POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA (INF)

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH (INS)

PRACA DYPLOMOWA

INŻYNIERSKA

Tytuł pracy

Shop Online

AUTOR:

Paweł Idziak

PROWADZĄCY PRACĘ:

Dr inż. Dariusz Jankowski, KSSK

OCENA PRACY:

WROCŁAW, 2017

**Spis treści**

[Spis rysunków 4](#_Toc499221462)

[Spis tabel 5](#_Toc499221463)

[Spis listingów 6](#_Toc499221464)

[1. Wstęp 7](#_Toc499221465)

[1.1. Cel pracy 7](#_Toc499221466)

[1.2. Stan wiedzy obecnej 7](#_Toc499221467)

[1.3. Przedstawienie zawartości poszczególnych rozdziałów pracy 10](#_Toc499221468)

[2. Teoria 11](#_Toc499221469)

[2.1. Rys historyczny, rozwój rynku elektronicznego 11](#_Toc499221470)

[2.2. Użyte technologie 14](#_Toc499221471)

[2.2.1. Angular 4 i TypeScript 14](#_Toc499221472)

[2.2.2. HTML i CSS (SASS) 17](#_Toc499221473)

[2.2.3. Firebase 20](#_Toc499221474)

[2.2.4. GIT / Bitbucket 23](#_Toc499221475)

[3. Implementacja 24](#_Toc499221476)

[3.1. Założenia projektowe 24](#_Toc499221477)

[3.2. Wykorzystywane technologie i narzędzia 25](#_Toc499221478)

[3.3. Opis działania systemu 26](#_Toc499221479)

[3.3.1. Rejestracja 27](#_Toc499221480)

[3.3.2. Logowanie 30](#_Toc499221481)

[3.3.3. Przeglądanie produktów 31](#_Toc499221482)

[3.3.4. Wyszukiwanie produktów 33](#_Toc499221483)

[3.3.5. Dodawanie produktów do koszyka 34](#_Toc499221484)

[3.3.6. Logika koszyka 37](#_Toc499221485)

[3.3.7. Finalizacja zamówienia 40](#_Toc499221486)

[3.4. Instrukcja instalacji 41](#_Toc499221487)

[4. Podsumowanie 42](#_Toc499221488)

[Literatura 43](#_Toc499221489)

# Spis rysunków

[Rys. 1. Logo Angular 4 14](file:///D:\Workspace\e-commerce\praca%20inzynierska.docx#_Toc499207884)

[Rys. 2. Logo TypeScript 17](file:///D:\Workspace\e-commerce\praca%20inzynierska.docx#_Toc499207885)

[Rys. 3. Logo HTML5 17](file:///D:\Workspace\e-commerce\praca%20inzynierska.docx#_Toc499207886)

[Rys. 4. Logo 18](file:///D:\Workspace\e-commerce\praca%20inzynierska.docx#_Toc499207887)

[Rys. 5. Panel administracyjny Firebase 20](file:///D:\Workspace\e-commerce\praca%20inzynierska.docx#_Toc499207888)

[Rys. 6. Dostępne metody logowania w Firebase 21](#_Toc499207889)

[Rys. 7. Baza danych Firebase 22](file:///D:\Workspace\e-commerce\praca%20inzynierska.docx#_Toc499207890)

[Rys. 8. Reguły zabezpieczeń bazy danych Firebase 23](#_Toc499207891)

[Rys. 9. Diagram przypadków użycia 26](#_Toc499207892)

[Rys. 10. Formularz rejestracji 28](#_Toc499207893)

[Rys. 11. Formularz logowania, dane przykładowe 30](#_Toc499207894)

[Rys. 12. Wyszukiwanie produktów 34](file:///D:\Workspace\e-commerce\praca%20inzynierska.docx#_Toc499207895)

Spis będzie aktualizowany.

# Spis tabel

[Tab. 1. Porównanie języka TypeScript i JavaScript 17](#_Toc499207869)

[Tab. 2. Przykład użycia HTML wraz z CSS 18](#_Toc499207870)

[Tab. 3. Porównanie CSS z SCSS 19](#_Toc499207871)

[Tab. 4. Przykład deklaracji domieszki i jej użycie: 19](#_Toc499207872)

[Tab. 5. Przykład deklaracji funkcji i jej użycie: 19](#_Toc499207873)

Spis będzie aktualizowany.

# Spis listingów

[Listing 1. Przykład wstrzykiwania zależności w Angular 4 15](#_Toc499207839)

[Listing 2. Kod odpowiedzialny za logowanie 30](#_Toc499207840)

[Listing 3. Wyszukiwanie produktów 33](#_Toc499207841)

Spis będzie aktualizowany.

1. Wstęp
   1. Cel pracy

Celem pracy jest zaprojektowanie i wykonanie aplikacji dającej możliwość kupowania i sprzedawania produktów przez Internet. Głównym zadaniem jest stworzenie portalu, na którym klienci mogą zapoznać się z ofertą i składać zamówienia. Aplikacja realizuje funkcjonalności bezpośredniego handlu elektronicznego (patrz punkt 1.2) oraz jest responsywna (patrz punkt 3.1), by można korzystać z niej na każdym urządzeniu niezależnie od wielkości ekranu czy zainstalowanego sytemu.

Podstawowa funkcjonalność to:

- rejestracja oraz logowanie;

- przeglądanie produktów (dla wszystkich klientów),

- zamówienie produktu (dla zalogowanego użytkownika),

- zmiana danych użytkownika.

Do stworzenia aplikacji zostały użyte najnowsze technologie internetowe i bazodanowe wraz ze wsparciem systemu kontroli wersji. Każda z użytych technologii / narzędzi zostanie omówiona w dalszej części pracy (patrz punkt 0).

* 1. Stan wiedzy obecnej

Handel elektroniczny [1] jest to rodzaj handlu polegający na sprzedaży towarów i usług oraz ich zakupie poprzez wykorzystanie sieci Internet jako środku wymiany pomiędzy sprzedawcą a konsumentem. Ze względu na swój globalny zasięg handel elektroniczny umożliwia dotarcie do konsumentów z całego świata.

Rodzaj handlu jaki reprezentuje zaimplementowana aplikacja to bezpośredni handel elektroniczny. Oznacza to, że całość transakcji poczynając od przeglądnięcia i wybrania produktu, aż do realizacji zamówienia ma miejsce poprzez sieć Internet.

Rozwiązań implementacji sklepu internetowego jest wiele. Poczynając od własnych implementacji, przez proste systemy zarządzania treścią, aż po potężne systemy używane przez duże firmy. Prowadzenie e-biznesu wiąże się z całą logistyką sprzedaży, dlatego podczas wyboru oprogramowania dla sklepu internetowego trzeba uwzględnić między innymi takie czynniki jak skalowalność oprogramowania, integracja z systemami zewnętrznymi czy wymogi serwerowe.

Jednym z największych i najpopularniejszych systemów do zarządzania sklepem internetowym jest platforma Magento. Jest to rozbudowany system, który polecany jest dla największych firm, posiadających dużą liczbę produktów. Przykłady firm posiadających swój sklep internetowy w platformie Magento to między innymi: FORD Motors, Nike, Cicso Systems.

Magento posiada szereg rozbudowanych funkcjonalności takich jak system analiz, raportowanie błędów, wbudowany system płatności czy tworzenie różnych szablonów zmieniających wygląd dla rożnych produktów. Dodatkowo system Magento umożliwia zarządzanie z panelu administratora kilkoma sklepami na raz.

Kolejną informacją o której warto wspomnieć jest wsparcie techniczne, które w przypadku systemu Magento jest bardzo dobre, posiada swoje odpowiedniki zarówno w języku angielskim jak i polskim. Ze względu na dużą społeczność, która korzysta z systemu, w sieci Internet znajduje się duża liczba informacji jak i poradników na temat wszelkich zagadnień związanych z platformą Magento.

Architektura *Magento* jest na tyle elastyczna, że rozbudowanie wybranych funkcjonalności doświadczonemu programiście nie powinno sprawić trudności. Język używany w systemie to język *PHP* [2].

Zalety:

- rozbudowane funkcjonalności,

- indywidualne szablony produktów,

- dobre wsparcie techniczne,

- możliwość zarządzania kilkoma sklepami z jednego panelu administracyjnego,

- możliwość indywidualnej rozbudowy skryptu.

Wady:

- duże wymagania serwerowe,

- skomplikowana administracja,

- platforma mało intuicyjna.

Kolejnym, również popularnym rozwiązaniem jest platforma *PrestaShop*. Skrypt ten przeznaczony jest dla mniejszych firm, które nie posiadają w swoim asortymencie wielu produktów. Ze względów na swoją intuicyjność, *PrestaShop* jest polecany tym, którzy dopiero rozpoczynają działalność związaną z handlem elektronicznym.

Platforma jest intuicyjna, co stanowi atut dla początkujących użytkowników systemu, posiada rozbudowaną społeczność, dzięki której istnieje wiele materiałów i poradników dotyczących pierwszych kroków w platformie, co również ułatwia wdrożenie projektu.

Wsparcie techniczne jest rozbudowane, przykładem mogą być polskie forma poświęcone tematowi platformy *PrestaShop*[[1]](#footnote-1)

System posiada moduły umożliwiające miedzy innymi obsługę dowolnej liczby wersji językowych, wystawianie faktur, zarządzanie płatnościami, śledzenie przesyłki i wiele innych[[2]](#footnote-2). Niestety w darmowej wersji użytkownik utrzymuje jedynie dostęp do podstawowych funkcjonalności, natomiast te bardziej rozbudowane są płatne.

Zalety:

- intuicyjny interfejs,

- prosta instalacja i modyfikacja,

- wydajność (dla relatywnie małej liczby produktów),

- rozbudowana społeczność.

Wady:

- płatne funkcjonalności,

- spora ilość poprawek,

- trudność w modyfikacji skryptów.

Istnieją tez inne systemy, tabelka, porównanie itp. (nazwa, cechy, opis)

* 1. Przedstawienie zawartości poszczególnych rozdziałów pracy

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

**+ to na końcu**

1. Teoria
   1. Rys historyczny, rozwój rynku elektronicznego

Na przestrzeni ostatnich lat, można zaobserwować dynamiczny rozwój handlu elektronicznego, co ma swoją przyczynę zarówno w rozwoju technologicznym jak i wzrostem liczby internautów. Dużym czynnikiem na wzrost popularności handlu elektronicznego miała również popularyzacja urządzeń mobilnych (tzw. „era urządzeń mobilnych”).

Jeszcze kilka lat temu (jakaś wzmianka ze 2017?) handel elektroniczny był na tyle mało popularny, że większość konsumentów nabywało dobra lub usługi w tradycyjny, „stacjonarny” sposób. Ludziom brakowało zaufania do zawierania transakcji przez sieć Internet. W czasach obecnych jest inaczej. Gwałtowny rozwój samej sieci Internet, wzrost zabezpieczeń podczas przesyłania danych czy powstawanie stron, a w tym sklepów internetowych, i ich globalna popularyzacja zaowocowała wzrostem liczby internautów. W roku pisania pracy tj. 2017 szacowana liczba internautów wynosi 3.4 biliona, co daje 46% populacji świata, z prawdopodobieństwem wzrostu do 56% do roku 2022[[3]](#footnote-3).

Prognozy dotyczące handlu elektronicznego na rok 2017, przewidują że w najbliższych latach zakupy dokonywane przez Internet będą w głównej mierze dokonywane za pośrednictwem urządzeń mobilnych, takich jak telefon komórkowy czy tablet. Urządzenia te umożliwiają korzystanie z Internetu z dowolnego miejsca, o dowolnej porze, co jest wielką zaletą dla sklepów internetowych, ponieważ użytkownik może w łatwy i szybki sposób dokonać zakupu. Warto więc zapewnić, by strona sklepu internetowego była dopasowana do urządzeń o mniejszym ekranie. Czynniki te mają wpływ na komfort potencjonalnego klienta, którego staramy się pozyskać.

Korzystanie z dóbr i usług sklepów stacjonarnych wiąże się z poświęceniem czasu i pewnymi kosztami np. dojazdu. Warto również wspomnieć, że wybierając się do centrum handlowego często możemy natrafić na kolejki, przez co tracimy nasz czas, przykuwające naszą uwagę, niezliczone banery informujące o wszelkich promocjach, czy płatności jedynie gotówkowe, co w dobie Internetu może okazać się uciążliwe. Zakupy produktów o większych gabarytach wiąże się również z kosztami dowozu do wskazanego miejsca lub posiadaniem własnego środka transportu.

Dzięki handlowi elektronicznemu mamy możliwość wyeliminowania wyżej wymienionych niedogodności. Mamy możliwość robienia zakupów nie wychodząc z domu, oszczędzając nasz czas, a często i pieniądze. Płatności elektroniczne udostępnianie przez sklepy internetowe są odpowiednio zabezpieczone, duża część z płatności jest dokonywanych w czasie rzeczywistym, dzięki czemu nie musimy czekać na zaksięgowanie transakcji w banku sprzedawcy.

Siedząc wygodnie w fotelu czy przy biurku możemy przeglądać interesujące nas dobra / usługi nie natrafiając przy tym na kolejki czy problemy związane z dostaniem się do jednostki sklepu. Obecnie większość sklepów internetowych oferuje swoim klientom sprawdzenie produktu przed jego zakupem, czy też bezpłatny zwrot przed upływem określonej liczby dni od daty zakupu, dlatego klient nie musi obawiać się jakości czy stanu towaru.

Towary w głównej mierze dostarczane są przez firmy zewnętrzne oferujące usługę dostaw dóbr pod wskazany adres, więc problem transportu zamówionych towarów jest rozwiązywany przez sprzedawcę.

W latach wcześniejszych, gdy zakupy przez Internet dla większości ludzi były nowością i jedynie pewnym zamysłem, do tworzenia sklepów internetowych używano takich języków jak *HTML* [3] wraz z *CSS* [4] do prezentacji strony internetowej i *PHP* [5] do komunikacji strony z bazą danych. Bazy danych były w głównej mierze bazami danych typu *SQL* [6].

Na przestrzeni ostatnich lat wachlarz dostępnych technologii mogących posłużyć do implementacji aplikacji internetowej, powiększył się o wiele nowych pozycji. Wybór jest spory, a każda z technologii wiąże się z szeroko rozumianym pojęciem „tworzenia stron internetowych” (ang. *web development*). Strony internetowe zostały wzbogacone o skryptowy język wykonywany po stronie klienta (*JavaScript*), interpretowany przez przeglądarkę. Dzięki temu na stronie zostały wprowadzone dynamiczne, a nie jedynie statyczne, elementy.

W dzisiejszych czasach po stronie klienckiej (strona internetowa) w głównej mierze wykorzystywane są języki takie jak *HTML5* [7], *CSS3* [8] i *JavaScript* [9], który stał się na tyle popularny, że doczekał się implementacji języków ze zgodną kompilacją wsteczną (np. *TypeScript*) czy też Framework’ów takich jak *Angular*, *React*, czy *Vue.js*.

Po stronie aplikacji realizującej funkcje serwera (ang. *backend*), możliwości również jest sporo, poczynając od wcześniej wymienionego języka *PHP* który w dniu pisania pracy doczekał się już 7 wersji, przez język obiektowy *JAVA*, *ASP.NET*, a nawet *JavaScript*. W przypadku użycia języka JavaScript należy użyć środowiska uruchomieniowego Node.js, dzięki któremu napisany kod będzie wykonywany po stronie serwera.

Bazy danych można podzielić na bazy typu *SQL* (ang. *Structured Query Language*), oraz bazy danych typu *NoSql* (ang. *Not Only SQL*). *NoSQL* [12] jest systemem baz danych, który, w odróżnieniu od baz danych typu SQL, nie posiada relacji oraz zapytań *SQL*, dane nie posiadają określonego schematu (ang. *schema*) i są przechowywane w postaci obiektów.

Wybór zarówno języków wykorzystywanych w aplikacji klienckiej, aplikacji serwerowej czy rodzaju bazy danych zależy tylko i wyłącznie od preferencji programisty. Podczas wyboru warto jednak zwrócić uwagę na wsparcie techniczne, społeczność użytkowników, dostępność literatury, czy wymagania serwerowe danego rozwiązania.

* 1. Użyte technologie

W poniższym podrozdziale przedstawiłem opis teoretyczny użytych w projekcie technologii, wraz z wytłumaczeniem pewnym zagadnień. Do implementacji aplikacji klienckiej został wykorzystany Framework *Angular 4* wraz z językami: *TypeScript*, *HTML* *5* i *SCSS*, logika aplikacji serwerowej znajduje się w platformie *Firebase*, a użyta baza danych jest typu *NoSQL*.

* + 1. Angular 4 i TypeScript

*Angular* [10] jest to platforma internetowa o otwartym kodzie (ang. open source) rozwijana przez firmę *Google.* *Angular* jest platformą do budowy aplikacji klienckiej opartej na językach *HTML* oraz językach kompilowanych do *JavaScript*. Platforma posiada wbudowane funkcjonalności takie jak szablony deklaratywne, wstrzykiwanie zależności oraz własną linię komend (CLI, ang. Commad Line Interface), dzięki którym w łatwy sposób można stworzyć nowy projekt czy nowy komponent. *Angular* umożliwia programiście implementację aplikacji internetowej, mobilnej lub komputerowej.

Rys. 1. Logo Angular 4

Szablony deklaratywne pozwalają oddzielić kod widoku (ang. *View*), gdzie znajduje się kod odpowiedzialny za wyświetlanie aplikacji (HTML i CSS), od kodu kontrolera (ang. Controller), gdzie znajduje się kod *TypeScript* odpowiadający za logikę danego fragmentu aplikacji. Dzięki szablonom deklaratywnym możemy emulować kod kontrolera z kodem widoku i na odwrót.

Wstrzykiwanie zależności (ang. *Dependency Injection*) jest to wzorzec projektowy polegający na usunięciu bezpośrednich zależności pomiędzy komponentami. Wzorzec ten przekazuje do komponentu gotowe instancje obiektów (tzw. serwisów) udostępniających swoje metody i właściwości. Obiekty te wstrzykuje się do konstruktora danego komponentu.

Listing 1. Przykład wstrzykiwania zależności w Angular 4

|  |
| --- |
| // child.component.ts  class ChildComponent {  constructor(private dataService: DataService) { ... }  }  // data.service.ts  @Injectable()  export class DataService { ... }  // app.component.ts  @Component({  ...  directives: [MyChildComponent],  providers: [DataService],  ...  })  class MyAppComponent { ... } |

Jak wspomniałem wyżej, *Angular* posiada własną linię komend. Dla programisty jest to duża wygoda, ponieważ dzięki prostej komendzie możemy stworzyć szkielet między innymi nowego projektu, komponentu, serwisu, klasy czy interfejsu[[4]](#footnote-4). Dodatkowo CLI umożliwia nam zbudowanie aplikacji, wystartowanie jej w przeglądarce wraz z nasłuchiwaniem zmian w plikach projektu, dzięki czemu gdy dodamy i / lub zmienimy kod, *Angular* automatycznie wykryje zmianę i odświeży stronę załadowaną w przeglądarce.

CZY TE PRZYKŁADY WGL MOGĄ BYĆ, nie za dlugi przykład? CZY DO KODU (NAWET JEDNOLINIJKOWEGO) DODAWAĆ ADNOTACJĘ Listing?

Przykłady użycia Angular CLI:

1. Aby móc zainstalować CLI, wymagany jest Node w wersji 6.9.0 lub wyższej oraz NPM w wersji 3 lub wyżej.
2. Instalacja:

npm install -g @angular/cli

1. Tworzenie nowego projektu:

ng new PROJECT-NAME

Po tej operacji, został stworzony nowy folder o nazwie PROJECT-NAME, a w nim wszystkie potrzebne foldery i pliki. W oknie konsoli powinniśmy ujrzeć:

create PROJECT-NAME/e2e/app.e2e-spec.ts (294 bytes)

create PROJECT-NAME/e2e/app.po.ts (208 bytes)

create PROJECT-NAME/e2e/tsconfig.e2e.json (235 bytes)

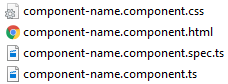
create PROJECT-NAME/karma.conf.js (923 bytes)

create PROJECT-NAME/package.json (1317 bytes)

create PROJECT-NAME/protractor.conf.js (722 bytes)

create PROJECT-NAME/README.md (1027 bytes)

...



Rys. 1. Zawartość folderu stworzonego komponentu

1. Tworzenie nowego komponentu:

cd PROJECT-NAME\

ng generate component COMPONENT-NAME

Na początku przeszliśmy do folderu z projektem, a następnie wygenerowaliśmy szkielet komponentu. W oknie konsoli widnieją nazwy plików stworzonego komponentu, oraz informacja, że plik *app.module.ts* został zaktualizowany (informacja odnoście stworzonego komponentu).

create src/app/component-name/component-name.component.html (33 bytes)

create src/app/component-name/component-name.component.spec.ts (678 bytes)

create src/app/component-name/component-name.component.ts (300 bytes)

create src/app/component-name/component-name.component.css (0 bytes)

update src/app/app.module.ts (520 bytes)

Przedstawione powyżej przykłady to jedynie namiastka możliwości linii komend Framework’u *Angular*. Po więcej przykładów należy zajrzeć do oficjalnej dokumentacji [[5]](#footnote-5).

W przedstawianej pracy, wraz z platformą (ang. Framework) *Angular 4* został użyty język programowania *TypeScript* [11].

Logo języka*TypesScript* jest to otwarto źródłowy (ang. open source) język programowania, będący nadzbiorem (ang. super set) języka *JavaScript*. Umożliwia on programowanie zorientowane obiektowo, posiada klasy, interfejsy i typy wyliczeniowe. Ponad to *TypeScript* posiada typowane zmienne dzięki czemu możemy określić typ danej zmiennej, funkcji czy argumentów. Język ten jest kompilowany wstecznie do JavaScript, więc wynikowo otrzymujemy kod JavaScript który jest interpretowany przez przeglądarkę.

Rys. 2. Logo TypeScript

Tab. 1. Porównanie języka TypeScript i JavaScript

|  |  |
| --- | --- |
| TypeScript | JavaScript |
| class SomeClass {  variable: string;  constructor(message: string) {  this.variable = message;  }  someFunction(){  return „Hello.”;  }  } | var SomeClass = (function () {  function SomeClass(message) {  this.variable = message;  }  SomeClass.prototype.someFunction = function() {  return „Hello.”;  }  return SomeClass;  })(); |

*TypeScrpit* stanowi jedynie „otoczkę” języka *JavaScript* oferując nam możliwość wygodniejszego programowania obiektowego. Jak widać w zamieszczonej wyżej tabeli kod TypeScript jest czytelniejszy i bardziej skalowalny.

* + 1. HTML i CSS (SASS)

*HTML5* (ang. *HyperText Markup Language*) jest językiem znaczników wykorzystywanym do tworzenia jak i prezentowania stron internetowych. Język *HTML5* pozwala opisać strukturę informacji, które powinny znaleźć się na stronie internetowej oraz nadaje znacznie odpowiednim selektorom.

Rys. 3. Logo HTML5

*CSS* (ang. *Cascading Style Sheets*) czyli kaskadowe arkusze stylów to język, za pomocą którego opisujemy formę prezentującą stronę internetową. Arkusz *CSS* zawiera listę reguł, które określają sposób wyświetlenia przez przeglądarkę internetową danego elementu. Można powiedzieć, że pliki *HTML* stanowią szkielet strony prezentacyjnej, natomiast pliki *CSS* określają jak ten szkielet wygląda.

Rys. 4. Logo

CSS 3

Tab. 2. Przykład użycia HTML wraz z CSS

|  |  |
| --- | --- |
| HTML | CSS |
| <!DOCTYPE html>  <html>  <head>  <title>Tyuł strony</title>  </head>  <body>  <div class="kontener">    <div class="przycisk">  <button>Przycisk 1</button>  </div>    </div>  </body>  </html> | .kontener {  background: black;  }  .przycisk {  margin: 0 auto;  } |

W projekcie zamiast kaskadowych arkuszy stylów (ang. CSS) w podstawowej wersji, został użyty język *SASS*. *SASS* jest to preprocesor *CSS*, który powstał w 2006 r. Jest to nadzbiór (ang. *super set*) języka CSS. Kod SASS jest kompilowany do wynikowego kodu CSS, a ten jest interpretowany przez przeglądarkę. Język SASS umożliwia programiście używania tzw. zagnieżdżonych reguł (patrz Tab. 3. Porównanie CSS z SCSS) i pozwala przyśpieszyć pracę. Programista ma możliwość wyboru typu składni, czyli SASS albo SCSS.

Tab. 3. Porównanie CSS z SCSS

|  |  |
| --- | --- |
| CSS | SCSS |
| #kontener {  width: 700px;  }  #kontener h1 {  font-size: 24px;  color: #fff;  }  #kontener h2 {  font-size: 24px;  color: #fff;  } | #kontener {  width: 700px;  h1 {  font-size: 24px;  color: #fff;  }  h2 {  font-size: 24px;  color: #fff;  }  } |

W języku SASS możemy również tworzyć zmienne, domieszki (ang. *mixin*) i funkcje, co znacznie ułatwia i przyśpiesza pracę, a dodatkowo kod staje się re-używalny, dzięki czemu unikamy redundancji kodu.

Przykład deklaracji zmiennej: $font-color: #000000;

Tab. 4. Przykład deklaracji domieszki i jej użycie:

|  |  |
| --- | --- |
| Deklaracja domieszki | Użycie domieszki |
| @mixin mixin-name($some-variable) {  color: $some-variable;  } | .kontener {  @include mixin-name(20px);  } |

Tab. 5. Przykład deklaracji funkcji i jej użycie:

|  |  |
| --- | --- |
| Deklaracja funkcji | Użycie funkcji |
| @function function-name ($num-1,  $num2){  @return $num1 + $num2  } | .kontener {  padding: function-name (10px, 5px);  } |

* + 1. Firebase

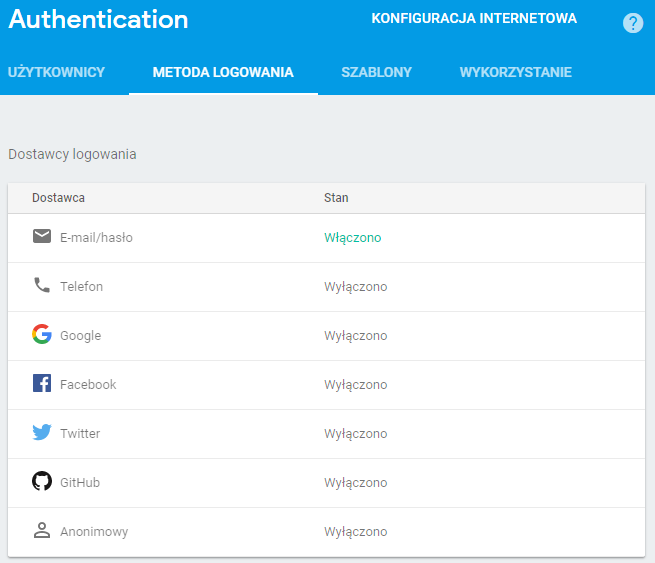
Firebase jest platformą firmy Google, umożliwiającą tworzenie zaplecza aplikacji internetowych i / lub aplikacji mobilnych na systemy Android czy iOS. Platforma posiada innowacyjne podejście, oferuje programiście prostotę i oszczędność czasu. Dzięki niej nie musimy tworzyć modelów encji czy relacji pomiędzy nimi. Co więcej, platforma *Firebase* posiada wbudowany system autoryzacji i uwierzytelnienia, więc programista może skupić się na produkcie i tym co zobaczy klient, a nie na podstawach logiki, które powtarzają się w większości aplikacji.

W skład całej platformy wchodzi wiele modułów tj. metody logowania, baza danych, powiadomienia, analiza błędów, statystyki użytkowników i wiele innych.. Każdą z funkcjonalności można używać niezależnie od siebie, dowolnie łącząc je z innymi produktami. Wszystkie funkcjonalności są gotowe do użycia zaraz po stworzeniu projektu, tuż po prostej konfiguracji. Wszystko prezentuje nam panel administracyjny, w którym możemy zarządzać daną funkcjonalnością.



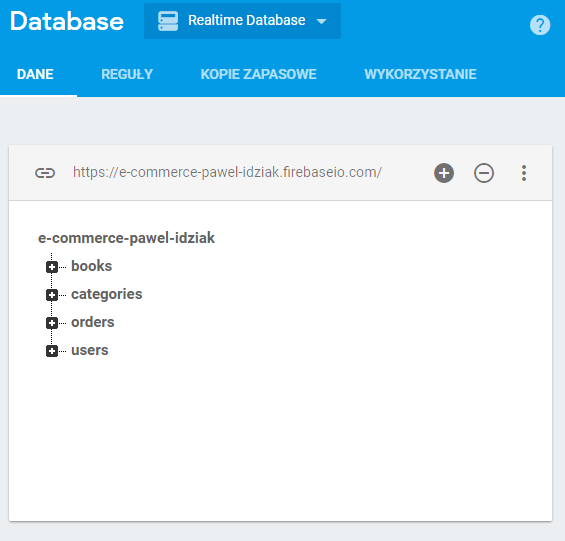
Rys. 5. Panel administracyjny Firebase

Firebase umożliwia nam wybranie kilku metod logowania (patrz Rys. 6). Na potrzeby projektu została uruchomiona metoda logowania poprzez założenie konta w systemie podając adres e-mail oraz hasło. W przypadku uruchomienia innych metod, należy skorzystać z instrukcji zamieszczonych przy każdej z nich.



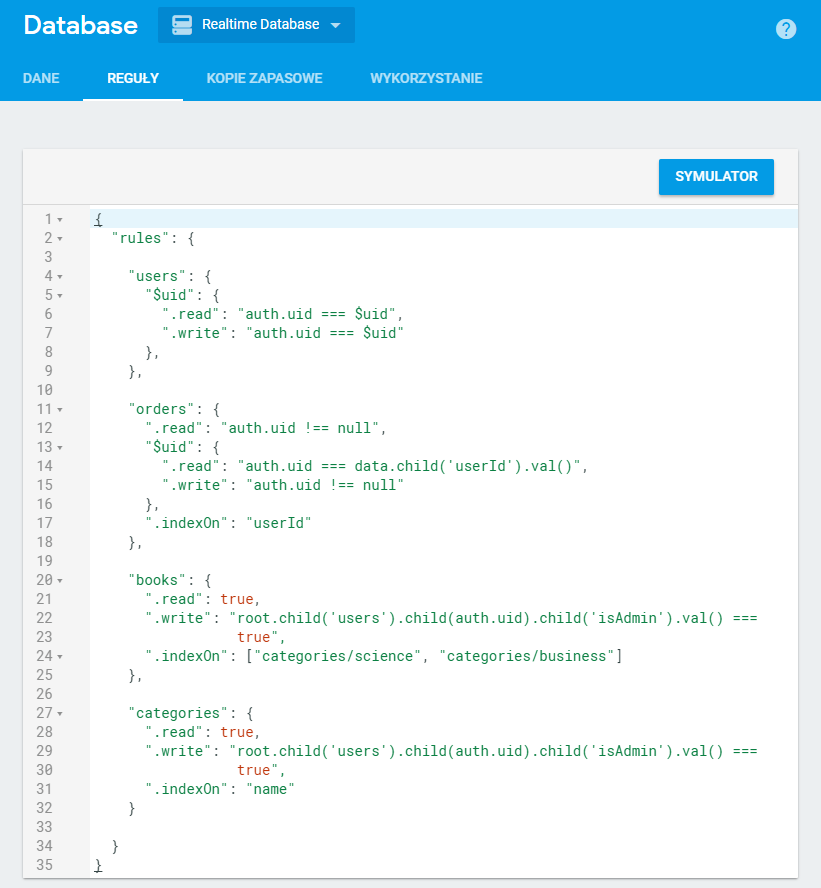
Rys. 6. Dostępne metody logowania w Firebase

*Firebase* posiada wbudowaną bazę danych czasu rzeczywistego typu *NoSQL* opartą na plikach *JSON* (ang. *real-time NoSQL JSON Database*). Oznacza to, że posiadamy nierelacyjną bazę danych, w której obiekty przechowywane są w postaci plików *JSON* [13]. Pliki te są przechowywane w chmurze i synchronizowane w czasie rzeczywistym do każdego połączonego klienta.



Rys. 7. Baza danych Firebase

Platforma umożliwia nam również zabezpieczenie bazy danych lub automatyczne tworzenie kopii zapasowej. W przypadku zabezpieczeń, należy stworzyć zbiór reguł (patrz Rys. 8), określających poziom dostępu do danej pozycji w bazie danych[[6]](#footnote-6). Rozwiązanie to jest skuteczne oraz wygodne, ze względu na to że możemy określić dostęp to każdej z pozycji indywidualnie.



Rys. 8. Reguły zabezpieczeń bazy danych Firebase

* + 1. GIT / Bitbucket

OPIS

1. Implementacja

Poniżej przedstawiono założenia projektowe wraz z opisem implementacji najważniejszych funkcjonalności systemu.

* 1. Założenia projektowe

Głównym celem projektu jest zaimplementowanie responsywnej aplikacji internetowej spełniającej podstawowe funkcjonalności sklepu internetowego:

- rejestracja oraz logowanie;

- przeglądanie produktów (dla wszystkich klientów),

- zamówienie produktu (dla zalogowanego użytkownika),

- zmiana danych użytkownika.

Aplikacja responsywna (RWD, ang. *Responsive Web Design*) jest to technika projektowania stron internetowych w sposób umożliwiający przeglądanie danej strony na urządzeniu o dowolnym rozmiarze ekranu. Oznacza to, że interfejs użytkownika jest automatycznie dostosowywany do rozmiaru ekranu na którym jest wyświetlany, niezależnie od rodzaju urządzenia wyświetlającego np. komputer, telefon czy tablet.

Wygląd aplikacji jest zaprojektowany na podstawie standardu *„Material Design”*, dzięki czemu prezencja programu jest przyjazna w odbiorze dla użytkownika. Ponadto interfejs użytkownika został zaprojektowany tak, by był w pełni przejrzysty oraz intuicyjny, przez co aplikacja jest wygodna użyciu. Komponenty aplikacji zostały zaczerpnię z oficjalnej biblioteki „*Angular Material”[[7]](#footnote-7)*.

*Material design* jest techniką projektowania aplikacji rozwijanym od 2014 r. przez firmę *Google*. Jest to zbiór zasad, zawierający podstawowe zasady wizualnego tworzenia aplikacji.

DALSZY OPIS MATERIAL DESIGN

OPIS ROZŁOŻENIA ELEMENTÓW

PROJEKT INTERFEJSU

Czego nie będzie (np. panel admina, płatności typu PayPal) (3.1.1 lub na końcu jeśli nie ma pod rozdziałów), „rozważałem płatności, są takie i takie ale wybrałem najprostsze bo..

* 1. Wykorzystywane technologie i narzędzia

W projekcie zostały wykorzystane technologie i narzędzia takie jak:

**Angular CLI** wersja 1.4.5

**TypeScript** wersja 2.6.1

**HTML 5**

**SASS**

**Firebase** wersja 3.15.2

**Webstorm** wersja 2017.2.5

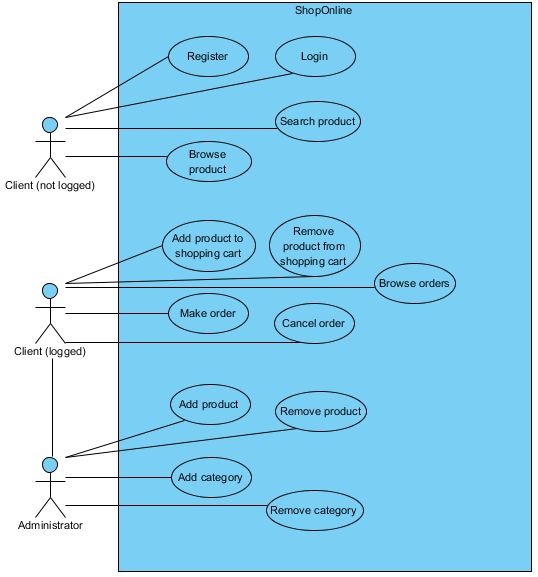
System kontroli wersji **GIT** wersja 2.14.1.windows.1

Repozytorium **Bitbucket**

Czy to opisywać?

* 1. Opis działania systemu

W poniższym podrozdziale zostaną przedstawione poszczególne funkcjonalności systemu, wraz z przykładem kodu oraz wyglądem danego elementu.



Rys. 9. Diagram przypadków użycia

* + 1. Rejestracja

Rejstracja odbywa się poprzez wypełnienie formularza rejestracji. Po podaniu poprawnych danych i zatwierdzeniu rejestracji, uzytkownik zostanie poinformowany o wysłaniu wiadomości z linkiem aktywacyjnym na podany podczas rejestracji adres e-mail (patrz Rys. 11).

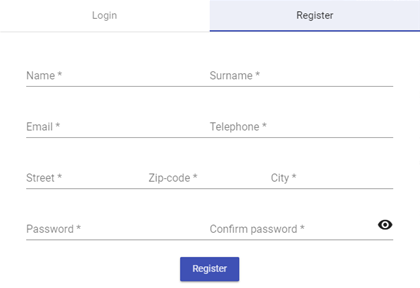
Fomularz rejestracji (patrz Rys. 10) zawiera następujące pola: imię, nazwisko, adres e-mail, numer telefonu, ulica, kod pocztowy, miasto, hasło oraz potwierdzenie hasła. Każde z pól jest odpowiednio walidowane, dlatego gdy jakiekolwiek z pól nie przejdzie walidacji, metoda odpowiedzialna za rejestracę nie zostanie wywołana, a użytkownik zostanie o tym poinformowany poprzez odpowiedni komunikat (patrz Rys. 12).

Tab. 6. Walidacja pól formularzu rejestracji

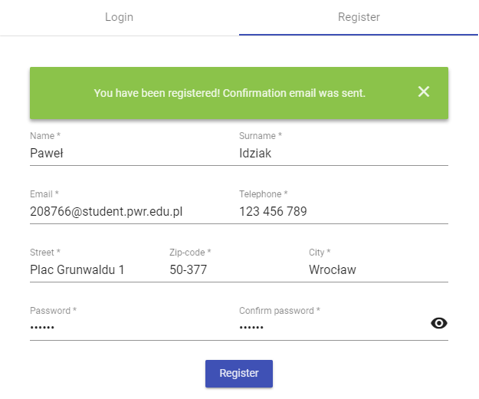
|  |  |
| --- | --- |
| **Pole** | **Walidacja** |
| **Imię, nazwisko, ulica, miasto** | Pole wymagane, mksymalna ilość znaków: 30 |
| **Email** | Pole wymagane, walidacja poprawności adresu e-mail poprzez mechanizm wbudowany w Angular[[8]](#footnote-8) (*Validators.email*). |
| **Telefon** | Pole wymagane, pole numeryczne |
| **Kod pocztowy** | Pole wymagane, sprawdzanie poprawności wg wzoru XX-YYY, gdzie X, Y to cyfry |
| **Hasło** | Pole wymagane, minimalna ilość znaków: 6, maksymalna ilość znaków: 30 |
| **Potwierdzenie hasła** | Pole wymagane, sprawdzenie zgodności wartości pola z wartością pola “Hasło” |

Listing 2. Kod odpowiedzialny za rejestrację

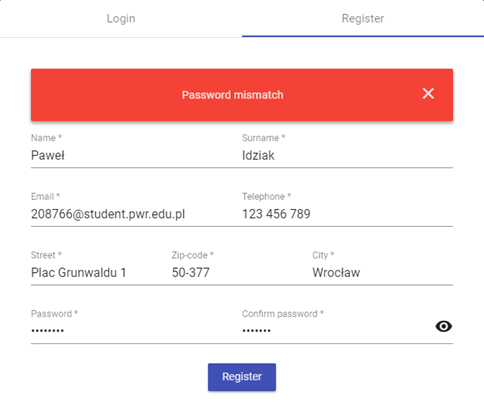
|  |
| --- |
| emailSignUp(email: string, password: string, newUser: IUser) {  return this.\_afAuth.auth.createUserWithEmailAndPassword(email,  password)  .then((user) => {  user.sendEmailVerification().catch((error: any) => {  throw new Error(error.message);  }  );  this.updateUserData(newUser, user.uid, email);  })  .catch((error: any) => {  throw new Error(error.message);  });  } |



Rys. 10. Formularz rejestracji



Rys. 11. Potwierdzenie rejestracji



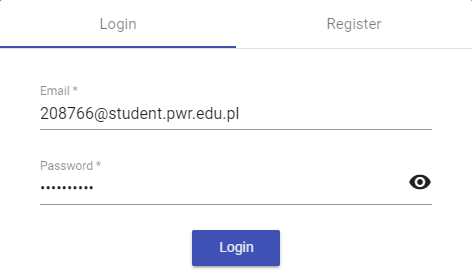
Rys. 12. Komunikat błędu rejestracji

* + 1. Logowanie

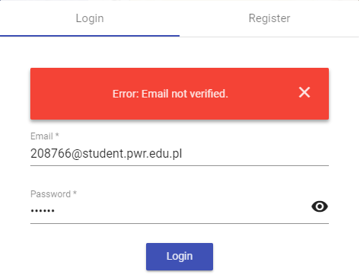
Formularz logowania zawiera dwa pola: email oraz hasło. Każde z pól jest wymagane, dlatego operacja logowania zostanie wywołana dopiero do uzupełnieniu danych. Gdy dane logowania nie są poprawne lub email przypisany do konta nie jest zweryfikowany, użytkownik zostanie o tym poinformowany poprzez wyświetlenie odpowiedniego komunikatu błędu (patrz Rys. 14). Po poprawnym zalogowaniu użytkownik uzyska dostęp do opcji przedstawionych w diagramie przypadków użycia (patrz Rys. 9).

Listing 3. Kod odpowiedzialny za logowanie

|  |
| --- |
| emailLogin(email: string, password: string) {  return this.\_afAuth.auth.signInWithEmailAndPassword(email, password)  .then((user) => {  if (user.emailVerified === false) {  throw new Error('Email not verified.');  } else {  this.user = user;  }  })  .catch((error: any) => {  throw new Error(error.message);  });  } |



Rys. 13. Formularz logowania, dane przykładowe



Rys. 14. Logowanie, błąd - niezweryfikowany email

* + 1. Przeglądanie produktów

Kod odpowiedzialny za przeglądanie produktów w dwóch wariantach + po kategoriach jest obszerny, muszę go zamieszczać?

Użytkownik ma możliwość wyboru metody przeglądania produktów: widok produktów w postaci miniatur (patrz Rys. 15) lub widok produktów w postaci listy (patrz Rys. 16). Domyślnie wybrany jest widok w postaci miniatur, zmiana następuje do kliknięcie w przycisk znajdujący się nad widokiem.

Produkty można również sortować wybierając odpowiednie pole z listy sortowania (wyświetlanie jako miniatury) lub klikając na daną kolumnę listy (wyświetlanie jako lista).

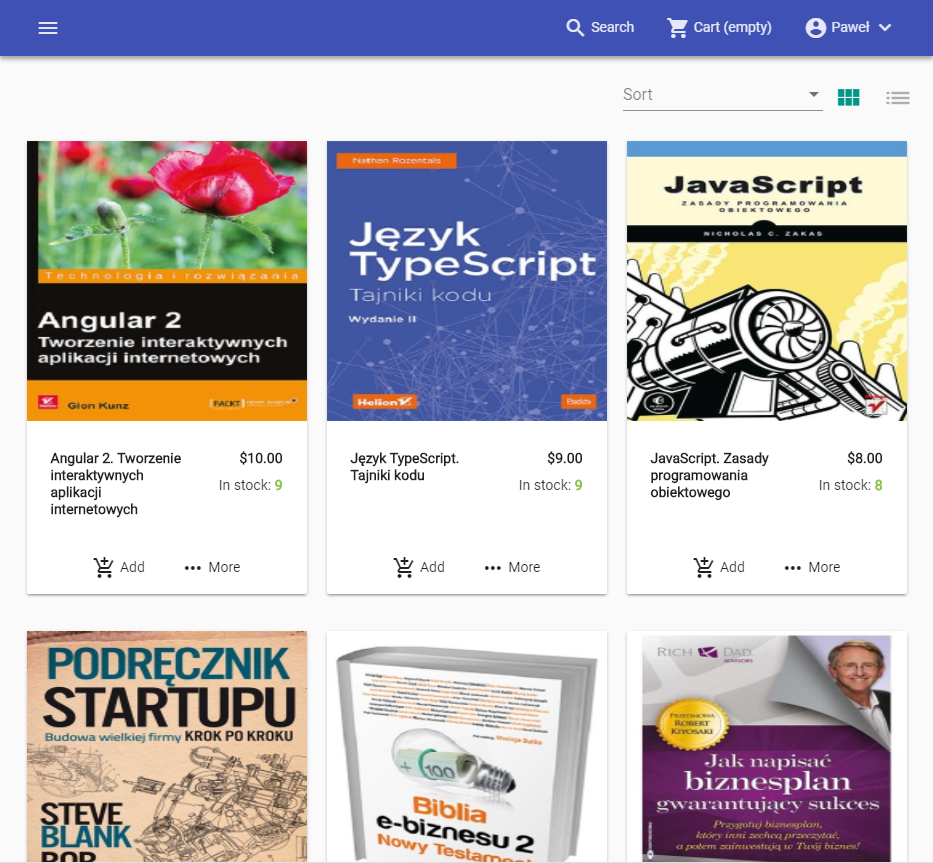
Dostępne opcje sortowania:

- cena (najmniejsza / największa),

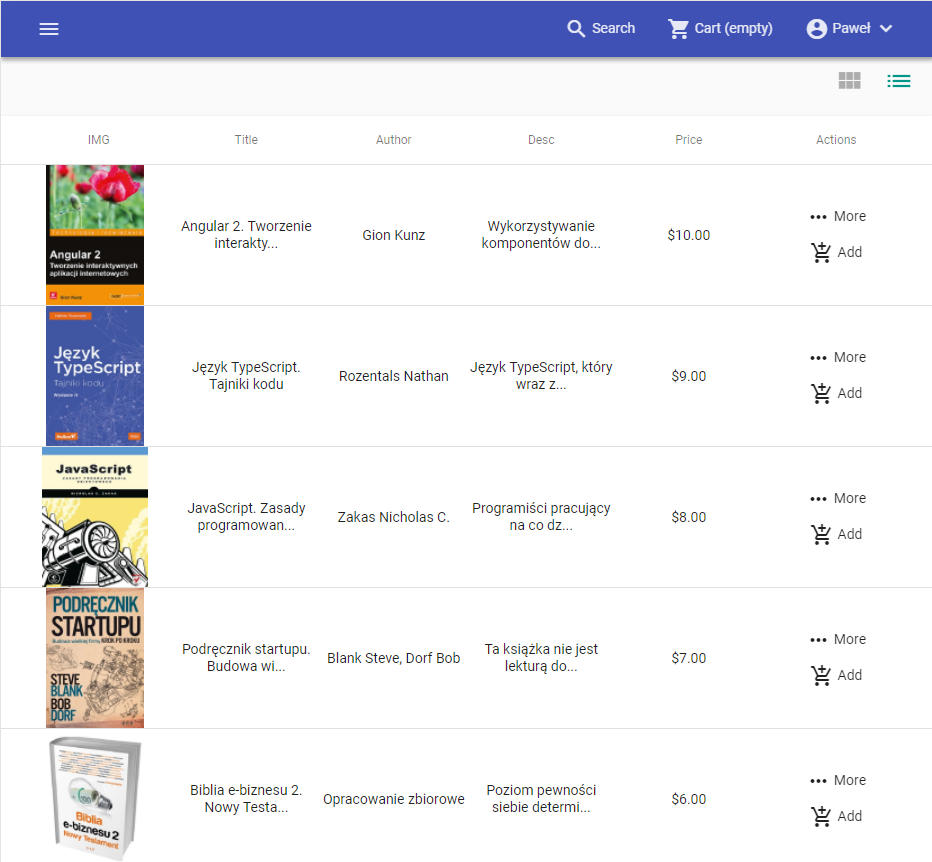
- tytuł (od A do Z / od Z do A),

- dostępność (od najmniejszej / największej liczby dostępnego produktu) – tylko dla

widoku jako miniatury



Rys. 15. Widok produktów – miniatury



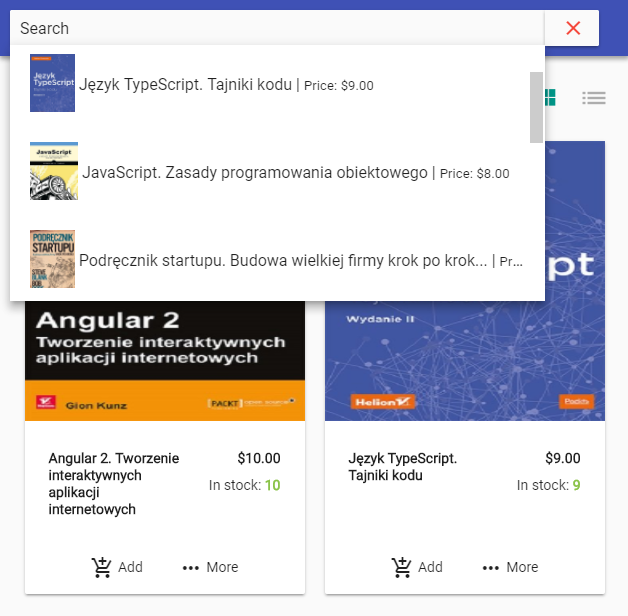
Rys. 16. Widok produktów - lista

* + 1. Wyszukiwanie produktów

Do wyszukiwania produktów został stworzony indywidualny przycisk, który znajduje się zawsze w górnym pasku strony. Dzięki temu uzytkownik z każdego miejsca aplikacji może wyszukać interesujący go produkt. Po kliknięciu na przycisk, zostaje rozwinięte pole tekstowe, w którym można wpisać tytuł produktu, wraz z podpowiedziami (patrz Rys. 17).

Listing 4. Wyszukiwanie produktów

|  |
| --- |
| getBooks() {  this.\_bookService.allBooks.subscribe(  books => {  this.options = books;  this.filteredOptions = this.myControl.valueChanges  .startWith(null)  .map(book => book && typeof book === 'object' ? book.title :  book)  .map(title => title ? this.filter(title) :  this.options.slice());  },  error => {  this.error = <any>error;  });  }    filter(name: string): IBook[] {  return this.options.filter(option =>  option.title.toLowerCase().includes(name.toLowerCase()));  }  displayFn(book: IBook): string {  return book ? book.title : '';  } |



Rys. 17. Wyszukiwanie produktów

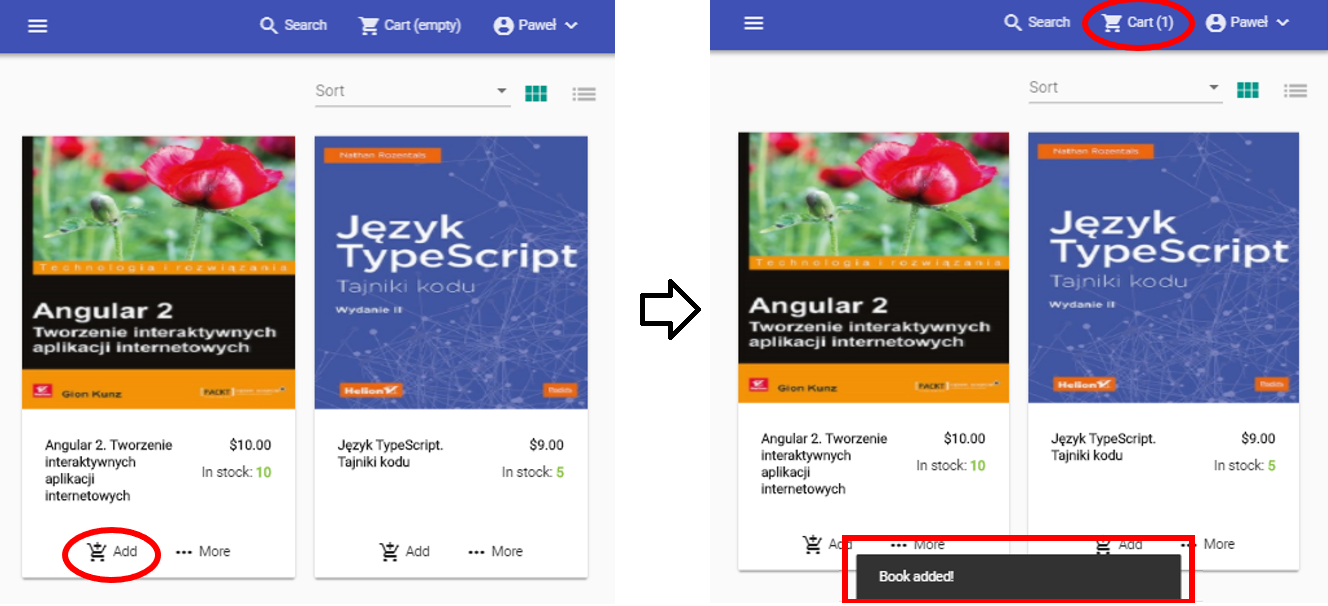
* + 1. Dodawanie produktów do koszyka

Dodawanie produktu odbywa się poprzez kliknięcie przycisku *„Add”*. Lokalizacja przycisku zależy od sposobu wyświetlania produktu. Metoda odpowiedzialna za dodanie produktu do koszyka (patrz Listing 5) sprawdza, czy dany produkt nie znajduje się w koszyku. Produkt zostaje dodany do koszyka tylko w przypadku gdy koszyk go nie zawiera. W obu przypadkach użytkownik zostanie poinformowany o rezultacie operacji (dodanie / niedodanie produktu do koszyka, patrz Rys. 18). Ilość danego produktu można zwiększyć lub zmniejszyć z poziomu widoku koszyka (patrz ).

W przypadku gdy dany produkt jest niedostępny (liczba egzemplarzy równa 0), przycisk *„Add”* zostaje automatycznie dezaktywowany, przez co produktu nie będzie można dodać do koszyka.

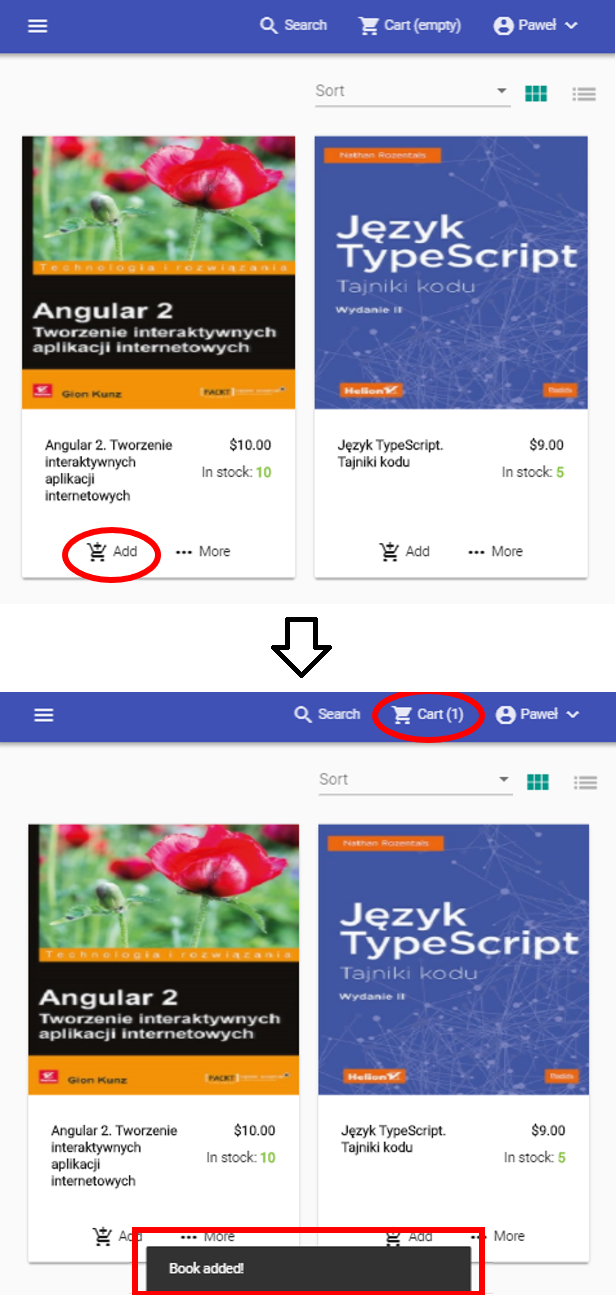
Listing 5. Dodawanie produktu do koszyka

|  |
| --- |
| public addBookToOrderList(book: IBook, quantity: number = 1) {  const foundedOrder = this.\_orders.find(x => x.book.key === book.key);  if (!foundedOrder && book.quantity > 0) {  const order: IOrder = {  book: book,  quantity: quantity  };  this.orders.push(order);  this.saveOrderListInLocalStorage();  this.calculateTotalPrice();  this.openSnackBar('Book added!');  } else {  this.openSnackBar('Book is already in shopping cart!');  }  this.checksOrders();  } |



Rys. 18. Przykład dodania produktu do koszyka

LUB TAKIE ZDJĘCIA



* + 1. Logika koszyka

Wygląd koszyka (patrz Rys. 19) jest zaimplementowany jako wysuwany z prawej strony panel (ang. *sidenav*). Przycisk odpowiedzialny za wyświetlanie (wysunięcie panelu) znajduje się na pasku górnym strony, dzięki czemu uzytkownik może wyświetlić zawartość koszyka z każdego miejsca aplikacji.

Produkty dodane do koszuka są przechowywane w formacie listy i automatycznie zapisywane (patrz Listing 6) w pamięci lokalnej przeglądarki (ang. *Local Storage*). Po uruchomieniu aplikacji zostaje wywołana metoda dopowiedzialna za sprawdzenie czy w pamięci znajduje się lista produktów, jeśli tak to jest ona odczytywana i przechowywana w aplikacji (patrz

Listing 7).

Dodatkowo, koszyk ,,nasłuchuje” na każdy z dodanych produktów i usuwa je, gdy liczba dostępnych egzemplarzy będzie równa zeru (patrz Listing 8). Dzięki temu użytkownik nie będzie mógł złożyć zamówienia na niedostępny produkt.

Cena całkowita zamówienia jest obliczana automatycznie po dodaniu produktu lub po załadowaniu listy z pamięci podręcznej przeglądarki (patrz Listing 9**Błąd! Nie można odnaleźć źródła odwołania.**) .

Każdą pozycję w koszyku zawiera miniaturę książki, jej tytuł, cenę oraz ilość egzemlarzy. Koszyk można wyczyścić, usuwając wszystkie pozycje na raz, lub każdą indywidualnie (patrz

Listing 10). Liczbę danego egzemplarza można zmienić z poziomu koszyka. Wprowadzona została dodatkowa walidacja, zmieniająca liczbę produktu na 1 w przypadku gdy uzytkownik poda liczbę mniejszą lub równą zeru, oraz zmianiająca liczbę produktu na maksymalną liczbę dostępnych egzemplarzy w przypadku gdy użytkownik poda liczbę większą od liczby dostępnych egzemplarzy (patrz Listing 11).

Gdy użytkownik sprawdzi poprawność pozycji koszyka (dany produnkt, liczba egzemplarzy, cena) może przejść do złożenia zamówienia klikając w przycisk *,,Checkout”*

Listing 6. Zapisywanie pozycji w koszyku do pamięci podręcznej przeglądarki

|  |
| --- |
| private saveOrderListInLocalStorage(): void {  const orderList = [{name: 'orders', list: this.orders}];  this.removeOrderListFromLocalStorage();  localStorage.setItem('orders', JSON.stringify(orderList));  } |

Listing 7. Odczytywanie listy produktów z pamięci podręcznej przeglądarki

|  |
| --- |
| private parseOrdersFromLocalStorage(lists: any[]) {  for (const list of lists) {  switch (list.name) {  case 'orders':  for (const order of list.list) {  const tmpOrder: IOrder = {  book: {  key: order.book.key,  title: order.book.title,  desc: order.book.desc,  author: order.book.author,  isbn: order.book.isbn,  price: order.book.price,  quantity: order.book.quantity,  image: order.book.image,  categories: order.book.categories,  releaseDate: order.book.releaseDate  },  quantity: order.quantity  };  this.orders.push(tmpOrder);  }  break;  }  }  } |

Listing 8. Sprawdzanie zawartości koszyka

|  |
| --- |
| private checksOrders() {  this.orders.forEach((order) => {  this.\_db.object('books/' +  order.book.key).valueChanges().subscribe((book: IBook) => {  if (book.quantity < order.quantity) {  this.orders = this.orders.filter(obj => obj !== order);  }  });  });  } |

Listing 9. Obliczanie ceny całkowitej koszyka

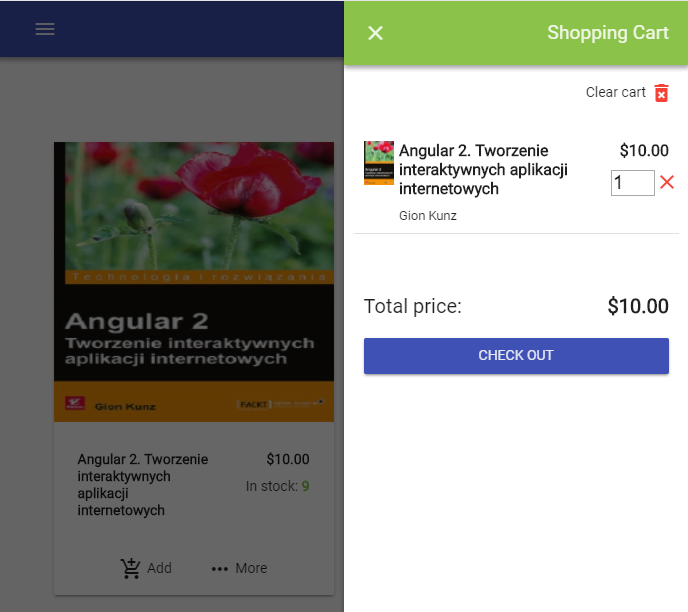
|  |
| --- |
| calculateTotalPrice() {  this.totalPrice = 0;  for (const order of this.orders) {  this.\_totalPrice += (order.book.price \* order.quantity);  }  } |

Listing 10. Usuwanie pozycji w koszyku

|  |
| --- |
| // clear all  public clearOrders() {  this.orders = [];  this.removeOrderListFromLocalStorage();  }  // remove one position  public removeOrder(order: IOrder): void {  this.\_orders = this.\_orders.filter(obj => obj !== order);  this.calculateTotalPrice();  this.saveOrderListInLocalStorage();  } |

Listing 11. Zmiana liczby egzemplarzy

|  |
| --- |
| public setQuantity() {  if (this.quantityControl.value <= 0) {  this.quantityControl.setValue('1');  }  if (this.quantityControl.value > this.order.book.quantity) {  this.quantityControl.setValue(this.order.book.quantity);  }  this.\_cartService.setNewQuantity(this.order,  this.quantityControl.value);  }  private setNewQuantity(order: IOrder, quantity: number) {  order.quantity = quantity;  this.calculateTotalPrice();  this.saveOrderListInLocalStorage();  } |



Rys. 19. Widok koszyka

* + 1. Finalizacja zamówienia

OPIS

KOD

RYS

* 1. Instrukcja instalacji

Prosty opis jak uruchomić środowisko (wraz z potrzebnymi narzędziami i ich instalacja)

Max 1str

Przed przystąpieniem do uruchomienia projektu należy zainstalować następujące narzędzia:

- środowisko uruchomieniowe Node.js w wersji …,

- menedżer pakietów npm w wersji …,

-

1. Podsumowanie

Uzyskane efekty i osiągnięcia pracy

Co udało się zrealizować, można się odnieść krytycznie (lekko) to swojego rozwiązania np. do tych płatności, do panelu administratora

Aplikacja spełnia takie i takie funkcjonalnośći, zadbano o to by kod był przejrzysty itp.

W pracy zrealizowano …

Dalsze możliwe ścieżki rozwoju pracy

Czy jest możliwość skalowalności sklepu, ograniczenia (np. przepustowac firebase)

PYTANIA:

* Opisywać jak produkty zostaly dodane do bazy?
* Opisywać skąd wziąłem informacje o książkach?

# Literatura

Adam Trachtenberg, D. S. (2007). *PHP. Receptury. Wydanie II .*

Beighley, L. (2012). *SQL. Rusz głową!*

Duckett, J. (2014). *HTML i CSS. Zaprojektuj i zbuduj witrynę WWW.* Helion.

Eric T Freeman, E. R. (2012). *HTML5. Rusz głową!* Helion.

Eric T. Freeman, E. R. (2015). *Programowanie w JavaScript. Rusz głową!* Helion.

Kunz, G. (2017). *Angular 2. Tworzenie interaktywnych aplikacji internetowych.* Helion.

Lavin, P. (2006). *Object-PHP. Programowanie obiektowePHP.* Helion.

Mariusz Chudzik, A. F. (2005). *Prawo handlu elektronicznego.* Oficyna Wydawnicza Branta.

Marrs, T. (2017). *JSON at Work. Practical Data Integration for the Web.* O'Reilly Media.

McFarland, D. S. (2013). *CSS3. Nieoficjalny podręcznik. Wydanie III.* Helion.

McFarland, D. S. (2016). *CSS. Nieoficjalny podręcznik. Wydanie IV.* Helion.

Rozentals, N. (brak daty). *Język TypeScript. Tajniki kodu. Wydanie II.*

Sullivan, D. (2016). *NoSQL. Przyjazny przewodnik.*

Literatura będzie zmieniana / aktualizowana

1. Polskie forma wsparcia PrestaShop: [http://prestashopforum.pl](http://prestashopforum.pl/) i [http://prestadesign.pl](http://prestadesign.pl/) [↑](#footnote-ref-1)
2. Wszystkie dostępne moduły znajdują się na oficjalnej stronie platformy: [www.prestashop.com/pl](http://www.prestashop.com/pl) [↑](#footnote-ref-2)
3. Informacje zaczerpnięte z serwisu [www.forbes.com](http://www.forbes.com) [↑](#footnote-ref-3)
4. Pełna lista możliwości znajduje się w oficjalnej dokumentacji: <https://cli.angular.io/> [↑](#footnote-ref-4)
5. Dokumentacja Angular CLI: <https://github.com/angular/angular-cli> [↑](#footnote-ref-5)
6. Reguły tworzenia zabezpieczeń: <https://firebase.google.com/docs/database/security/> [↑](#footnote-ref-6)
7. Komponenty standardu *Material* dla Frameworku *Angular*: <https://material.angular.io/> [↑](#footnote-ref-7)
8. Pełna lista dostępnych metod walidacji: <https://angular.io/api/forms/Validators> [↑](#footnote-ref-8)