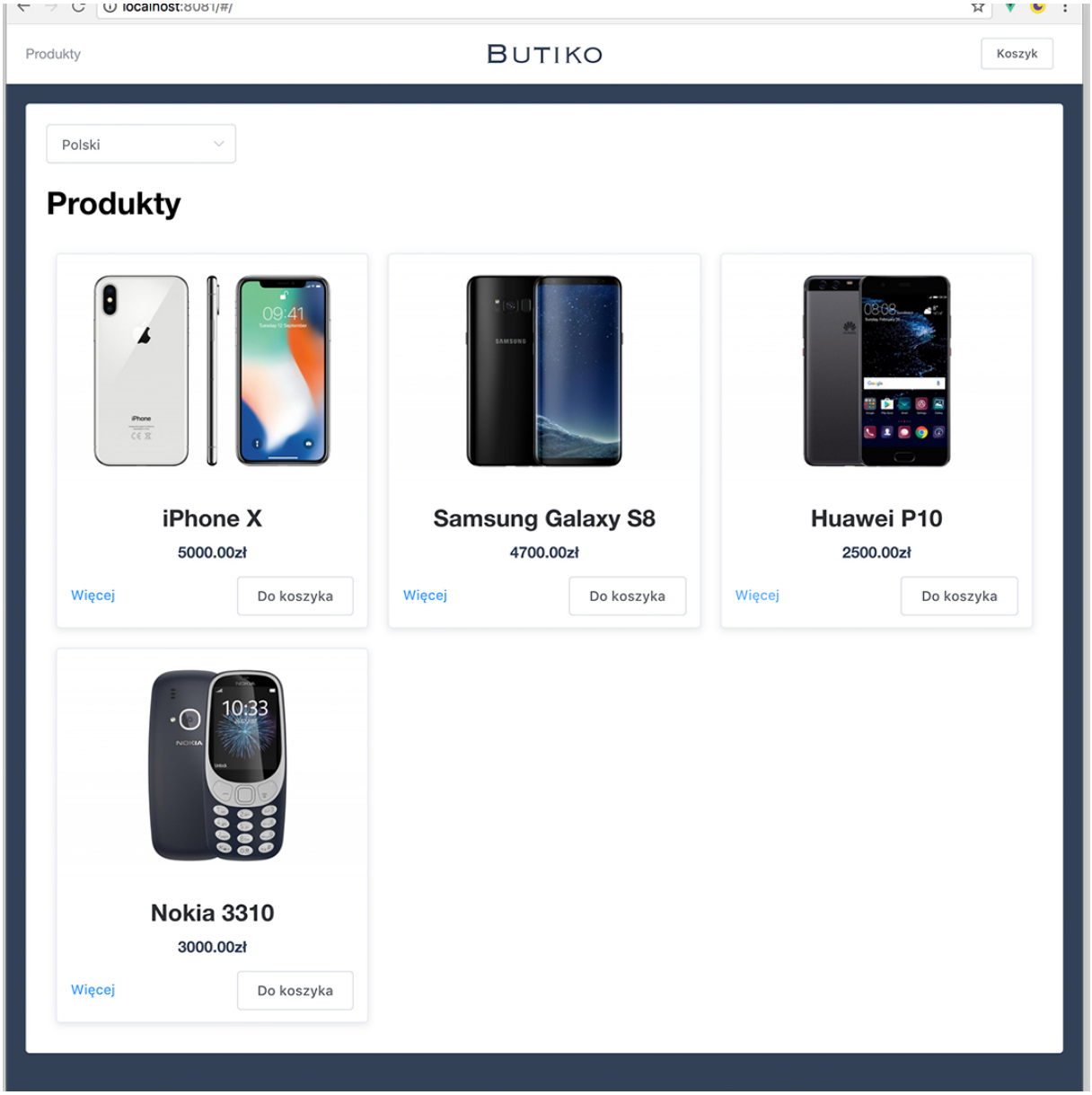
Poniżej zostaną pokrótce opisane wymienione widoki.

Widok katalogu produktów

Widok odpowiedzialny za wyświetlenie wielu produktów. Widok ten może wyświetlić wszystkie produkty, które są dostępne w sklepie, albo produkty, które posiadają określony tag. Rysunek 5.4 przedstawia widok wszystkich produktów dostępnych w przykładowym sklepie.

Widok koszyka

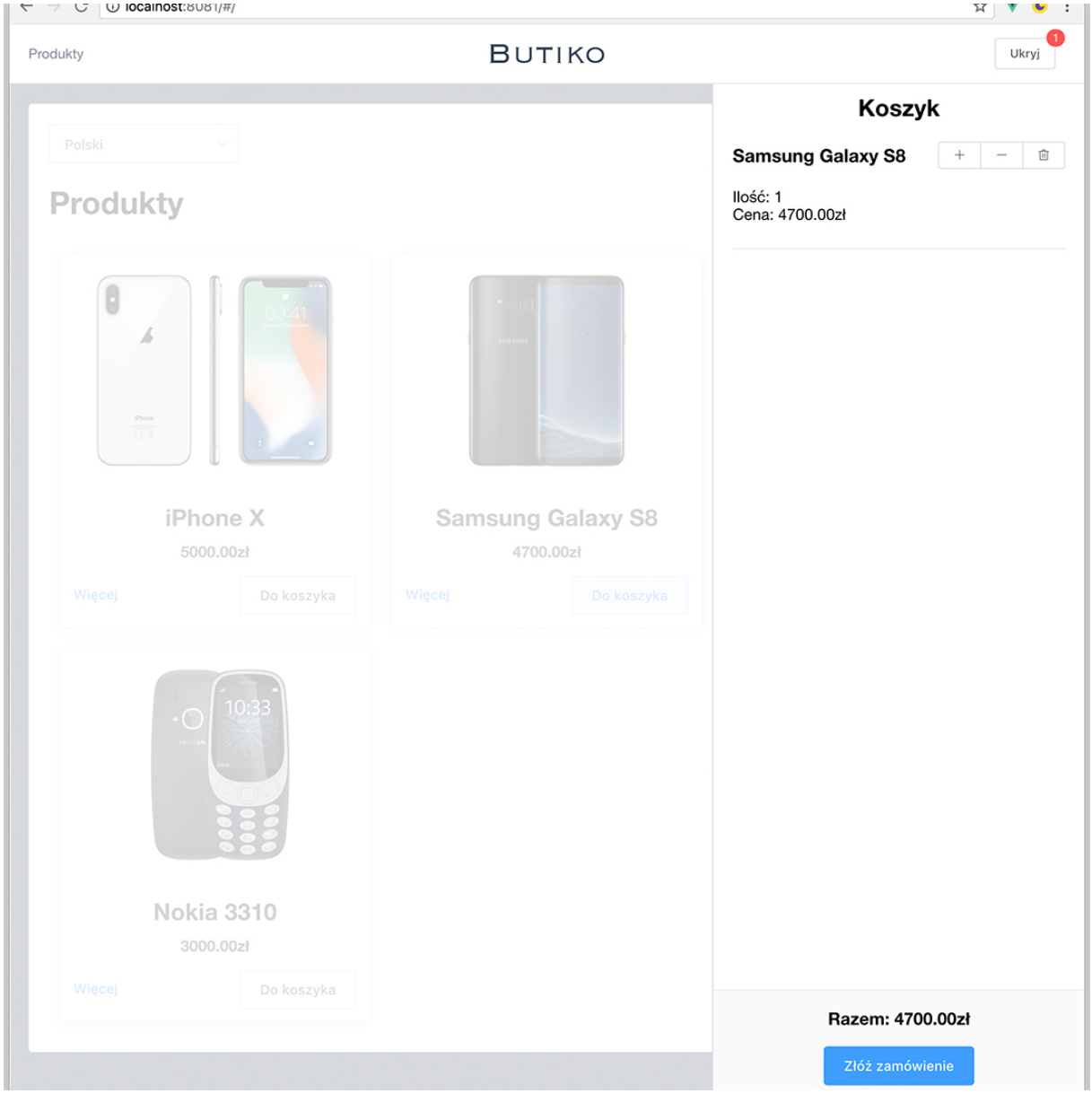
Jest to widok, który różni się od pozostałych tym, że nie jest wyświetlany przez component <router-view>. Komponent koszyka jest widgetem, który jest dostępny z każdego widoku (oprócz widoku formularza zamówienia) podobnie jak nagłówek. Widok ten prezentuje aktualny stan koszyka oraz zawiera odnośnik do przejścia do formularza zamówienia. Rysunek 5.5 przedstawia widok strony, kiedy widget jest aktywny.



Rysunek 5.4: Butiko: Widok katalogu

Widok formularza zamówienia

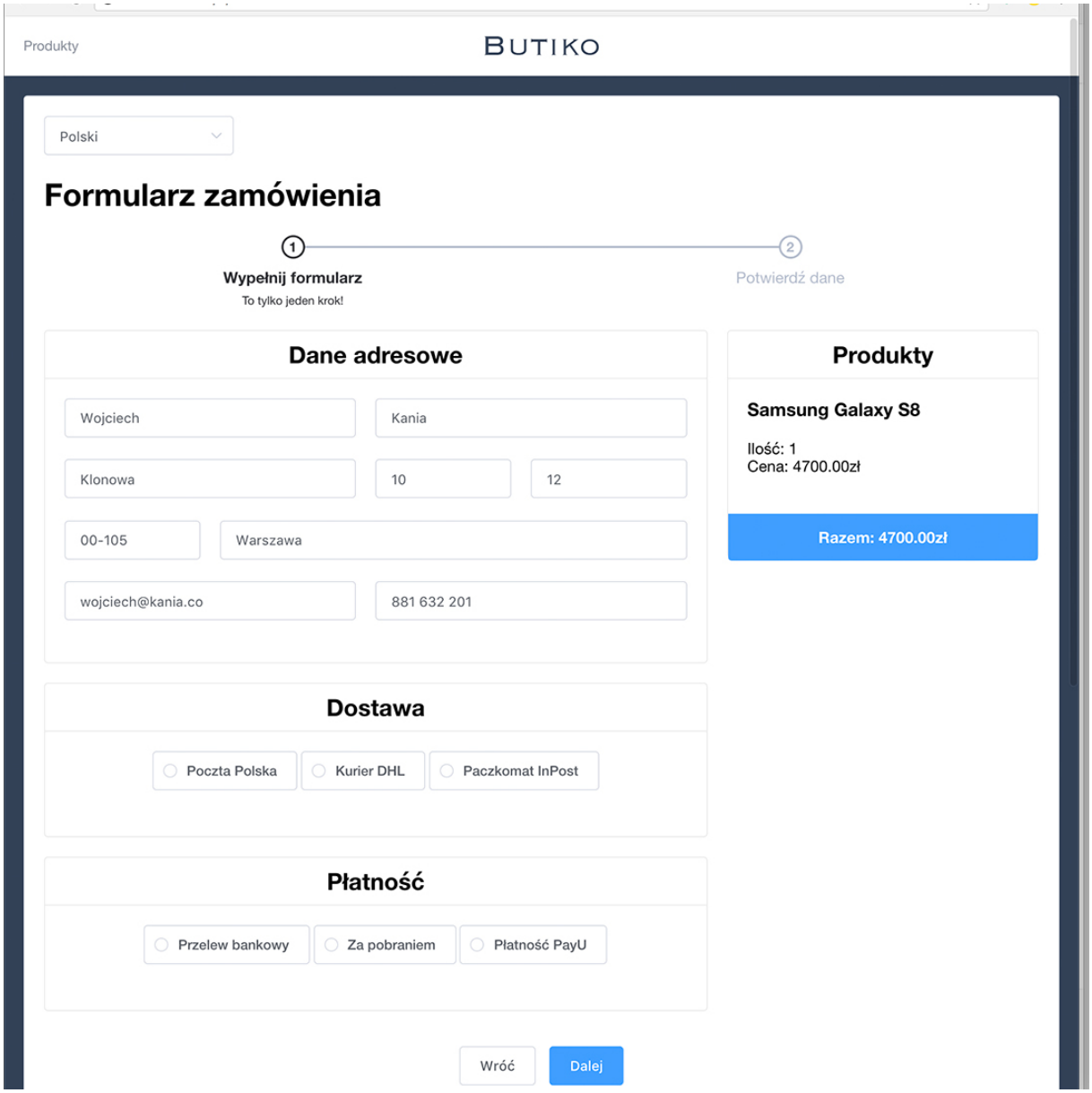
Widok ten odpowiada za wyświetlenie formularza z polami niezbędnymi do realizacji zamówienia. Użytkownik w tym miejscu musi wprowadzić adres dostawy, wybrać formę dostawy oraz płatności. Poprawność danych wprowadzonych w pola w formularzu są sprawdzane już podczas ich wpisywania przez użytkownika. Oprócz formularza użytkownik ma podgląd do produktów jakie chce zamówić. Przycisk składający zamówienie posiada formułę 'Zamawiam i płacę', która jest wymagana przez polskie prawo. Rysunek 5.6 przedstawia widok formularza zamówienia.



Rysunek 5.5: Butiko: Widok koszyka

Widok strony statycznej

Widok strony statycznej odpowiada za wyświetlenie treści takich jak: regulamin sklepu, dane kontaktowe, informacje o dostawie itd. Strona statyczna jest zbiorem modułów. Moduły te mogą mieć różną formę - tekstu z nagłówkiem, galerii zdjęć, slidera z banerami, czy formularza. Dzięki parametrowi v-if, komponenty te mogą być zadeklarowane w kodzie, ale dodane do drzewa DOM zostaną tylko te, które spełniają warunek, będący wartością parametru v-if. Dzięki temu można zadeklarować wiele różnych komponentów i nie wpłynie to na szybkość działania strony. Moduły strony statycznej przechowywane są w tablicy. W widoku strony statycznej znajduje się pętla v-for, która dodaje do drzewa DOM w każdej iteracji komponent

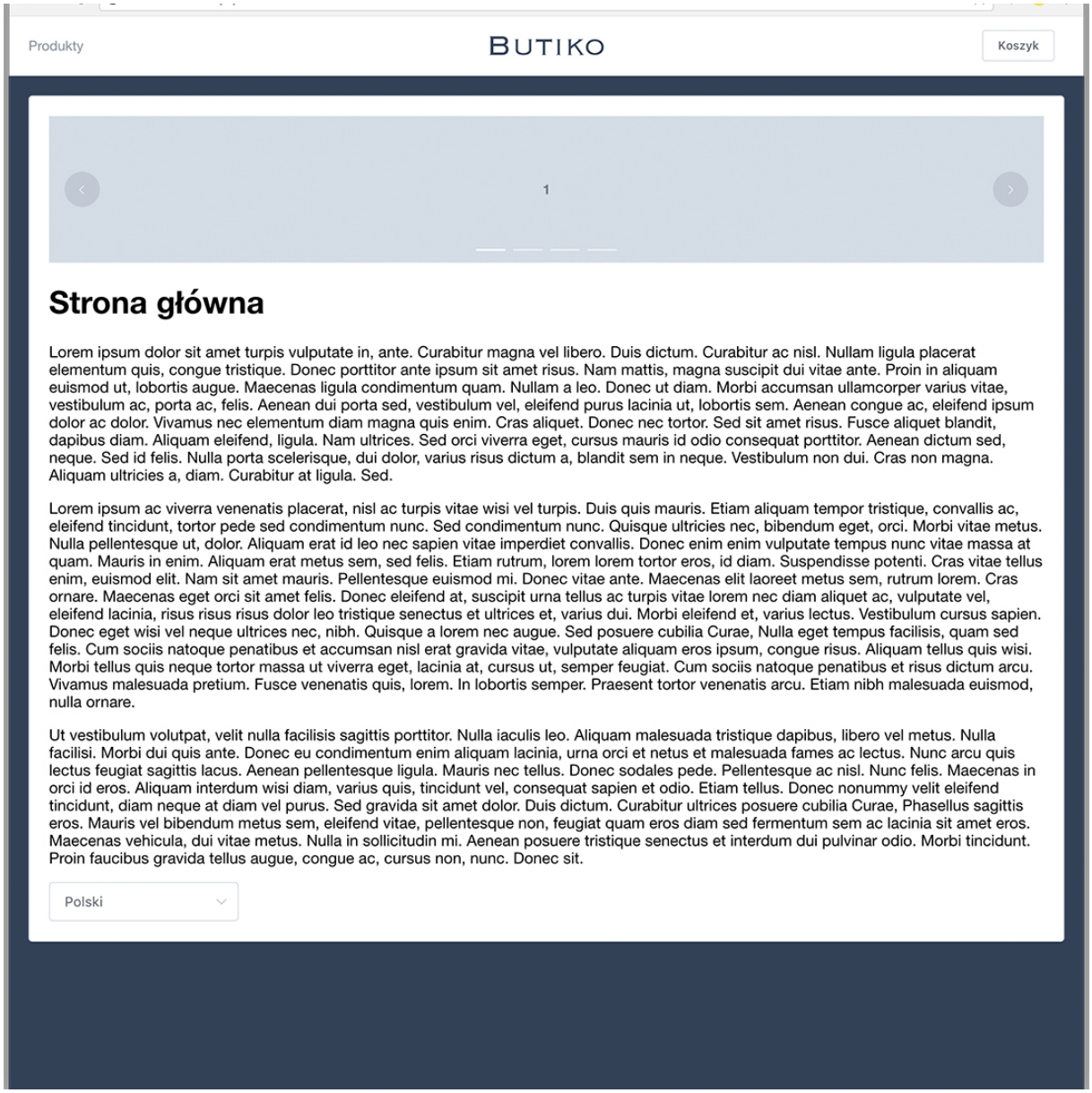


Rysunek 5.6: Butiko: Widok formularza zamówienia

przypisany do danego modułu. Rysunek 5.7 przedstawia widok przykładowej strony z dwoma modułami: slidera oraz tekstu z nagłówkiem.

5.5 Panel Administracyjny

Panel Administracyjny to również aplikacja stworzona we frameworku Vue.js. Działa identycznie jak wcześniej opisana aplikacja sklepu internetowego. W Panelu Administracyjnym możemy wyróżnić następujące widoki:

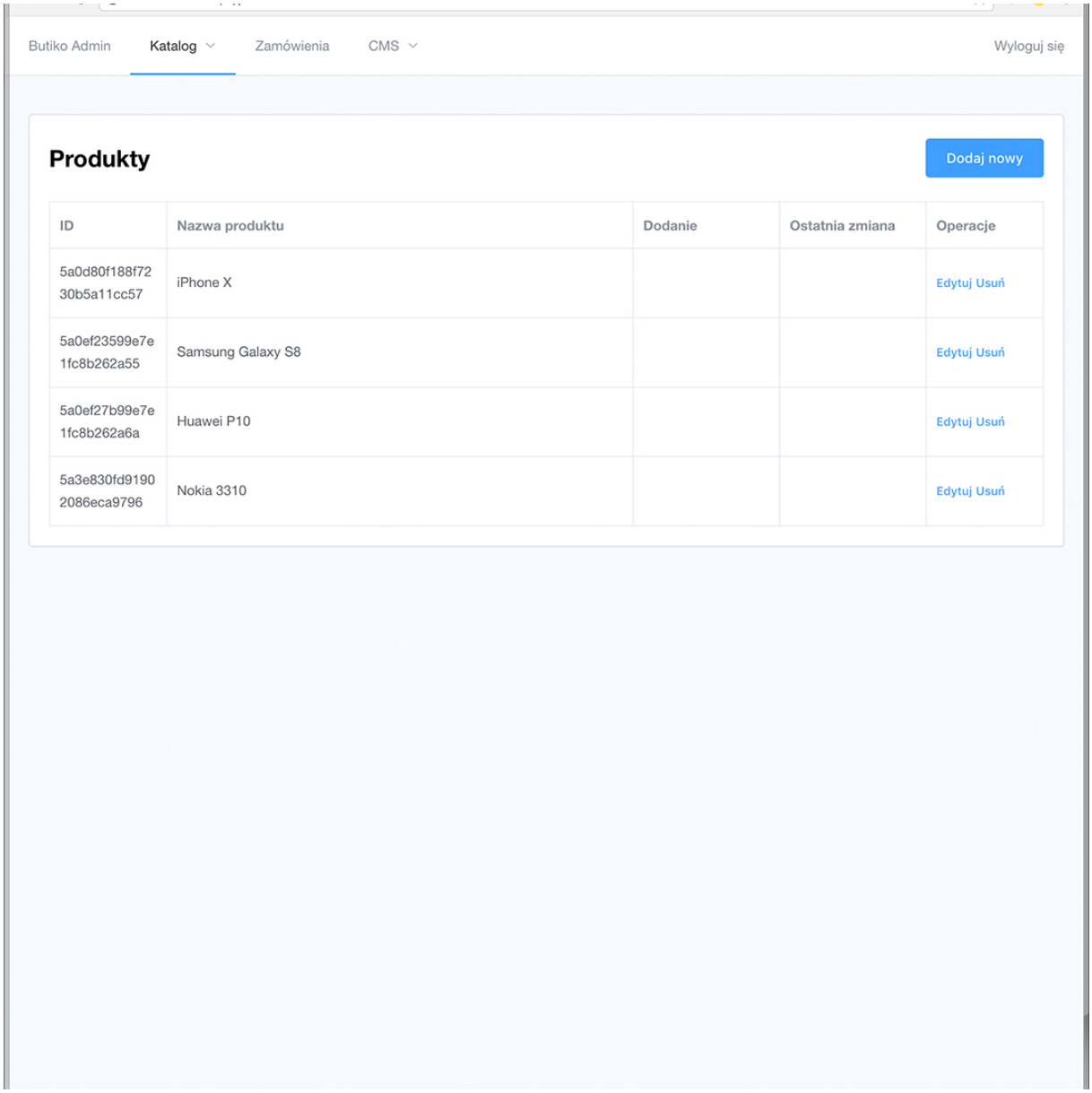


Rysunek 5.7: Butiko: Widok strony statycznej

- widok listy,
- widok formularza,
- widok listy z formularzem,
- widok formularza z zakładkami.

Widok listy

Widok listy jest tabelą w która prezentuje różne zestawy obiektów: produkty, zamówienia, tagi, moduły, strony statyczne. W tabeli znajdują się również odnośniki do formularza tworzący nowy obiekt, albo edytujący wybrany obiekt. Rysunek 5.8 przedstawia widok przykładowej tabeli.



Rysunek 5.8: Butiko: Widok listy produktów

Widok formularza

Widok ten zawiera formularz dzięki któremu można dodawać nowe lub edytować obecne obiekty. W zależności od typów obiektu formularz ten może mieć różne pola.

W przeciwieństwie do systemów opisanych w tej pracy, formularz ten wyróżnia walidacja już podczas wpisywania wartości w pola przez użytkownika.

Alternatywne systemy miały problem ze zdjęciami. Magento obsługiwało wgrywanie zdjęć przez Flasha, a osCommerce zawierał standardowy formularz dodania dowolnego pliku. Zastosowane rozwiązania sprawiają, że zdjęcia wgrywane są w tle, a kiedy zostaną pomyślnie wgrane drzewo DOM zostanie zaktualizowane i użytkownik zobaczy je od razu. Animacja pojawienia się zdjęć w produkcie sprawia, że użytkownik ma wrażenie wgrywania zdjęć w czasie rzeczywistym.

Formularz wyróżnia się na tle konkurencji również tym, że jego struktura jest dynamiczna. Pozwala to na dodawanie kolejnych pól do formularza, a nawet tworzenie skomplikowanych konstrukcji. Przypadek użycia pokaże, że dzięki dynamicznej strukturze formularza zarządzanie cechami produktów i ich zależnościami jest proste i intuicyjne w porównaniu do rozwiązań konkurencyjnych systemów.

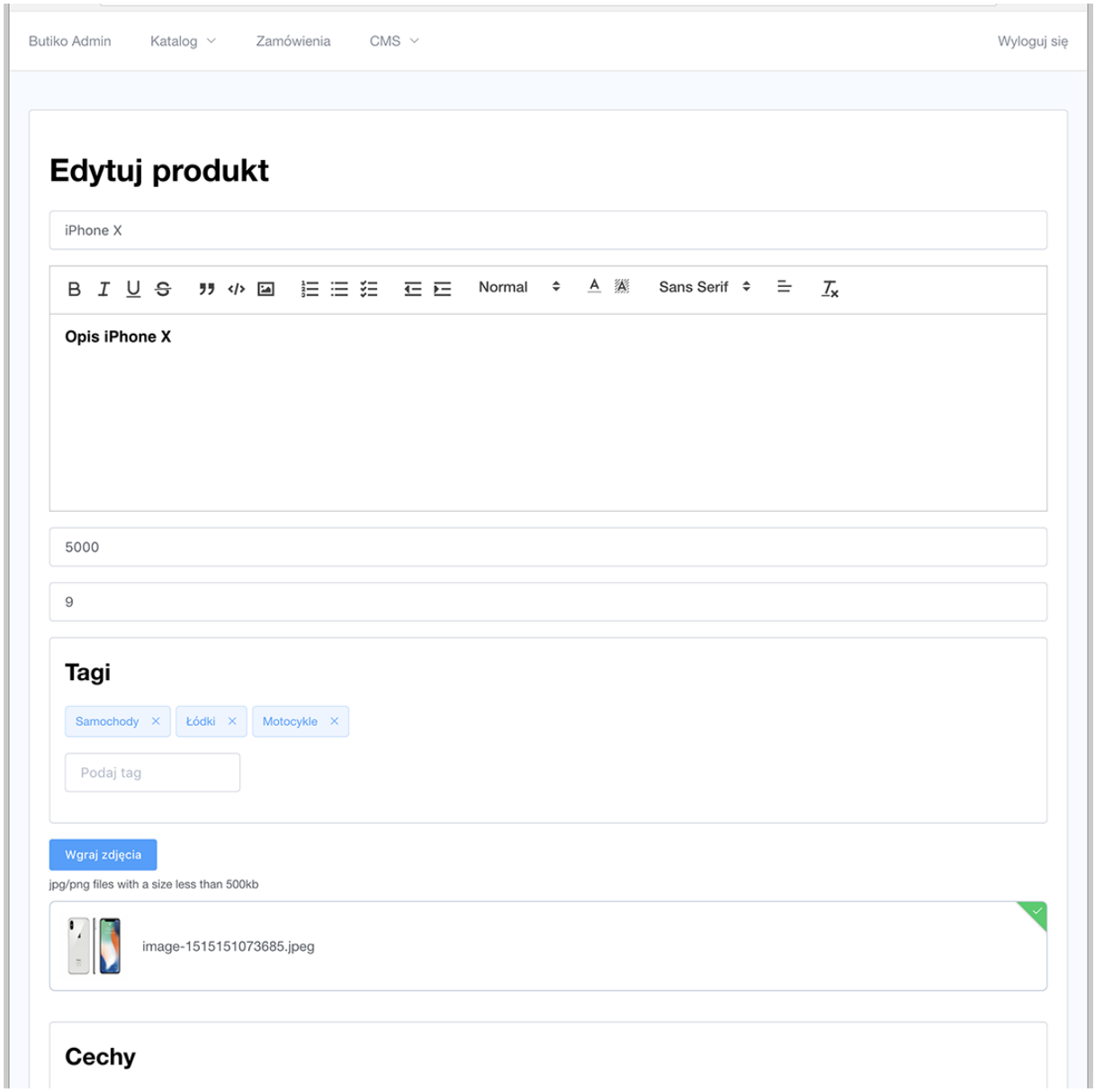
Formularz tworzenia produktu posiada miejsce na tytuł, opis, cenę oraz ilość produktów. Do produktu można dodać również tagi. Użytkownik wprowadzając nazwę tagu, który chce dodać otrzymuje podpowiedzi - listę tagów, których tytuł posiada wpisany przez użytkownika tekst. Użytkownik może dodawać zdjęcia, ma od razu ich podgląd i może je również usuwać. Ostatnią rzeczą to cechy, które użytkownik może dodać do danego produktu. Istnieje możliwość wprowadzenia określonej ilości produktu występującego w danej konfiguracji. Rysunek 5.9 przedstawia fragment formularza produktu.

Formularz tworzenia strony posiada miejsce na tytuł strony, a także listę modułów, które znajdują się na stronie. Moduły można usuwać i zmieniać ich kolejność. Można również dodawać nowe moduły wybierając je z listy. Istnieje możliwość dodania wielokrotnie tego samego modułu. Rysunek 5.10 przedstawia formularz edycji strony głównej.

Formularz modułu posiada listę rozwijaną z której użytkownik może wybrać typ modułu. Zmieniając rodzaj modułu pola formularza dynamicznie dostosują się do wybranej struktury. Rysunek 5.11 przedstawia formularz tworzenia nowego modułu typu tekst z nagłówkiem.

Widok listy z formularzem

Widok listy z formularzem wykorzystywany jest, kiedy pól formularzu jest bardzo mało. Przykładem może być formularz tworzący nowy tag, który zawiera tylko jedno pole (Rysunek 5.14.

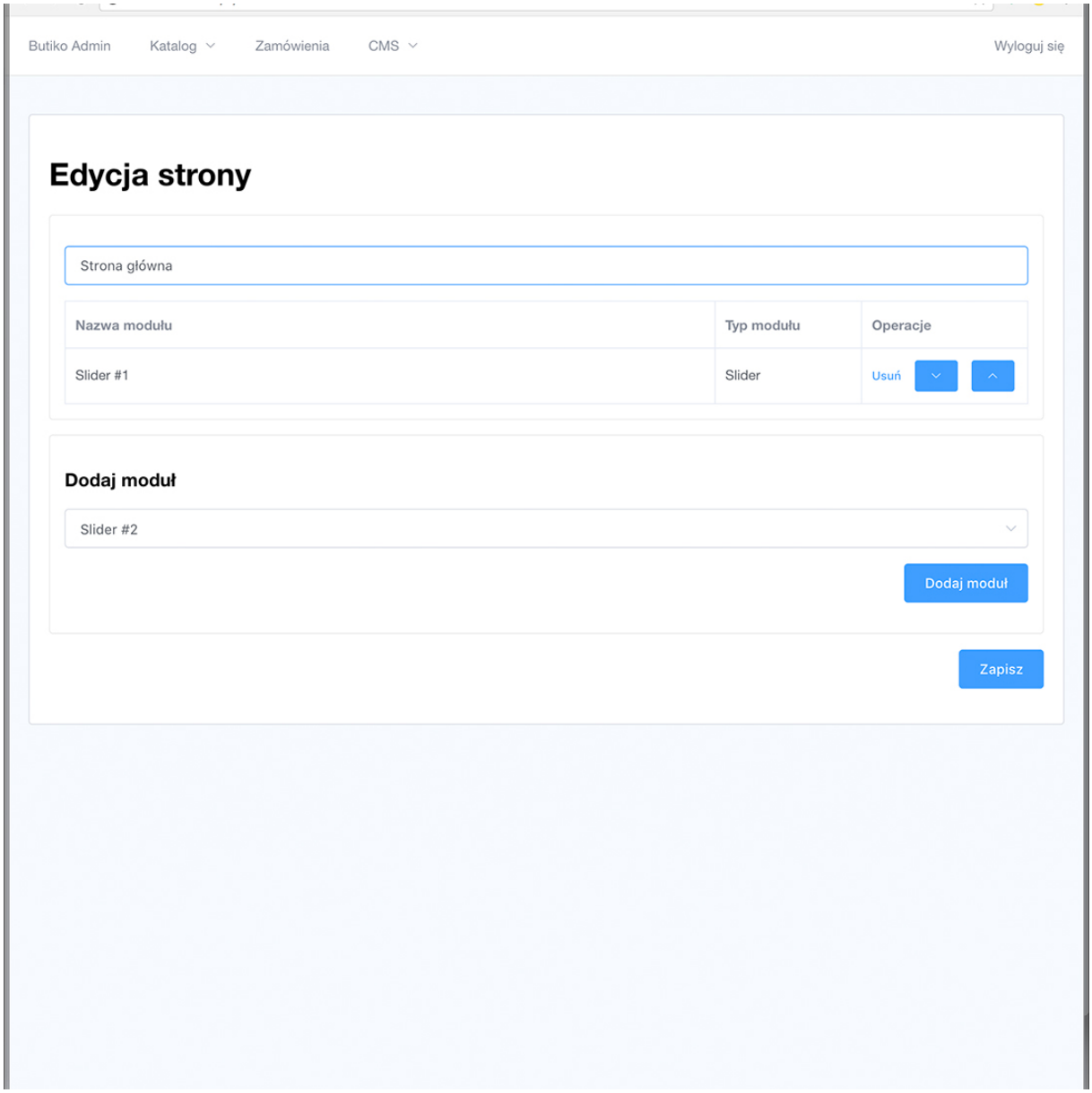


Rysunek 5.9: Butiko: Widok formularza produktu

Użycie widoku listy z formularzem jest wykorzystywany kiedy obiekty posiadają taką strukturę, że dla użytkownika intuicyjny będzie podgląd obecnej struktury z formularzem tworzącym elementy. Przykładem takich obiektów mogą być linki tworzone w nawigacji (Rysunek 5.13).

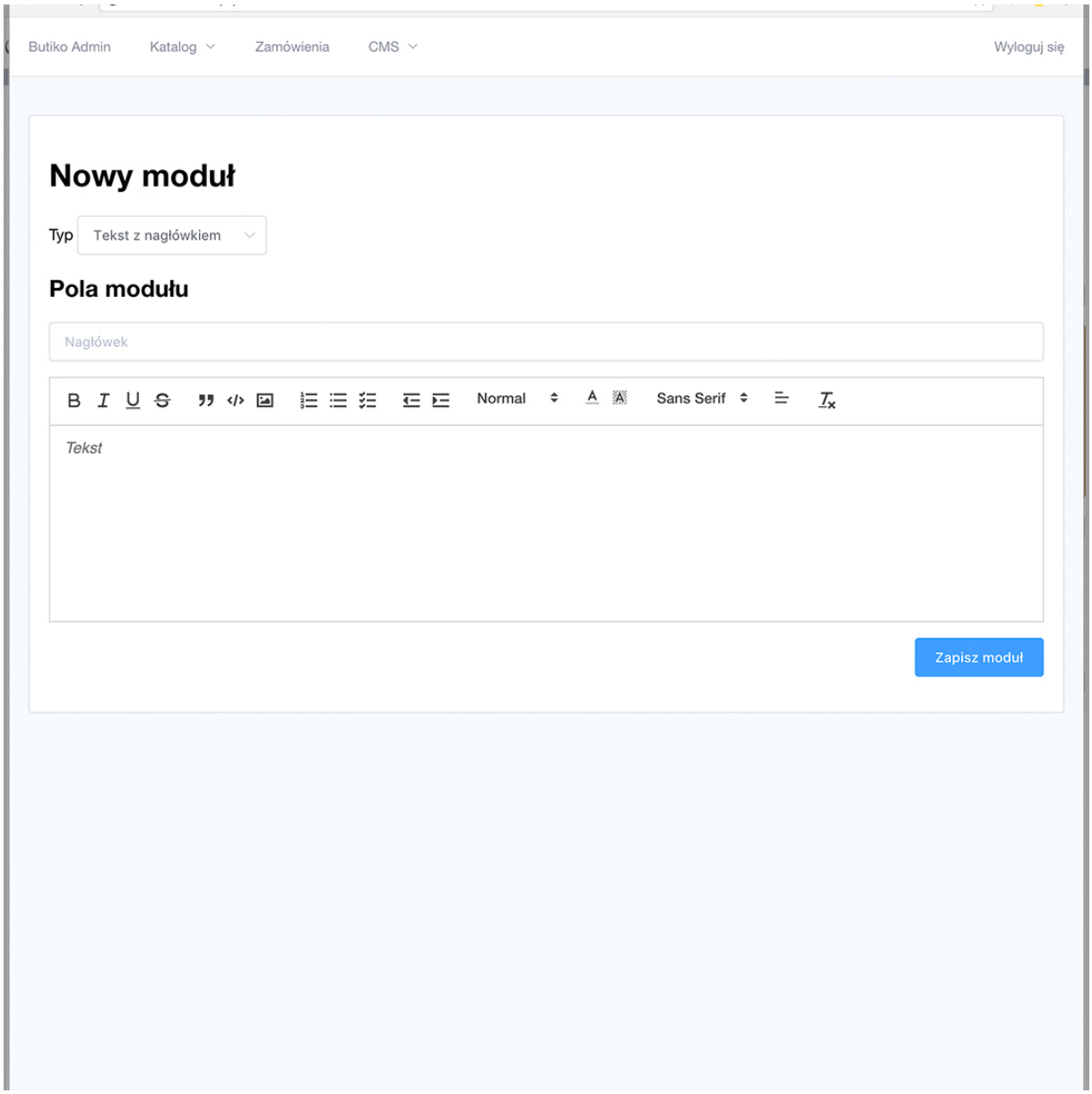
Widok formularza z zakładkami

Widok ten jest wykorzystywany, kiedy istnieje bardzo dużo pól w formularzu. W tym widoku pola formularza grupowane są w sekcje, które są wyświetlane w formie zakładek i/lub kart Przykładem takiego formularza może być formularz z tłumaczeniami (Rysunek 5.14). Pola tego formularza zostały podzielone na języki i wyświetlone jako zakładki. A dodatkowo pola danego

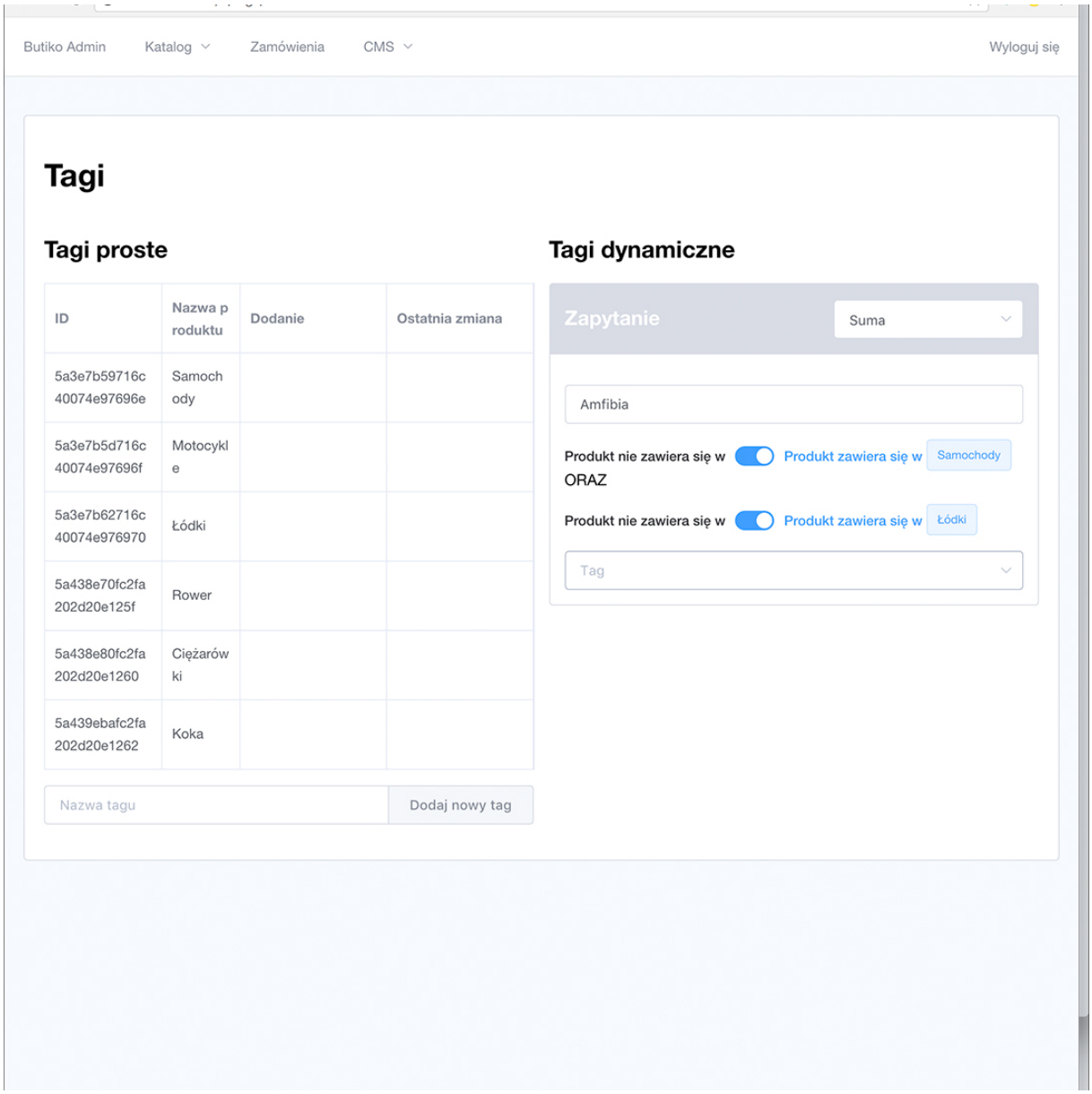


Rysunek 5.10: Butiko: Widok formularza strony

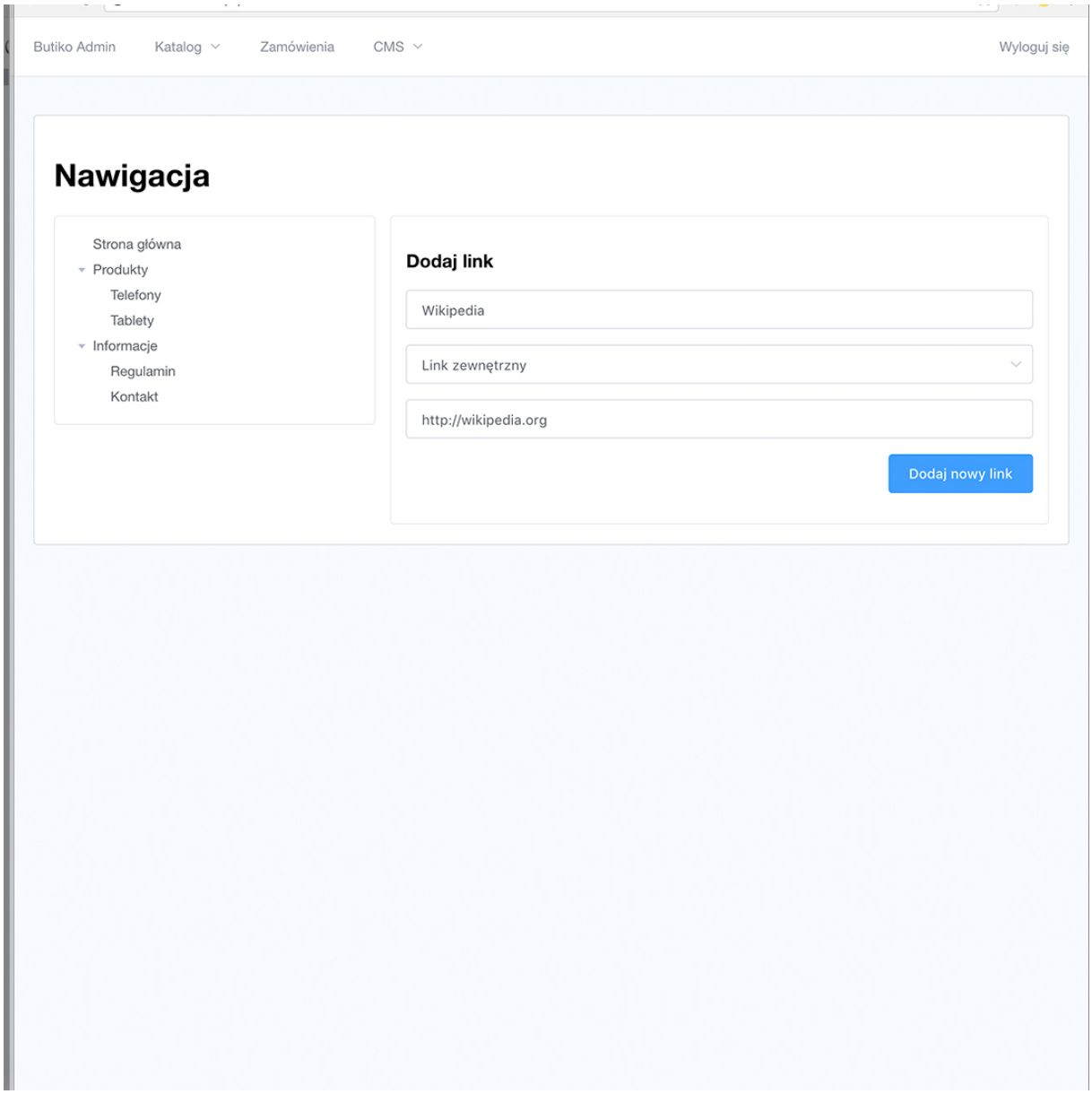
języka został jeszcze podzielony na sekcje takie jak katalog, czy koszyk. Przykład ten pokazuje, że można utworzyć formularz zawierający wiele pól i dzięki odpowiednim strukturom (zakładki i karty) może on zostać przedstawiony w intuicyjnej i nieprzytłaczającej formie.



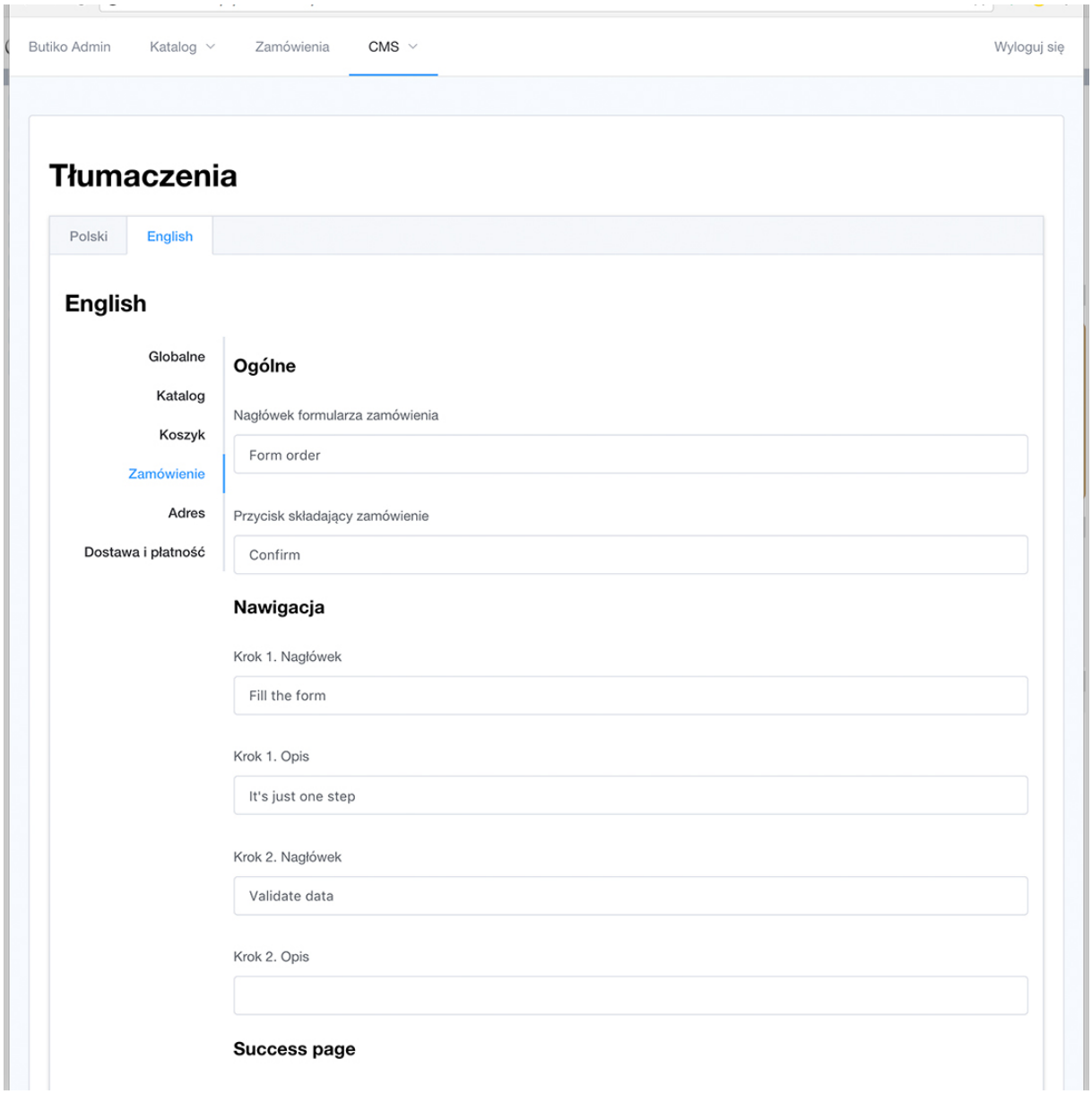
Rysunek 5.11: Butiko: Widok formularza modułu



Rysunek 5.12: Butiko: Widok listy tagów



Rysunek 5.13: Butiko: Widok nawigacji



Rysunek 5.14: Butiko: Widok formularza tłumaczenia

Rozdział 6

Przypadki użycia

W tym rozdziale przedstawione zostaną przypadki użycia, które pokażą różnice w rozwiązaniach przedstawionych w analzie istniejących rozwiązań, a z rozwiązaniami użytych w prototypie.

6.1 Tworzenie nowego produktu

Etap dodawania produktu w systemach opisanych na początku tej pracy miały następujące wady, które utrudniały użytkownikowi bez doświadczenia informatycznego:

- wymagały znajomość HTML,
- zdjęcia wczytywane były przez Flash lub istniała możliwość wgrania tylko jednego zdjęcia,
- walidacja wartości pól formularza odbywała się dopiero po próbie zapisu lub nie było jej w ogóle,
- zarządzanie cechami produktu było skomplikowane.

Dodając nowy produkt użytkownik, kiedy wprowadzi złą wartość jest od razu informowany o tym przez zmiany obramowania pola na czerwony i umieszczenie pod polem komunikatu o błędzie. Zablokowano możliwość wprowadzania liter tam gdzie pole powinno być liczbą, a przy cenie produktu, jeżeli użytkownik wprowadzi przecinek, zamieniany jest on na kropkę.

W prototypie we wszystkich polach, które zawierać powinny kod HTML jest użyty edytor WYSIWYG.

W prototypie wysyłanie zdjęcia na serwer odbywa się przez asynchroniczne funkcje, które wysyłają zdjęcie i czekają na odpowiedź od serwera. Jeżeli odpowiedź będzie pozytywna do listy ze zdjęciami zostanie dodane nowe zdjęcie i dynamicznie zostanie ono dodane do drzewa DOM.

Z poziomu formularza tworzenia nowych produktów można tworzyć jego cechy. W Magento i osCommerce ten proces jest o wiele bardziej skomplikowany. W prototypie dodawanie cech jest dynamiczne. Przykład spodni użyty podczas analizy istniejących rozwiązań, pokaże, że rozwiązanie użyte w prototypie wymaga mniej czasu i jest bardziej intuicyjne. Rysunek 6.1 pokazuje cechy utworzone w formularzu produktów. Na rysunku widać, że istnieje cecha Tkanina, która posiada wartość Bawełna. Bawełniane spodnie mogą mieć różne kolory, na rysunku 6.1 przedstawiono wartość niebieski. Bawełniane spodnie w kolorze niebieskim mają różne rozmiary - S, M, L i XL. Cechy te są powiązane ze stanem magazynowym, więc jest możliwość dopisania ilości do najgłębiej zagnieżdżonej cechy.

6.2 Tworzenie tagów

W systemach Magento i osCommerce katalogowanie produktów stworzone było na bazie dwóch koncepcji:

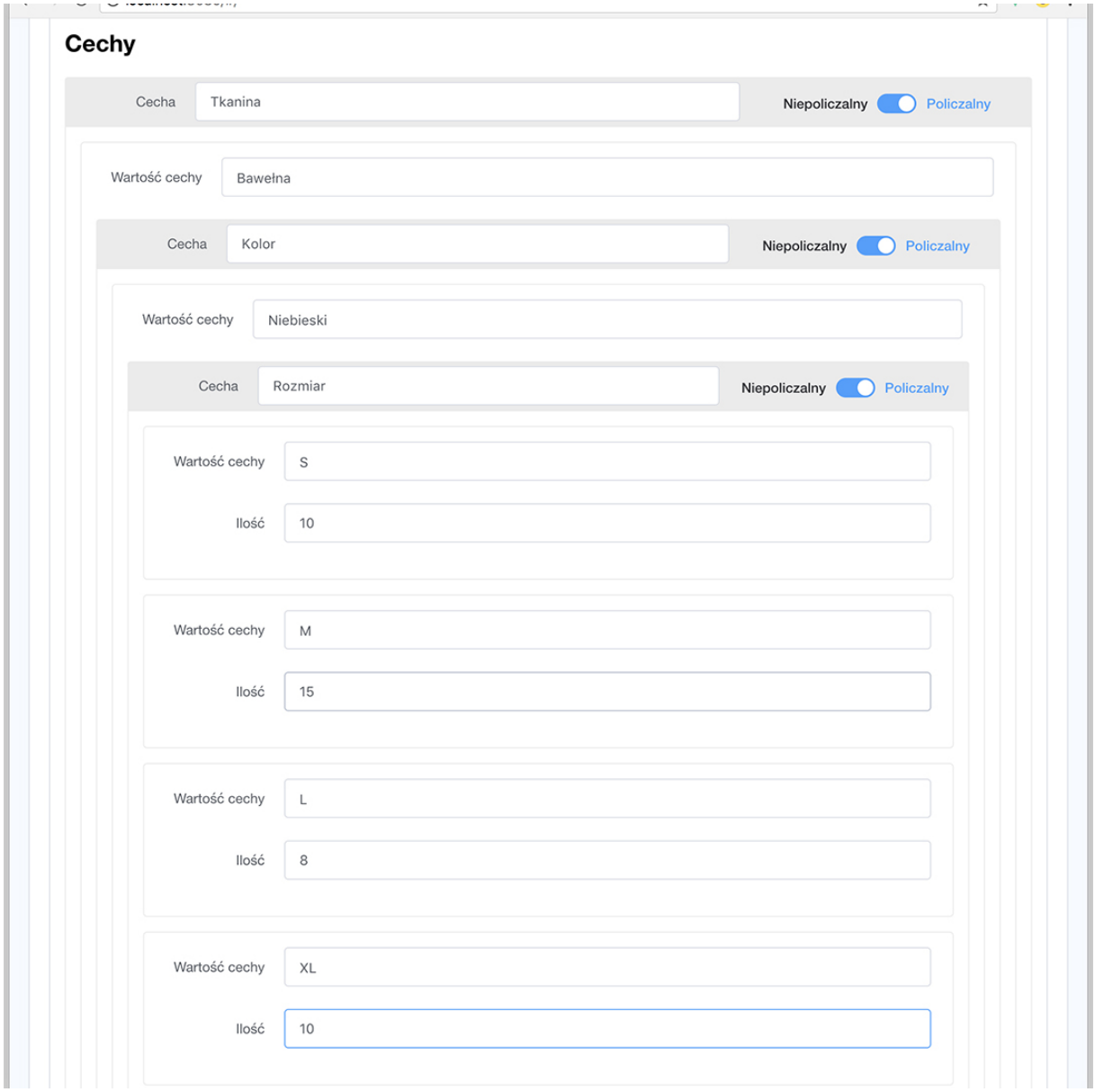
- System pilków - koncepcja ta nie pozwala na dodanie produktu do wielu kategorii,
- Kategorie - koncepcja pozwala na dodanie produktu do wielu kategorii, ale nie można było utworzyć kategorii na bazie operacji na zbiorach

W prototypie przedstawiono koncepcję tagów. Tagom nadano funkcję znane ze zbiorów. Istnieje możliwość tzw. tagów dynamicznych. Tag prosty posiada tylko jedno pole nazwa. Tag dynamiczny oprócz nazwy posiada typ działania na zbiorach (suma, iloczyn, różnica, alternatywa) oraz na tagach które wchodzą w skład równania (możliwa konfiguracja to zawiera i nie zawiera się w zbiorze X). Rysunek 5.12 przedstawia formularz dodawania takiego tagu.

Dzięki koncepcji, że produkty wchodzą w skład zbiorów (a nie do folderu plików) i że między zbiorami mogą występować relacje, katalogowanie produktów jest o wiele bardziej rozbudowane, a jednocześnie nie wymaga od użytkownika zaawansowanej konfiguracji.

Przykładowe zbiory produktów przedstawiono na Rysunku 6.2. W prototypie, aby stworzyć takie zbiory należałoby dodać trzy tagi proste:

- Czapki
- Spodnie

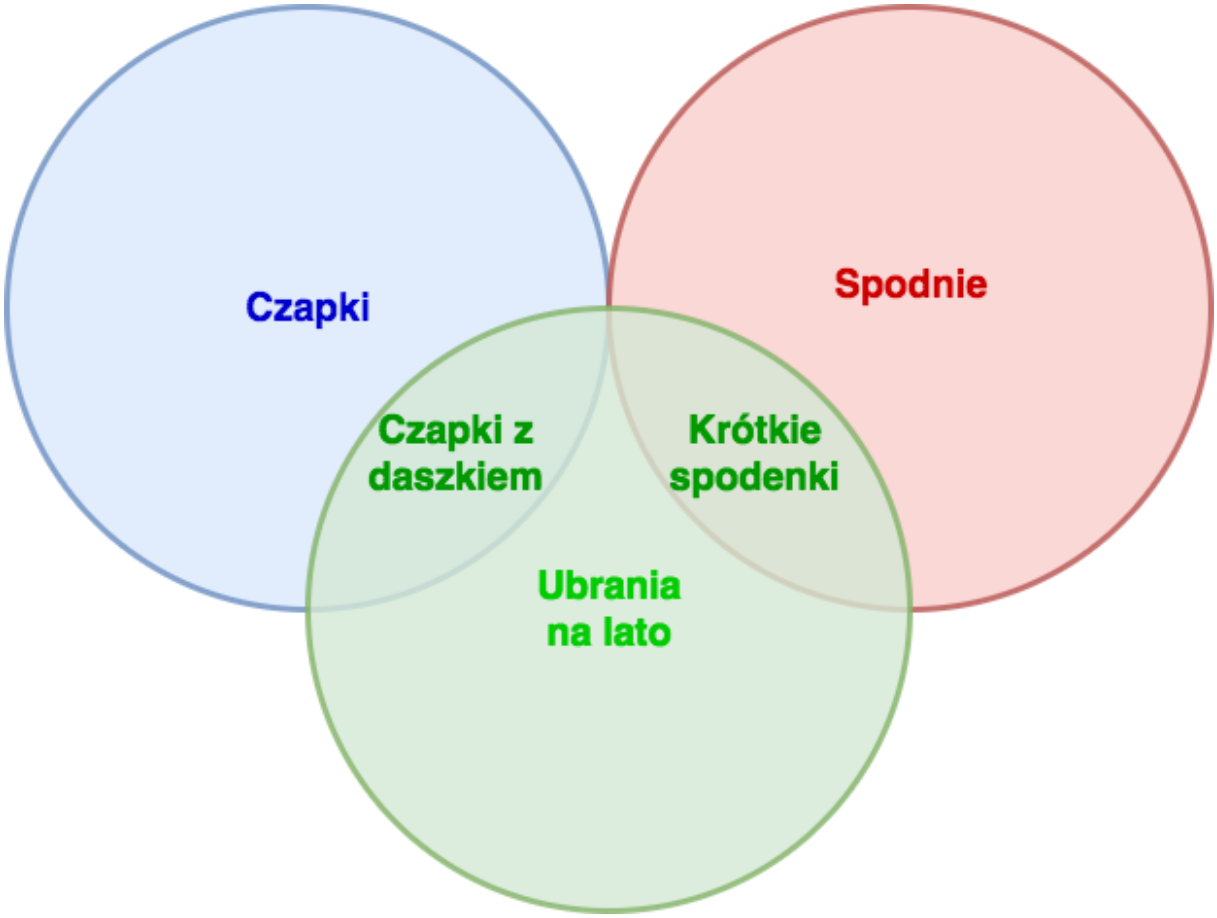


Rysunek 6.1: Butiko: Widok cech produktów

- Ubrania na lato

Aby wyświetlić czapki albo spodnie które nadają się do noszenia latem wystarczy dodać dwa tagi dynamiczne:

- Czapki z daszkiem - będące iloczynem zbiorów czapki i ubrania na lato
- Krótkie spodnie - będące iloczynem zbiorów spodnie i ubrania na lato.



Rysunek 6.2: Przykładowe zbiory produktów

6.3 Zarządzanie tłumaczeniem

Prototyp przedstawia koncepcję zarządzania tłumaczeniem z poziomu Panelu Administracyjnego (Rysunek 5.14). Jest to propozycja, która nie wymaga dużej wiedzy informatycznej w przeciwieństwie do rozwiązań z Magento, gdzie tłumaczenia przechowywane są w plikach CSV, czy w osCommerce, gdzie tłumaczenia przechowywane są w stałych.

W prototypie tłumaczenie znajduje się w jednej kolekcji Mongo. Takie rozwiązanie daje możliwość edytowania tłumaczeń z poziomu Panelu Administratora.

Aplikacja prototypowa sklepu internetowego przechowuje tłumaczenie w Local Storage przeglądarki. Jeżeli nie ma tłumaczenia w Local Storage, tłumaczenie pobierane jest bazy, za pośrednictwem API, jednorazowo i zapisane do Local Storage. Sklep wykonując żądanie do serwera wysyła informację z datą od kiedy znajduje się tłumaczenie w Local Storage. Jeżeli od tego czasu zaszła jakaś zmiana na serwerze to wraz z odpowiedzią serwera zostanie wysłane nowe tłumaczenie.

Takie rozwiązanie daje możliwość użytkownikowi bez wiedzy informatycznej edycję ist-

niejących i tworzenie nowych tłumaczeń.

6.4 Zarządzanie nawigacją

W najpopularniejszych systemach sprzedażowych opisanych wyżej albo nie istnieje możliwości zarządzania nawigacją, albo ogranicza się ona do wyłączania kategorii z nawigacji. Aby dokonać zmian w nawigacji należy zmienić szablon sklepu, co wymaga zalogowania się do FTP lub SSH i edycji kodu PHP i HTML.

Utworzony prototyp posiada funkcjonalność zarządzania nawigacją. Rysunek 5.13 pokazuje formularz zarządzania linkami. Linki zostały podzielone na trzy typy:

- odnośnik do tagu - odnośnik do listy produktów, które posiadają mają określony tag prosty lub dynamiczny,
- odnośnik do strony statycznej - odnośnik do utworzonej strony w panelu administracyjnym strony,
- odnośnik zewnętrzny - odnośnik do zewnętrznej strony.

Wszystkie przypadki wymagają od użytkownika podania nazwy odnośnika. W przypadku odnośników do tagu i strony statycznej użytkownik musi z listy rozwijanej wybrać tag lub stronę statyczną, gdzie ma przekierowywać odnośnik. Odnośnik zewnętrzny wymaga od użytkownika wprowadzenia adresu URL.

6.5 Zarządzanie treścią

W systemie osCommerce zarządzanie statycznymi stronami nie istnieje. Dodawanie stron jest możliwe tylko wgrywając nowe pliki. Aby dodać stronę statyczną wymagana jest więc więdza programistyczna.

Magento oferuje możliwość tworzenia bloków statycznych oraz stron. Jednak dodawanie bloków statycznych do stron, a także dodawanie treści na stronie również wymaga znajomości co najmniej języka HTML.

Zaprezentowana w prototypie koncepcja daje możliwość tworzenia stron statycznych bez wiedzy informatycznej. Idea opiera się na Web Componentach. Skomplikowane struktury HTML i stylu CSS, a także oprogramowanie zachowań elementów w języku JavaScript została schowana w różnego rodzaju komponentach.

Przykładowy slider ze strony głównej (Rysunek 5.7) przyjmuje różne parametry:

- speed - wartość w milesekundach określająca długość wyświetlania slajdu
- slides - tablica z adresami url do obrazów, jakie mają być prezentowane w slajdach.

Cała strukutra HTML, CSS i JavaScript jest ukryta wewnątrz komponentu, natomiast parametry pobierane są z bazy danych. Parametry te użytkownik może zmieniać z poziomu panelu administracyjnego.

Utworzone moduły można dodać następnie na nową stronę. Strona jest zbiorem modułów (Rysunek 5.10).

Koncepcja ta pozwala użytkownikowi stworzyć zaawansowane struktury (które jednak muszą być wcześniej zaimplementowane) bez wiedzy informatycznej.

Rozdział 7

Podsumowanie

W tym rozdziale przedstawione zostaną zalety oraz wykryte wady, a także plan rozwoju prototypu.

7.1 Zalety i wady

Podstawową zaletą utworzonego prototypu jest realizacja określonego celu - stworzenie systemu sprzedażowego, którego odbiorcą będzie użytkownik bez wiedzy informatycznej. W Panelu Administracyjnym użytkownik nie musi nigdzie wprowadzać żadnego kodu HTML, CSS, czy JavaScript.

Użytkownik otrzymał możliwość zarządzania tłumaczeniem i nawigacją, którymi nie można zarządzać w Magento czy osCommerce.

Sklep działa na zasadzie Single Page Application i jest to innowacyjne rozwiązanie, którego nie można zastosować przy tworzeniu sklepu tylko w języku PHP. Aplikacje SPA cechuje intuicyjność, zaawansowane i przjazne GUI oraz bardzo dobre działanie na urządzeniach mobilnych.

Wadami tego systemu mogą być nowe technologię. osCommerce i Magento zostało napisane w języku PHP i MySQL. Na rynku istnieje zdecydowanie więcej firm oferujących hosting/serwer z tymi językami, niż z wymaganymi przez prototyp Dockerem, Node.js, czy Mongo DB.

Wadą prototypu jest ograniczona liczba funkcjonalności. Prototypowi bliżej jest osCommerce niż Magento, które oferuje bardzo zaawansowane funkcje raportowania, czy zarządzania klientami.

Mimo, że prototyp daje możliwość tworzenia stron statycznych bez wiedzy programistycz-

nej to jednak tworzenie treści ogranicza się do liczy komponentów, które zostały wcześniej zaimplementowane.

7.2 Plan rozwoju

Sporządzenie listy wad rozwiązania pozwoliły na opracowanie planu rozwoju prototypu.

Dzięki stworzeniu odpowiedniej ilości komponentów, a także stworzeniu zaawansowanego systemu raportowania i zarządzania klientami prototyp ma szanse konkurować z Magento.

Problemy z architekturą w przyszłości moim zdaniem będzie rozwiązany, gdyż popularność Dockera i Node.js cały czas bardzo szybko rośnie.

7.3 Zakończenie

Rynek ecommerce wciąż się rozwija. Przed twórcami oprogramowania stoi wyzwanie stworzenia zaawansowanych rozwiązań i połączenia tego z łatwością obsługi. W niniejszej pracy przedstawiona zostały technologię i koncepcja, która może stać się lepszą alternatywą dla obecnych rozwiązań.

Bibliografia

[1] BSON. Bson (binary json) serialization. http://bsonspec.org/. Dostęp: 2017-11-27.

[2] Docker. Overview of docker hub. https://docs.docker.com/docker-hub/. Dostęp: 2017-11-27.

[3] Docker. That is docker? https://www.docker.com/what-docker. Dostęp: 2017-11-27.

[4] Gemius. E-commerce w polsce 2017. gemius dla e-commerce polska. https://www.gemius.pl/wszystkie-artykuly-aktualnosci/najnowsze-dane-o-polskim-e-commerce-juz-dostepne.html. Dostęp: 2017-11-27.

[5] Git. About git. https://git-scm.com/about/. Dostęp: 2017-11-27.

[6] GitHub. The world's largest leading software development platform. https://github.com/. Dostęp: 2017-11-27.

[7] Google. Chrome v8. https://developers.google.com/v8/. Dostęp: 2017-11-27.

[8] S. Jobs. Thoughts on flash. https://www.apple.com/hotnews/thoughts-on-flash/. Dostęp: 2017-11-27.

[9] JSON. Introducing json. http://json.org/. Dostęp: 2017-11-27.

[10] S. Krause. Javascript framework benchmark. https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts-results/table.html. Dostęp: 2017-11-27.

[11] MongoDB. What is mongodb? https://www.mongodb.com/what-is-mongodb. Dostęp: 2017-11-27.

[12] Netflix. Node.js in flames. https://medium.com/netflix-techblog/node-js-in-flames-ddd073803aa4. Dostęp: 2017-11-27.

[13] Node.js. About node.js. https://nodejs.org/en/about/. Dostęp: 2017-11-27.

[14] OsCommerce.com. oscommerce online merchant v2.3.4. https://www.oscommerce.com/Us&News=150. Dostęp: 2017-11-27.

[15] Paypal. Node.js at paypal. https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/. Dostęp: 2017-11-27.

[16] Robo3T. Native mongodb management tool (admin ui). https://robomongo.org/. Dostęp: 2017-11-27.

[17] Techtarget.com. Statistics for websites using open source technologies. https://trends.builtwith.com/shop/open-source. Dostęp: 2017-11-27.

[18] v2tec. Wachtower. https://github.com/v2tec/watchtower. Dostęp: 2017-11-27.

[19] Vue.JS. Comparison with other frameworks. https://vuejs.org/v2/guide/comparison.html. Dostęp: 2017-11-27.

[20] Vuex. What is vuex? https://vuex.vuejs.org/en/intro.html. Dostęp: 2017-11-27.

[21] W3C. Custom elements. https://w3c.github.io/webcomponents/spec/custom/. Dostęp: 2017-11-27.

[22] W3C. Html imports. https://w3c.github.io/webcomponents/spec/imports/. Dostęp: 2017-11-27.

[23] W3C. Shadow dom. https://w3c.github.io/webcomponents/spec/shadow/. Dostęp: 2017-11-27.

[24] WHATWG. Html standard. https://html.spec.whatwg.org/multipage/scripting.html#the-template-element. Dostęp: 2017-11-27.