

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA (INF)

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH (INS)

PRACA DYPLOMOWA
INŻYNIERSKA

Sklep internetowy
Shop Online

AUTOR:

Paweł Idziak

PROWADZĄCY PRACĘ:

Dr inż. Dariusz Jankowski, W4\K2

OCENA PRACY:

WROCŁAW, 2017

Spis treści

1.	Wstęp.....	4
1.1.	Rozwój rynku elektronicznego	4
1.2.	Cel pracy.....	6
1.3.	Stan wiedzy obecnej	7
1.4.	Przedstawienie zawartości poszczególnych rozdziałów pracy.....	10
2.	Użyte technologie.....	11
2.1.	Angular 4 i TypeScript	11
2.2.	HTML i CSS (SASS)	14
2.3.	Firebase.....	17
2.4.	GIT.....	20
3.	Implementacja	21
3.1.	Założenia projektowe	21
3.2.	Wykorzystywane technologie i narzędzia	24
3.3.	Opis działania systemu	25
3.3.1.	Rejestracja	25
3.3.2.	Logowanie	28
3.3.3.	Przeglądanie produktów	31
3.3.4.	Wyszukiwanie produktów	35
3.3.5.	Dodawanie produktów do koszyka	36
3.3.6.	Logika koszyka	39
3.3.7.	Finalizacja zamówienia	42
3.3.8.	Anulowanie zamówienia	48
3.3.9.	Zmiana danych użytkownika	49
3.4.	Instrukcja instalacji.....	51
4.	Podsumowanie	52

Literatura	53
Dodatek A	54

1. Wstęp

1.1. 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 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 szacowana liczba internautów wynosi 3.4 biliona, co daje 46% populacji świata, z prawdopodobieństwem wzrostu do 56% do roku 2022¹.

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.

¹ Informacje zaczerpnięte z serwisu www.forbes.com

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 udostępniane 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 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* [4] wraz z *CSS* [14] do prezentacji strony internetowej i *PHP* [1] do komunikacji strony z bazą danych. Bazy danych były w głównej mierze bazami danych typu *SQL* [2].

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* [5], *CSS3* [13] i *JavaScript* [6], 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* [18] (ang. *Non SQL*). *NoSQL* 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.

1.2. Cel pracy

Celem pracy jest zaprojektowanie oraz wykonanie aplikacji dającej możliwość kupowania 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 pkt. 1.3) oraz jest responsywna (patrz pkt. 3.1), by można korzystać z niej na każdym urządzeniu niezależnie od wielkości ekranu czy zainstalowanego systemu.

Podstawowa funkcjonalność to:

- rejestracja oraz logowanie;
- przeglądanie produktów (dla wszystkich klientów),
- zamówienie produktu (dla zalogowanego użytkownika),
- zmiana danych użytkownika.

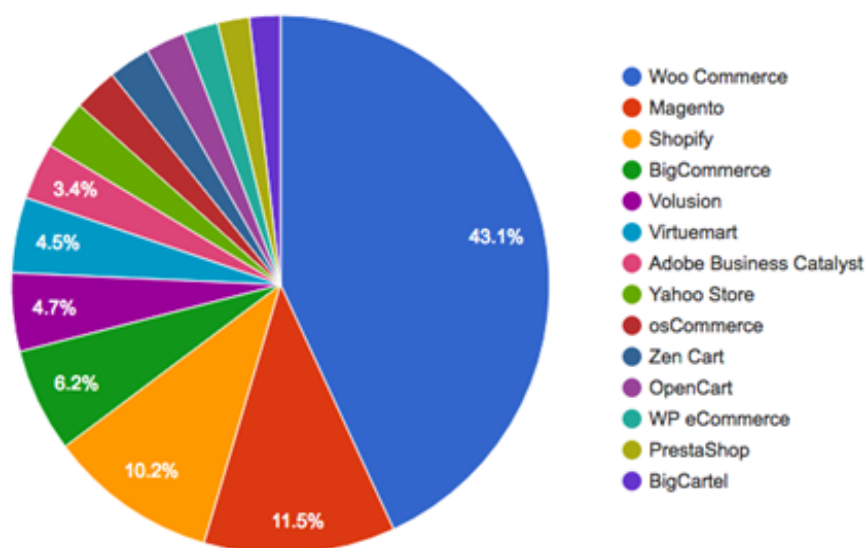
Do stworzenia aplikacji zostały użyte najnowsze technologie internetowe i bazodanowe wraz ze wsparciem systemu kontroli wersji. Każda z użytych technologii / narzędzi zostanie omówiona w dalszej części pracy (patrz pkt. 2.1).

1.3. Stan wiedzy obecnej

Handel elektroniczny [11] jest to rodzaj handlu polegający na sprzedaży towarów i usług oraz ich zakupie poprzez wykorzystanie sieci Internet jako środka 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 (patrz rys. 1). Poczynając od własnych implementacji, przez proste systemy zarządzania treścią, aż po potężne systemy używane przez duże firmy. Prowadzenie handlu elektronicznego 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 sprzętowe.



Procenty reprezentują liczbę stron internetowych zaimplementowanych w danej technologii, podzieloną przez liczbę całkowitą wszystkich stron internetowych.

Rys. 1. Statystyki technologii sklepów internetowych – sierpień 2016. Źródło: <https://commerce-forge.com/top-technologies-driving-us-ecommerce-august-2016/>

Jednym z najpopularniejszych rozwiązań umożliwiających zarządzanie sklepem internetowym jest platforma *WooCommerce* [16]. Jest to darmowy dodatek do systemu CMS (ang. *Content Management System*) WordPress².

Dodatek ten umożliwia tworzenie i zarządzanie z poziomu panelu administratora sklepem internetowym. Ze względu na swoją prostotę, *WooCommerce* 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.

System funkcjonalności moduły umożliwiające między innymi obsługę zarządzania produktami i zamówieniami, konfigurację podatków, sposobów i kosztów wysyłki czy metod płatności. Dodatkowo, administrator sklepu posiada również wgląd do raportów sprzedaży, gdzie wszystkie informacje prezentowane są w formie wykresów. Wszystkie wymienione wyżej funkcjonalności są darmowe i dostępne zaraz po instalacji. Mamy również możliwość skorzystania z dodatkowych rozszerzeń, które mogą się okazać płatne. Pełna lista rozszerzeń znajduje się na stronie: <https://woocommerce.com/>. Język używany do implementacji funkcjonalności to *PHP* [9].

Zalety platformy *WooCommerce*:

- intuicyjny interfejs,
- prosta instalacja i modyfikacja,
- duża baza rozszerzeń,
- rozbudowana społeczność,
- wersja darmowa i płatna

Wady platformy *WooCommerce*:

- większość rozszerzeń jest płatna,
- brak integracji z usługami zewnętrznymi,
- kłopotliwe zarządzanie dużą liczbą produktów

² Oficjalna strona systemu *WordPress*: <https://pl.wordpress.org/>

Kolejnym z najpopularniejszych systemów do zarządzania sklepem internetowym jest platforma Magento [10]. System ten jest bardzo rozbudowany, posiada szereg rozbudowanych funkcjonalności takich jak system analiz, raportowanie błędów, wbudowany system płatności czy tworzenie różnych szablonów zmieniających wygląd dla różnych 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 *PHP*.

Zalety platformy *Magento*:

- rozbudowane funkcjonalności,
- indywidualne szablony produktów,
- dobre wsparcie techniczne,
- możliwość zarządzania kilkoma sklepami z jednego panelu administracyjnego,
- możliwość indywidualnej rozbudowy skryptu.

Wady platformy *Magento*:

- duże wymagania serwerowe,
- skomplikowana administracja,
- platforma mało intuicyjna.

1.4. Przedstawienie zawartości poszczególnych rozdziałów pracy

Niniejsza praca inżynierska została podzielona na cztery główne rozdziały. W rozdziale pierwszym – Wstęp – zostało przedstawione wprowadzenie do tematu związanego z handlem elektronicznym. Dodatkowo uwzględniony został cel pracy wraz z przedstawieniem bieżących rozwiązań.

Rozdział drugi – Użyte technologie – zawiera opis wszystkich używanych w projekcie technologii wraz z przykładami użycia niektórych z nich.

W rozdziale trzecim – Implementacja – uwzględniony został cały proces związany z implementacją aplikacji wraz z przykładami kodu oraz efektem (w postaci rysunku) danej operacji. Uwzględnione zostały założenia projektowe oraz instrukcja instalacji. Rozdział ten przedstawia szczegóły działania takich funkcjonalności jak: rejestracja, logowanie, przeglądanie i wyszukiwanie produktów, dodawanie produktów do koszyka, logika koszyka, finalizacja oraz anulowanie zamówienia i zmiana danych użytkownika.

W rozdziale ostatnim – Podsumowanie – znajduje się opis uzyskanych efektów pracy wraz z wnioskami dotyczącymi procesu powstawania aplikacji. Zostały również uwzględnione ograniczenia aplikacji i możliwy, dalszy rozwój projektu.

2. Użyte technologie

Poniższy rozdział przedstawia opis teoretyczny użytych w projekcie technologii, wraz z wytłumaczeniem pewnych 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*.

2.1. Angular 4 i TypeScript

Angular [8] 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. Command Line Interface), dzięki którym w łatwy sposób można stworzyć nowy projekt czy nowy komponent. *Angular* umożliwia programiście implementację aplikacji internetowej, mobilnej lub komputerowej.



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 (patrz prog. 1)

Prog. 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 zostało wspomniane 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 interfejsu³. Dodatkowo CLI umożliwia nam zbudowanie aplikacji, wystartowanie jej w przeglądarce wraz z nasłuchiowaniem zmian w plikach projektu, dzięki czemu gdy dodamy i / lub zmienimy kod, *Angular* automatycznie wykryje zmianę i odświeży stronę załadowaną w przeglądarce.

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
```

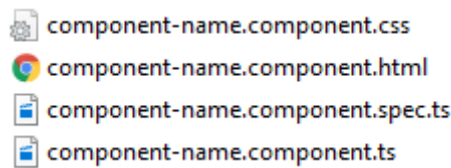
3. Tworzenie nowego projektu:

```
ng new PROJECT-NAME
```

³ Pełna lista możliwości znajduje się w oficjalnej dokumentacji: <https://cli.angular.io/>

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. 2. Zawartość folderu stworzonego komponentu

4. 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 odnośnie 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 ⁴.

⁴ Dokumentacja Angular CLI: <https://github.com/angular/angular-cli>

W przedstawianej pracy, wraz z platformą (ang. Framework) *Angular 4* został użyty język programowania *TypeScript* [17].

Typescript jest to otwarty źródłowy (ang. open source) język programowania, będący nadzbiorem (ang. super set) języka *JavaScript*. Umożliwia on programo-

wanie 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.



Tab. 1. Porównanie języka *TypeScript* i *JavaScript*

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

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

2.2. 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.



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.



Tab. 2. Przykład użycia HTML wraz z CSS

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

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. zagnieźdzonych reguł (patrz tab. 3) i pozwala przyspieszyć pracę. Programista ma możliwość wyboru typu składni, czyli SASS albo SCSS.

Tab. 3. Porównanie CSS z SCSS

CSS	SCSS
<pre>#kontener { width: 700px; } #kontener h1 { font-size: 24px; color: #fff; } #kontener h2 { font-size: 24px; color: #fff; }</pre>	<pre>#kontener { width: 700px; h1 { font-size: 24px; color: #fff; } h2 { font-size: 24px; color: #fff; } }</pre>

W języku SASS możemy również tworzyć zmienne, domieszki (ang. *mixin*) i funkcje, co znacznie ułatwia i przyspiesza pracę, a dodatkowo kod staje się re-używalny, dzięki czemu unikamy redundancji kodu. Przykłady użycia prostej domieszki oraz funkcji został zaprezentowany w tabelach 4 oraz 5.

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

Tab. 4. Przykład deklaracji domieszki i jej użycie:

Deklaracja domieszki	Użycie domieszki
<pre>@mixin mixin-name(\$some-variable) { color: \$some-variable; }</pre>	<pre>.kontener { @include mixin-name(20px); }</pre>

Tab. 5. Przykład deklaracji funkcji i jej użycie:

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

2.3. Firebase

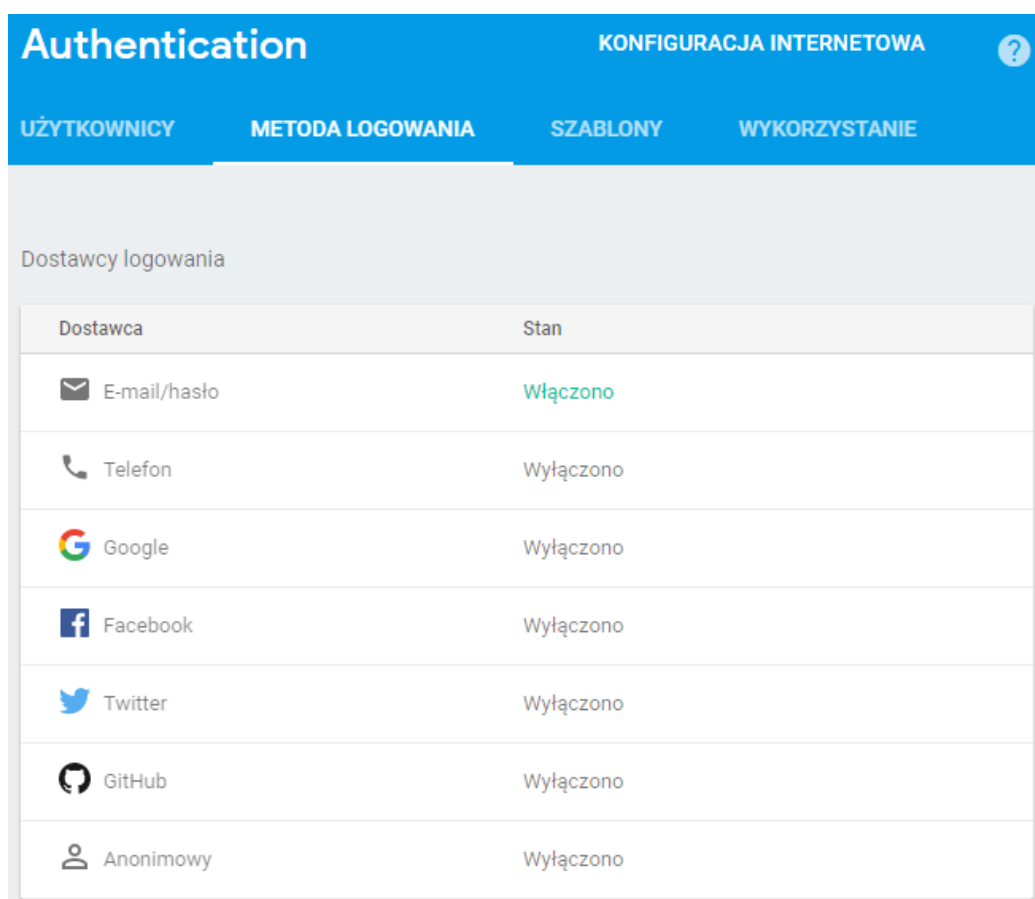
Firebase [3] jest platformą firmy Google, umożliwiającą tworzenie zaplecza aplikacji internetowych i / lub aplikacji mobilnych na systemy Android czy iOS. Platforma posiada innowacyjne podejście, oferuje programiście prostotę i oszczędność czasu. Dzięki niej nie musimy tworzyć modeli encji czy relacji pomiędzy nimi. Co więcej, platforma *Firebase* posiada wbudowany system autoryzacji i uwierzytelnienia, więc programista może skupić się na produkcji 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 (patrz rys. 3), w którym możemy zarządzać daną funkcjonalnością.



Rys. 3. Panel administracyjny Firebase

Firebase umożliwia nam wybranie kilku metod logowania (patrz rys. 4). 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.

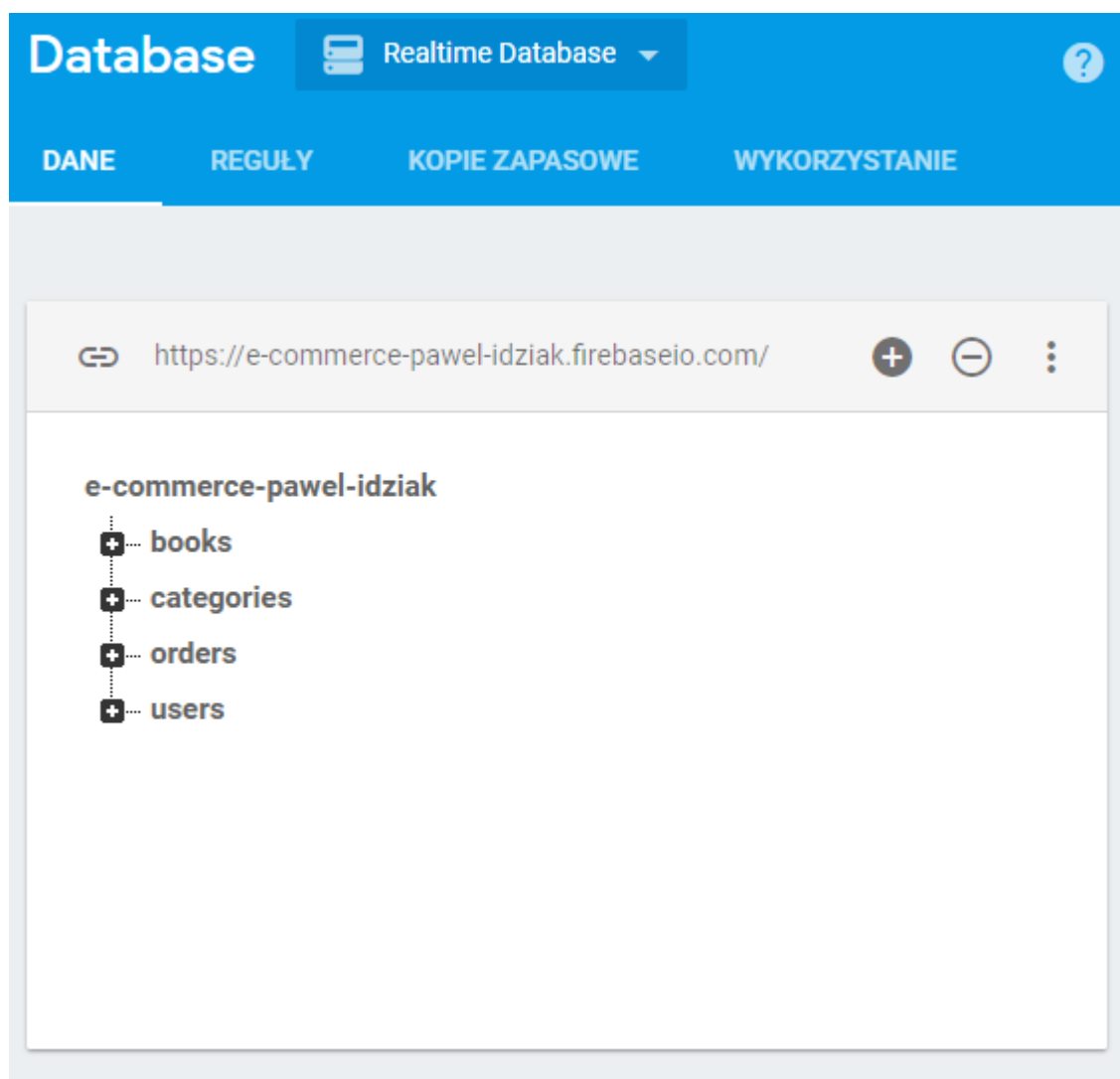


The screenshot shows the 'Authentication' section of the Firebase console, specifically the 'METHODA LOGOWANIA' (Login Methods) tab. It displays a table of login providers with their status. The 'E-mail/hasło' (Email/Password) provider is turned on, while others like 'Telefon', 'Google', 'Facebook', 'Twitter', 'GitHub', and 'Anonimowy' are turned off.

Authentication		KONFIGURACJA INTERNETOWA	?
UŻYTKOWNICY METODA LOGOWANIA SZABLONY WYKORZYSTANIE			
Dostawcy logowania			
Dostawca		Stan	
✉ E-mail/hasło		Włączono	
☎ Telefon		Wyłączono	
🌐 Google		Wyłączono	
📘 Facebook		Wyłączono	
🐦 Twitter		Wyłączono	
🐙 GitHub		Wyłączono	
👤 Anonimowy		Wyłączono	

Rys. 4. Dostępne metody logowania w Firebase

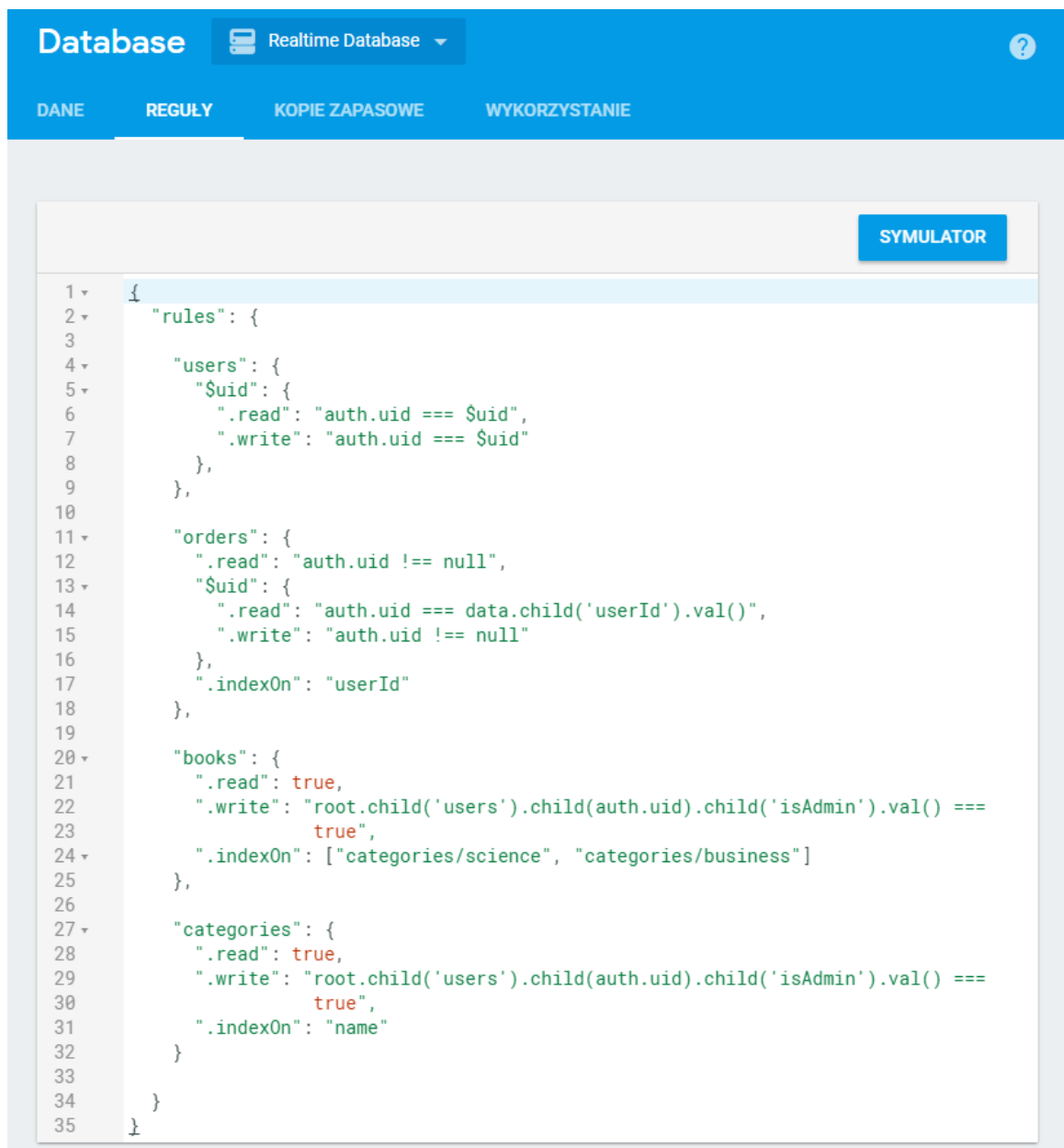
Firebase posiada wbudowaną bazę danych (patrz rys. 5) typu *NoSQL* opartą na plikach *JSON* [12]. Oznacza to, że posiadamy nierelacyjną bazę danych. Pliki w bazie danych są przechowywane w chmurze i synchronizowane w czasie rzeczywistym do każdego połączanego klienta.



Rys. 5. Baza danych Firebase

Platforma umożliwia nam również zabezpieczenie bazy danych lub automatyczne tworzenie kopii zapasowej. W przypadku zabezpieczeń, należy stworzyć zbiór reguł (patrz rys. 6), określających poziom dostępu do danej pozycji w bazie danych⁵. Rozwiązanie to jest skuteczne oraz wygodne, ze względu na to że możemy określić dostęp to każdej z pozycji indywidualnie.

⁵ Reguły tworzenia zabezpieczeń: <https://firebase.google.com/docs/database/security/>



```
1 {
2   "rules": {
3
4     "users": {
5       "$uid": {
6         ".read": "auth.uid === $uid",
7         ".write": "auth.uid === $uid"
8       },
9     },
10
11    "orders": {
12      ".read": "auth.uid !== null",
13      "$uid": {
14        ".read": "auth.uid === data.child('userId').val()",
15        ".write": "auth.uid !== null"
16      },
17      ".indexOn": "userId"
18    },
19
20    "books": {
21      ".read": true,
22      ".write": "root.child('users').child(auth.uid).child('isAdmin').val() ===
23                true",
24      ".indexOn": ["categories/science", "categories/business"]
25    },
26
27    "categories": {
28      ".read": true,
29      ".write": "root.child('users').child(auth.uid).child('isAdmin').val() ===
30                true",
31      ".indexOn": "name"
32    }
33  }
34 }
35 }
```

Rys. 6. Reguły zabezpieczeń bazy danych Firebase

2.4. GIT

GIT [7] jest to system kontroli wersji, dzięki któremu mamy możliwość monitorowania zmian w plikach projektu. Dzięki temu możemy w prosty sposób przywrócić dowolną wcześniejszą wersję naszego projektu. Narzędzie GIT jest szczególnie przydatne w projektach, w których bierze udział co najmniej kilka osób.



3. Implementacja

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

3.1. 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.

Projekt aplikacji jest responsywny (RWD, ang. *Responsive Web Design*), oznacza to że aplikacja została zaprojektowana w sposób umożliwiający przeglądanie danej strony na urządzeniu o dowolnym rozmiarze ekranu. Dzięki temu interfejs użytkownika jest automatycznie dostosowywany do rozmiaru ekranu na którym jest wyświetlany, niezależnie od rodzaju urządzenia wyświetlającego np. komputer, telefon czy tablet. Aby wprowadzić responsywność w opisywanej aplikacji, użyłem modelu pozycjonowania *Flexbox*⁶.

Aplikacja jest typu SPA (ang. *Single-Page Application*) oznacza to, że aplikacja współdziała z każdym połączonym użytkownikiem poprzez dynamiczne nadpisywanie treści bez przeładowywania strony. Dzięki takiemu rozwiązaniu strona ładowana jest tylko raz, na samym początku, a później każda odpowiedź z serwera na akcję użytkownika wykonuje się bez odświeżenia strony. Rozwiązanie to poprawia szybkość ładowania treści, dzięki temu aplikacja jest szybsza w komunikacji z użytkownikiem.

Wygląd aplikacji jest zaprojektowany na podstawie standardu „*Material Design*” [15], 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

⁶ https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes

aplikacja jest wygodna użyciu. Komponenty aplikacji zostały zaczerpnięte z oficjalnej biblioteki „*Angular Material*”⁷.

Material design jest techniką projektowania aplikacji rozwijaną od 2014 r. przez firmę *Google*. Jest to zbiór zasad, zawierający podstawowe zasady wizualnego tworzenia aplikacji. Jest to system projektowania wyglądu aplikacji, który skupia się na czytelności oraz przejrzystości widoku. Zastosowane metody są sprawdzone i popularne. W standardzie określono między innymi zasady tworzenia poszczególnych elementów, ich kolorystykę, zachowanie czy efekty takie jak cienie. Cała specyfikacja dostępna jest na oficjalnej stronie⁸.

Główny widok (patrz rys. 7) aplikacji jest podzielony na trzy główne elementy: pasek narzędzi (ang. *toolbar*), pasek boczny (ang. *sidebar*) oraz kontekst główny, w którym znajdują się informacje dotyczące danej podstrony.



Rys. 7. Widok aplikacji dla większych urządzeń

⁷ Komponenty standardu *Material* dla Frameworku *Angular*: <https://material.angular.io/>

⁸ Oficjalna strona *Material Design*: <https://material.io/>

Aplikacja jest responsywna, dlatego widok dla mniejszych urządzeń zmienia się poprzez ukrycie paska bocznego (patrz rys. 8), a przycisk odpowiadający wysunięciu paska bocznego pojawia się na pasku narzędzi. Pasek narzędzi znajduje się zawsze na górze aplikacji, niezależnie od miejsca gdzie znajduje się użytkownik.



Rys. 8. Widok aplikacji dla mniejszych urządzeń

W opisywanej aplikacji będzie znajdował się jedynie prosty panel administratora, który zawiera dwa przyciski odpowiadające dodaniu kategorii, oraz książek do bazy danych. Rozważyłem również wprowadzenie różnych metod płatności np. *PayPal*⁹, lecz ze względu na złożoność zagadnienia wybrałem najprostszy sposób płatności, czyli wyświetlenie użytkownikowi danych konta bankowego, wraz z unikalnym tytułem przelewu.

System jest elastyczny i można wprowadzić różnorodność w typach sprzedawanych produktów, lecz w niniejszej pracy ograniczyłem się do sprzedaży książek. Dodanie książek i kategorii odbywa się za pomocą prostego skryptu (patrz prog. 2).

⁹ Więcej informacji odnośnie implementacji PayPal: <https://developer.paypal.com/docs/>

Prog. 2. Dodawanie kategorii i książek do bazy danych

```
for (const book of books) {  
    this.db.list('books').push(book);  
}  
for (const category of categories) {  
    this.db.list('categories').push(category);  
}  
}
```

Gdzie *books* oraz *categories* to listy zawierające odpowiednio pozycje książek oraz kategorii. Dzięki nierelacyjnej bazie danych pozycje listy mogą być dowolnego typu, dzięki czemu system wyboru produktów jest elastyczny.

3.2. 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.

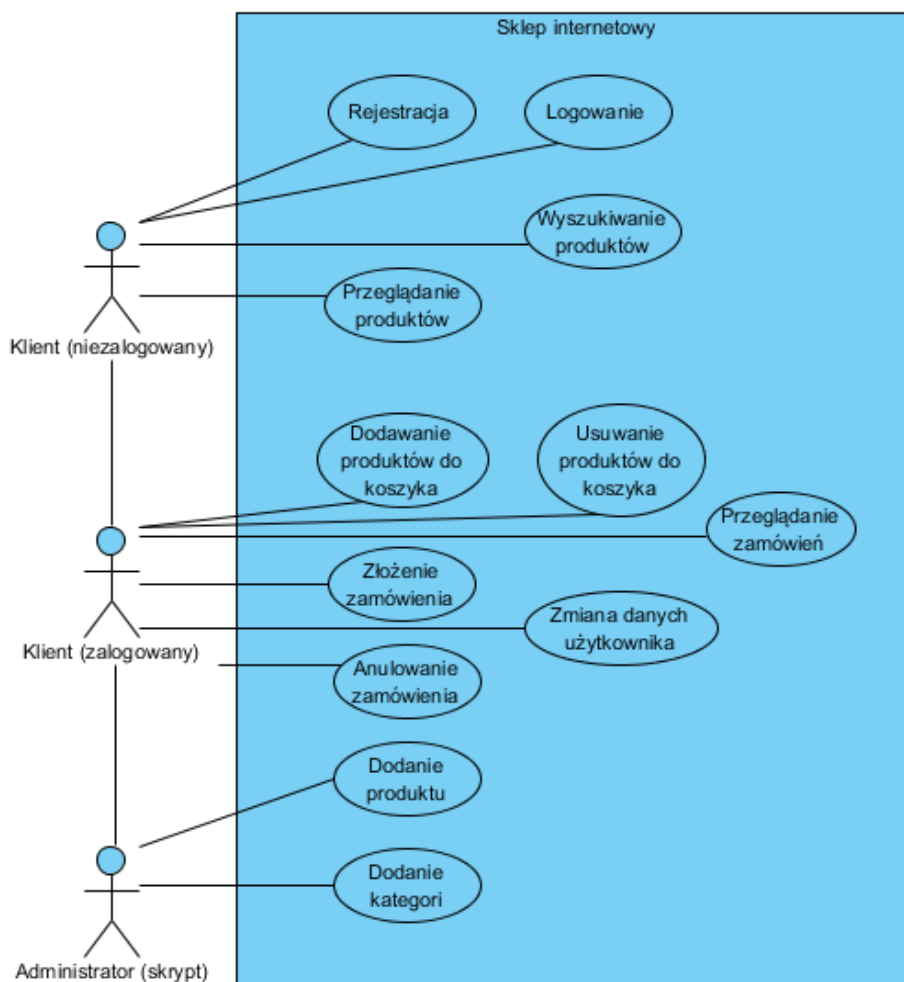
System kontroli wersji **GIT** wersja 2.14.1.windows.1.

Repozytorium **Bitbucket.**

Wszystkie wymienione wyżej technologie i narzędzia są darmowe z wyjątkiem narzędzia Webstorm. Edytor ten jest dostępny w wersji darmowej na 30 dni. Na potrzeby projektu została wykorzystana licencja studencka, która umożliwia pobranie pełnej wersji programu.

3.3. Opis działania systemu

W poniższym podrozdziale zostaną przedstawione poszczególne funkcjonalności systemu, wraz z przykładem kodu oraz wyglądem danego elementu. Na rysunku 9 zaprezentowano diagram przypadków użycia realizowanych przez aplikację.



Rys. 9. Diagram przypadków użycia

3.3.1. Rejestracja

Rejestracja odbywa się poprzez wypełnienie formularza rejestracji (patrz rys. 10) i wywołania odpowiedniej metody (patrz prog. 3).

Prog. 3. 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);
  });
}

```

The image shows a registration form with a light gray background. At the top, there are two tabs: 'Login' and 'Register'. The 'Register' tab is selected and highlighted in light blue. Below the tabs, there are several input fields arranged in rows. The first row contains 'Name *' and 'Surname *'. The second row contains 'Email *' and 'Telephone *'. The third row contains 'Street *', 'Zip-code *', and 'City *'. The fourth row contains 'Password *' and 'Confirm password *'. To the right of the 'Confirm password *' field, there is an eye icon. At the bottom center, there is a blue button with the text 'Register'.

Rys. 10. Formularz rejestracji

Fomularz rejestracji zawiera następujące pola: imię, nazwisko, adres e-mail, numer telefonu, ulica, kod pocztowy, miasto, hasło oraz potwierdzenie hasła. Każde z pól jest odpowiednio walidowane (patrz tab. 6), dlatego gdy jakiegolwiek z pól nie przejdzie walidacji, metoda odpowiedzialna za rejestrację nie zostanie wywołana, a użytkownik zostanie o tym poinformowany poprzez odpowiedni komunikat (patrz rys. 11).

Tab. 6. Walidacja pól formularzu rejestracji


Pole	Walidacja
Imię, nazwisko, ulica, miasto	Pole wymagane, maksymalna ilość znaków: 30
Email	Pole wymagane, walidacja poprawności adresu e-mail poprzez mechanizm wbudowany w Angular ¹⁰ (<i>Validators.email</i>).
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”

Login

Register

✕

Password mismatch

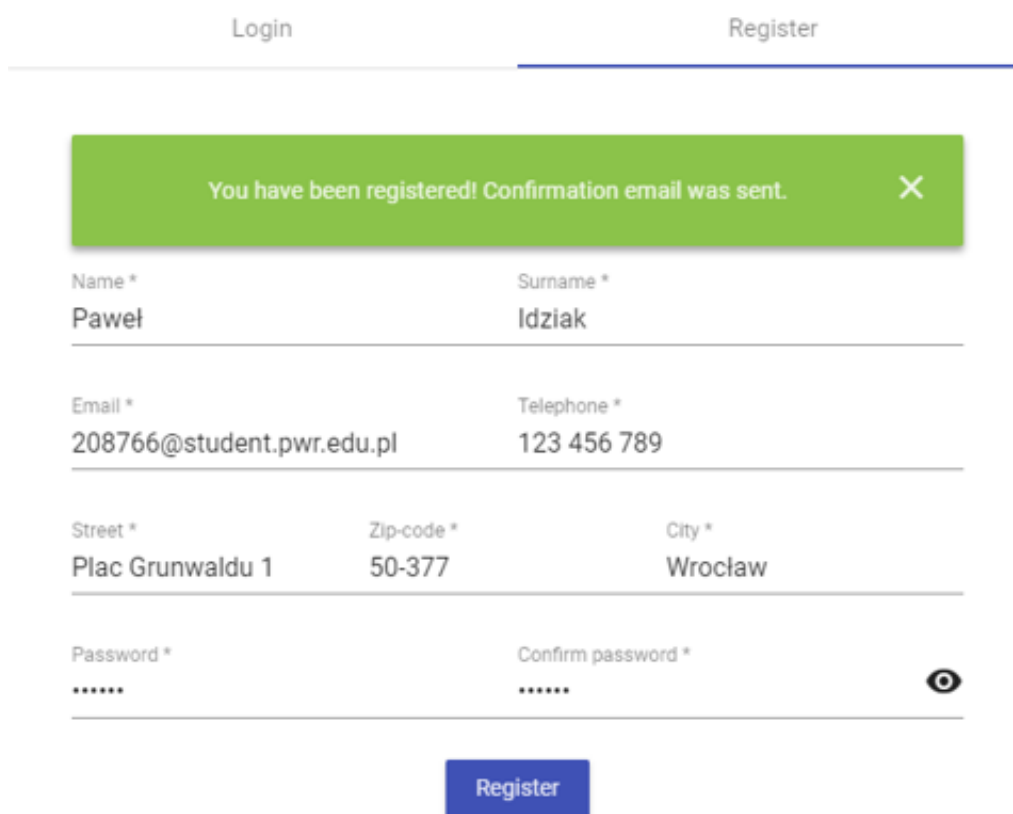
Name *	Surname *	
Paweł	Idziak	
Email *	Telephone *	
208766@student.pwr.edu.pl	123 456 789	
Street *	Zip-code *	City *
Plac Grunwaldu 1	50-377	Wrocław
Password *	Confirm password *	
*****	*****	

Register

Rys. 11. Komunikat błędu rejestracji

¹⁰ Pełna lista dostępnych metod walidacji: <https://angular.io/api/forms/Validators>

Po podaniu poprawnych danych i zatwierdzeniu rejestracji, użytkownik zostanie poinformowany o wysłaniu wiadomości z linkiem aktywacyjnym na podany podczas rejestracji adres e-mail (patrz rys. 12).



The image shows a web form for registration confirmation. At the top, there are two tabs: 'Login' and 'Register', with 'Register' being the active tab. Below the tabs is a green success message box that says 'You have been registered! Confirmation email was sent.' with a close button (X). The form fields are organized into four rows. The first row contains 'Name *' with the value 'Paweł' and 'Surname *' with the value 'Idziak'. The second row contains 'Email *' with the value '208766@student.pwr.edu.pl' and 'Telephone *' with the value '123 456 789'. The third row contains 'Street *' with the value 'Plac Grunwaldu 1', 'Zip-code *' with the value '50-377', and 'City *' with the value 'Wrocław'. The fourth row contains 'Password *' with masked characters '*****' and 'Confirm password *' with masked characters '*****'. There is an eye icon to the right of the password fields. At the bottom of the form is a blue 'Register' button.

Login		Register	
You have been registered! Confirmation email was sent. X			
Name *	Surname *		
Paweł	Idziak		
Email *	Telephone *		
208766@student.pwr.edu.pl	123 456 789		
Street *	Zip-code *	City *	
Plac Grunwaldu 1	50-377	Wrocław	
Password *	Confirm password *		👁
*****	*****		
Register			

Rys. 12. Potwierdzenie rejestracji

3.3.2. Logowanie

Formularz logowania (patrz rys. 13) zawiera dwa pola: email oraz hasło. Każde z pól jest wymagane, dlatego operacja logowania zostanie wywołana dopiero do uzupełnienia danych.

The image shows a web interface for a login system. At the top, there are two tabs: 'Login' (which is active) and 'Register'. Below the tabs, there are two input fields. The first is labeled 'Email *' and contains the text '208766@student.pwr.edu.pl'. The second is labeled 'Password *' and contains a series of dots to mask the password. To the right of the password field is an eye icon for toggling visibility. At the bottom of the form is a blue button labeled 'Login'.

Rys. 13. Formularz logowania, dane przykładowe

Po kliknięciu w przycisk „*Login*” zostaje wywołana metoda z prog. 4. 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).

Prog. 4. 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);  
    });  
}
```

The screenshot shows a web interface with two tabs at the top: 'Login' (active) and 'Register'. Below the tabs is a red error banner that reads 'Error: Email not verified.' with a close button (X). Underneath the banner are two input fields: 'Email *' containing '208766@student.pwr.edu.pl' and 'Password *' containing six asterisks. To the right of the password field is an eye icon for toggling visibility. At the bottom center is a blue 'Login' button.

Rys. 14. Logowanie, błąd - niezweryfikowany email

Za utrzymanie sesji odpowiedzialny jest mechanizm z platformy Firebase przedstawiony w prog. 5. Token sesji utrzymywany jest w pamięci podręcznej przeglądarki (ang. *Local Storage*), dlatego autoryzacja utrzymana jest w sytuacji gdy zalogowany użytkownik wróci do aplikacji po zamknięciu przeglądarki. Za wylogowywanie odpowiada metoda przedstawiona w prog. 6.

Prog. 5. Utrzymanie sesji

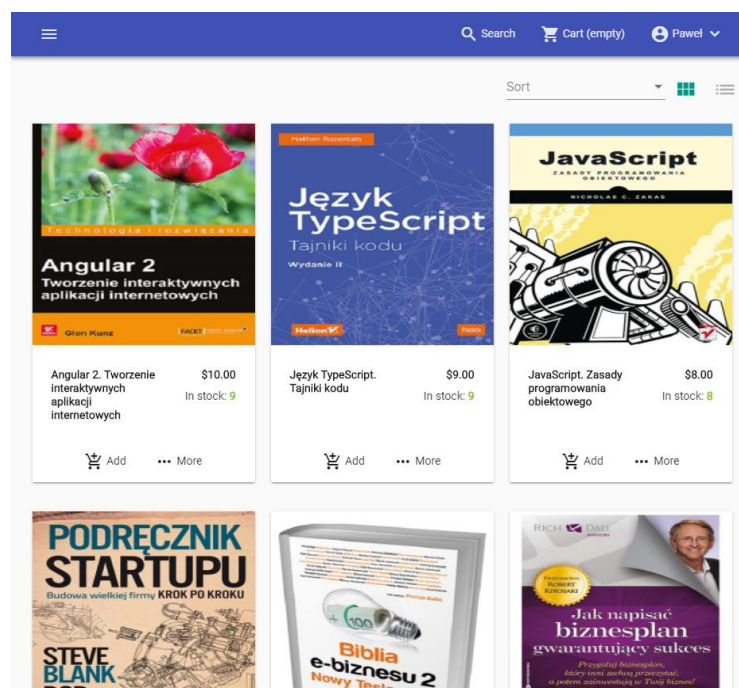
```
constructor(private _afAuth: AngularFireAuth, private _db:
AngularFirestore) {
  _afAuth.authState.subscribe((auth) => {
    this.user = auth;
    if (this.user) {
      localStorage.setItem('uid', this.user.uid);
    }
  });
}
```

Prog. 6. Wylogowanie

```
signOut(): void {
  this._afAuth.auth.signOut();
}
```

3.3.3. Przeglądanie produktów

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 po kliknięciu w przycisk znajdujący się nad widokiem.








Rys. 15. Widok produktów – miniatury

Pobieranie wszystkich produktów przedstawia prog. 7, natomiast pobieranie produktów z danej kategorii zostało zaprezentowane w prog 8. W każdym z przypadków, metoda łączy się z bazą danych i pobiera odpowiednie pola.

Prog. 7. Pobieranie wszystkich produktów

```
getAllBooks(): Observable<any[]> {  
    return this._db.list('books').snapshotChanges().map(changes => {  
        return changes.map(c => ({key: c.payload.key, ...c.payload.val()}));  
    });  
}
```

<div> <div></div> <div>Search</div> <div>Cart (empty)</div> <div>Paweł</div> </div>					
IMG	Title	Author	Desc	Price	Actions
	Angular 2. Tworzenie interakty...	Gion Kunz	Wykorzystywanie komponentów do...	\$10.00	<div>More</div> <div>Add</div>
	Język TypeScript. Tajniki kodu	Rozentals Nathan	Język TypeScript, który wraz z...	\$9.00	<div>More</div> <div>Add</div>
	JavaScript. Zasady programowan...	Zakas Nicholas C.	Programiści pracujący na co dz...	\$8.00	<div>More</div> <div>Add</div>
	Podręcznik startupu. Budowa wi...	Blank Steve, Dorf Bob	Ta książka nie jest lekturą do...	\$7.00	<div>More</div> <div>Add</div>
	Biblia e-biznesu 2. Nowy Testa...	Opracowanie zbiorowe	Poziom pewności siebie determi...	\$6.00	<div>More</div> <div>Add</div>

Rys. 16. Widok produktów - lista

Prog. 8. Pobieranie produktów z danej kategorii

```

getBooksByCategory(category: string | null): Observable<any[]> {
  return this._db.list('books', ref => category ?
    ref.orderByChild('categories/' + category).equalTo(true) : ref)
    .snapshotChanges().map(changes => {
      return changes.map(c => ({key: c.payload.key, ...c.payload.val()}));
    });
}

```

Produkty można również sortować (patrz prog. 9) 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

Prog. 9. Sortowanie produktów

```
sortBooks(array: Array<IBook>, sortType: String): Array<IBook> {
    switch (sortType) {
        case 'price-lowest':
            array.sort((a, b) => {
                return a.price < b.price ? -1 : 1;
            });
            break;
        case 'price-highest':
            array.sort((a, b) => {
                return a.price > b.price ? -1 : 1;
            });
            break;
        case 'title-a-z':
            array.sort((a, b) => {
                return a.title < b.title ? -1 : 1;
            });
            break;
        case 'title-z-a':
            array.sort((a, b) => {
                return a.title > b.title ? -1 : 1;
            });
            break;
        case 'quantity-lowest':
            array.sort((a, b) => {
                return a.quantity < b.quantity ? -1 : 1;
            });
            break;
        case 'quantity-highest':
            array.sort((a, b) => {
                return a.quantity > b.quantity ? -1 : 1;
            });
            break;
    }
    return array;
}
```

Każdy produkt z osobna posiada swoje szczegóły, które można ujrzeć po kliknięciu w przycisk „More”. Użytkownik zostanie przekierowany do podstrony ze szczegółami (patrz rys. 17) posiadającej takie informacje jak: miniatura książki, tytuł, autor, data wydania książki, cena oraz opis szczegółowy.

Search

Cart (1)

Paweł

Nathan Rozentals

Język TypeScript

Tajniki kodu

Wydanie II

Helion

Books

Język TypeScript. Tajniki kodu

Author: Rozentals Nathan

Release date: 11/24/2017 (M/D/Y)

Description:

Język TypeScript, który wraz z kompilatorem i zestawem narzędzi jest udostępniany na zasadach open source, zyskuje ogromne uznanie tysięcy projektantów aplikacji. TypeScript pozwala na pracę w zgodzie ze standardami języka JavaScript (ES5, ES6 i ES7), co pozwala programistom na używanie klas, interfejsów, typów ogólnych itd. Okazuje się, że TypeScript umożliwia tworzenie solidnych aplikacji przy wykorzystaniu technik obiektowych – i są to nie tylko aplikacje WWW, lecz także aplikacje serwerowe, aplikacje dla urządzeń mobilnych, a nawet oprogramowanie do sterowania urządzeniami w internecie rzeczy (IoT).

Price: \$9.00

In stock: 9

Quantity:

Total price: \$9.00

ADD

CHECKOUT

Rys. 17. Szczegóły produktu

Metoda odpowiedzialna za pobieranie z bazy danych pojedynczego produktu została przedstawiona w prog. 10. Z tego miejsca aplikacji można również dodać dany produkt wpisując liczbę egzemplarzy i zatwierdzając klikając w przycisk „Add”, i / lub przejść do realizacji zamówienia – klikając w przycisk „Checkout”.

Prog. 10. Pobieranie pojedynczego produktu z bazy danych

```

getBookByKey(key: string): Observable<IBook> {
    return this._db.object('books/' + key).valueChanges();
}

```

3.3.4. Wyszukiwanie produktów

Do wyszukiwania produktów został stworzony indywidualny przycisk, który znajduje się zawsze w górnym pasku strony. Dzięki temu użytkownik 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. 18) oraz automatycznie zostaje wywoływana metoda zaprezentowana w prog. 11.

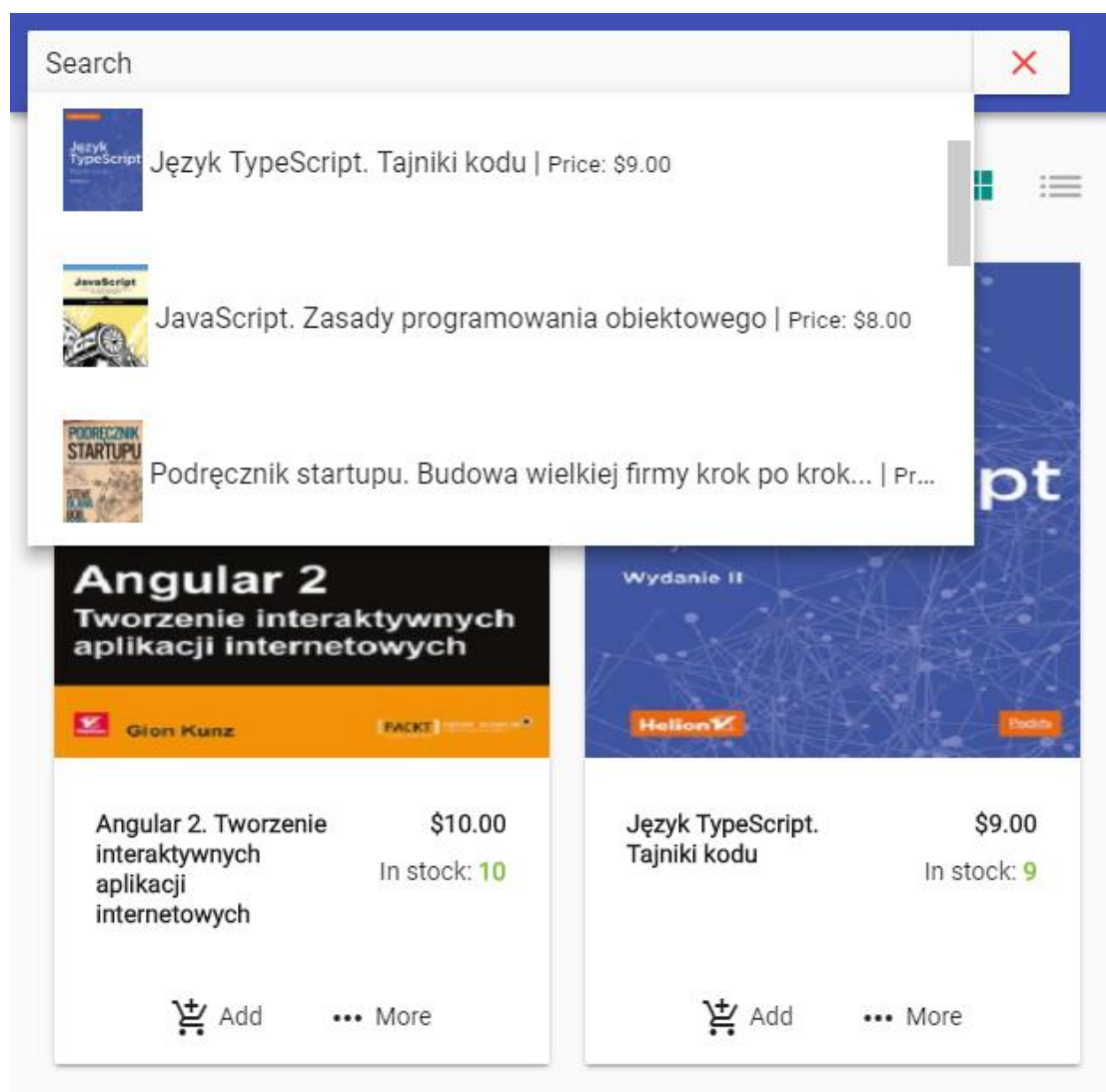
Prog. 11. Wyszukiwanie produktów

```
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 : '';
}
```



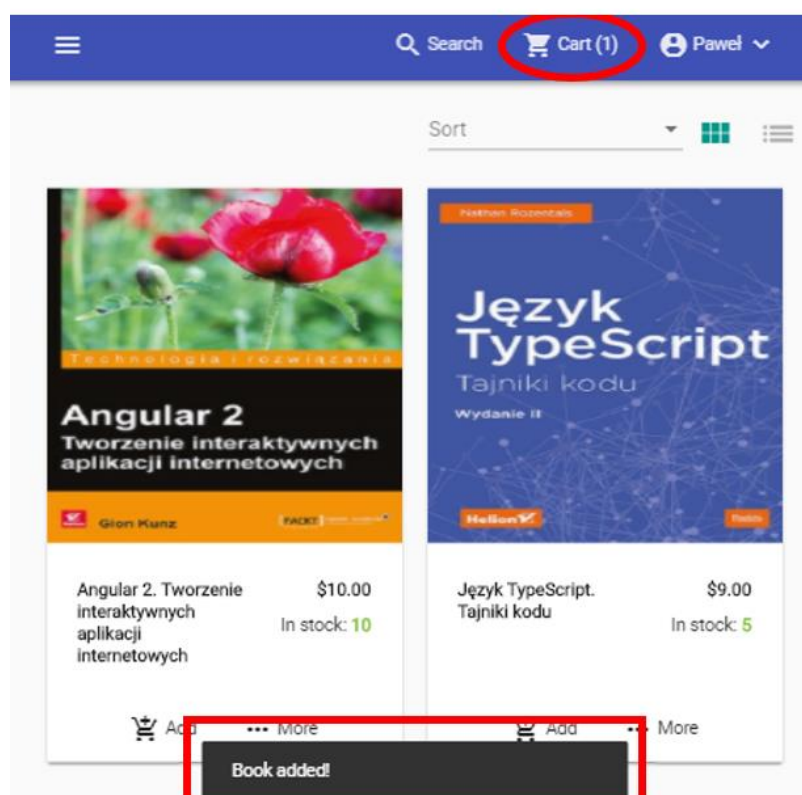
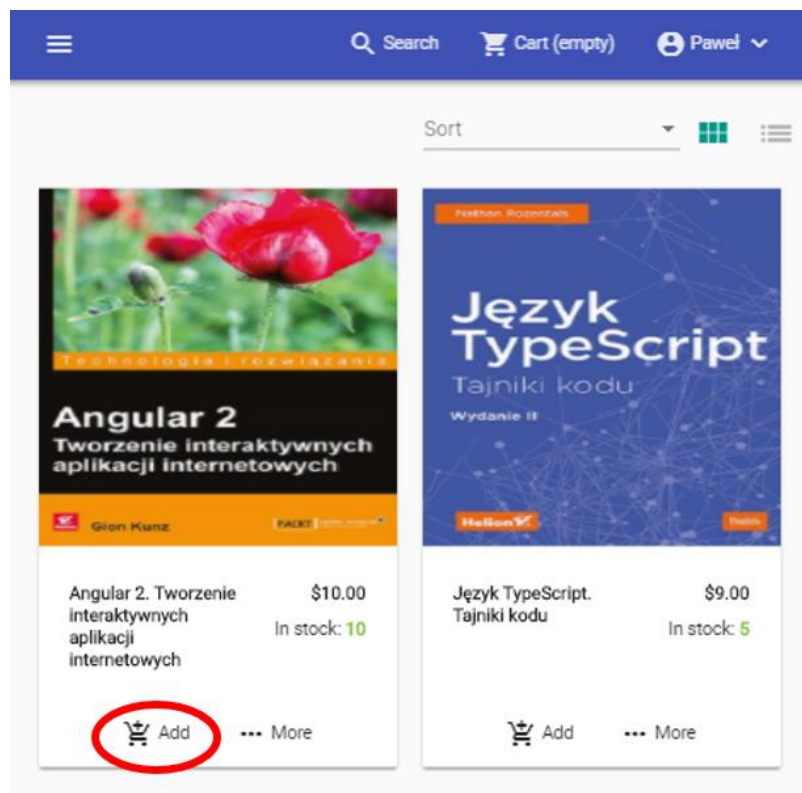
Rys. 18. Wyszukiwanie produktów

3.3.5. Dodawanie produktów do koszyka

Dodawanie produktu odbywa się poprzez kliknięcie przycisku „Add”. Lokalizacja przycisku zależy od sposobu wyświetlania produktu, (patrz pkt. 3.3.3). Metoda odpowiedzialna za dodanie produktu do koszyka (patrz prog. 12) sprawdza, czy dany produkt nie znajduje się w koszyku. Produkt zostaje dodany do koszyka tylko w przypadku gdy koszyk go nie zawiera. W obu sytuacjach użytkownik zostanie poinformowany o rezultacie operacji (dodanie / niedodanie produktu do koszyka (patrz rys. 19). Liczbę egzemplarzy danego produktu można zwiększyć lub zmniejszyć z poziomu widoku koszyka (patrz rys. 20). W przypadku gdy dany produkt jest niedostępny (liczba egzemplarzy równa jest zero), przycisk „Add” zostaje automatycznie dezaktywowany, przez użytkownik nie ma możliwości dodania produktu do koszyka.

Prog. 12. Dodawanie produktu do koszyka

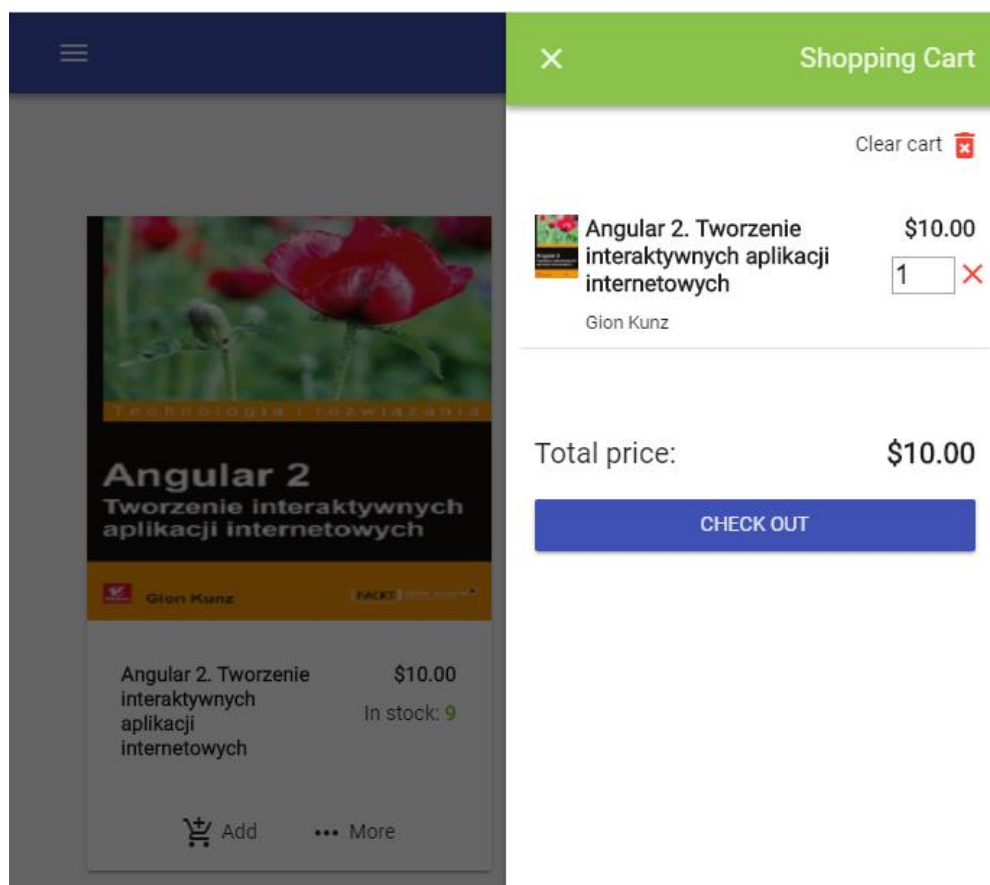
```
public addBookToOrderList(book: IBook, quantity: number = 1) {
  const foundedOrder = this._orders.find(x => x.book.key === book.key);
  if (!foundedOrder && book.quantity > 0) {
    const order: IOrder = {
      book: book,
      quantity: quantity
    };
    this.orders.push(order);
    this.saveOrderListInLocalStorage();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



Rys. 19. Przykład dodania produktu do koszyka

3.3.6. Logika koszyka

Wygląd koszyka (patrz rys. 20) jest zaimplementowany jako wysuwany z prawej strony panel (ang. *sidebar*). Każda pozycję w koszyku zawiera miniaturę książki, jej tytuł, cenę oraz ilość egzemplarzy. Przycisk odpowiedzialny za wyświetlanie (wysunięcie panelu) znajduje się na pasku górnym strony, dzięki czemu użytkownik może wyświetlić zawartość koszyka z każdego miejsca aplikacji.



Rys. 20. Widok koszyka

Produkty dodane do koszyka są przechowywane w formacie listy i automatycznie zapisywane (patrz prog. 13) w pamięci lokalnej przeglądarki (ang. *Local Storage*). Po uruchomieniu aplikacji zostaje wywołana metoda odpowiedzialna za sprawdzenie czy w pamięci znajduje się lista produktów, jeśli tak to jest ona odczytywana i przechowywana w aplikacji (patrz prog. 14).

Prog. 13. 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));
}
```

Prog. 14. 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;
        }
    }
}
```

Dodatkowo, koszyk „nasłuchuje” na każdy z dodanych produktów i usuwa je, gdy liczba dostępnych egzemplarzy jest równa zero (patrz prog. 15). Dzięki temu użytkownik nie będzie mógł złożyć zamówienia na niedostępny produkt.

Prog. 15. 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);
      }
    });
  });
}
```

Cena całkowita zamówienia jest obliczana automatycznie po dodaniu produktu lub po załadowaniu listy z pamięci podręcznej przeglądarki (patrz prog. 16) .

Prog. 16. Obliczanie ceny całkowitej koszyka

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

Koszyk można wyczyścić, usuwając wszystkie pozycje na raz, lub każdą indywidualnie (patrz prog. 17). Liczbę danego egzemplarza można zmienić z poziomu koszyka.

System uwzględnia również sytuację, gdy użytkownik poda liczbę mniejszą lub równą zero – system automatycznie zmieni wpisywaną liczbą na 1, lub gdy użytkownik poda liczbę większą niż liczba dostępnych egzemplarzy – system zmieni liczbę na liczbę dostępnych sztuk danego produktu (patrz prog. 18).

Gdy użytkownik sprawdzi poprawność pozycji koszyka (dany produkt, liczba egzemplarzy, cena) może przejść do finalizacji zamówienia klikając w przycisk „Checkout”.

Prog. 17. 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();
}
```

Prog. 18. 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();
}
```

3.3.7. Finalizacja zamówienia

Finalizacja zamówienia składa się z trzech kroków: sprawdzenie zamówienia (patrz rys. 21), potwierdzenie lub edycja danych do wysyłki (patrz rys. 23) oraz potwierdzenia zamówienia (patrz rys 24).

W kroku pierwszym użytkownik ma możliwość sprawdzenia, czy zamówienie jest zgodne z jego założeniem, czyli czy wyświetlone pozycje są zgodne z pozycjami wybranymi. Gdy użytkownik uzna, że wszystko się zgadza może przejść do następnego kroku. Możliwość ta jest dostępna jedynie dla zalogowanych użytkowników, w przeciwnym razie przycisk odpowiadający za przejście będzie nieaktywny, a klient zostanie powiadomiony o konieczności zalogowania (patrz rys. 22).

Search


Cart (1)

Paweł

1 Check Your order

2 Shipping address

3 Confirm

IMG	Title	Author	Price	Quantity
	Angular 2. Tworzenie interaktywnych aplikacji internetowych	Gion Kunz	\$10.00	<input type="text" value="1"/>

Total price:

\$10.00

Next

Rys. 21. Sprawdzenie zamówienia

Search


Cart (1)

Login

1 Check Your order

2 Shipping address

3 Confirm

IMG	Title	Author	Price	Quantity
	Angular 2. Tworzenie interaktywnych aplikacji internetowych	Gion Kunz	\$10.00	<input type="text" value="1"/>

Total price:

\$10.00

Sign up first

Next

Rys. 22. Adres wysyłki

W kroku następnym wyświetla się formularz z danymi przypisanymi do konta. Metoda odpowiedzialna za pobieranie informacji o kliencie z bazy danych przedstawiona jest w prog 19. Użytkownik ma możliwość edycji tych danych – produkty mogą zostać wysłane na inny adres.

The screenshot shows a web application interface for confirming an order. At the top, there is a blue navigation bar with a menu icon, a search bar, a cart icon labeled 'Cart (1)', and a user profile icon labeled 'Paweł'. Below the navigation bar, there is a progress indicator with three steps: '1 Check Your order', '2 Shipping address' (the current step), and '3 Confirm'. The 'Shipping address' form contains the following fields and values:

Field	Value
Name *	Paweł
Surname *	Idziak
Email *	208766@student.pwr.edu.pl
Telephone *	123 456 789
Street *	Plac Grunwaldu 1
Zip-code *	50-377
City *	Wrocław

At the bottom of the form, there are two blue buttons: 'Back' on the left and 'Next' on the right.

Rys. 23. Potwierdzenie zamówienia

Prog. 19. Pobieranie danych użytkownika

```
getUserFormDB(uid: string): Observable<IUser> {  
    return this._db.object('users/' + uid).valueChanges();  
}
```

Kolejnym, ostatnim krokiem jest potwierdzenie całego zamówienia, gdzie widnieją wypunktowane produkty, ich szczegóły, cena całkowita uwzględniająca koszty wszystkich pozycji oraz dane do wysyłki. Przyjęty rodzaj wysyłki to Poczta Polska, a płatność to przelew bankowy.

Search

Cart (1)


Paweł

Check Your order

Shipping address

3 Confirm

Your order

IMG	Title	Author	Price	Quantity
	Angular 2. Tworzenie interaktywnych aplikacji internetowych	Gion Kunz	\$10.00	1

Total price: \$10.00

Shipping information

Name	Surname
Paweł	Idziak
Email	
208766@student.pwr.edu.pl	
Telephone	
132 456 789	
Street	
Plac Grunwaldu 1	
Zip-code	City
50-377	Wrocław

Shipping:

Payment:

☒ Polish Post (2-3 days)

☒ Bank transfer


Back

ORDER

Rys. 24. Potwierdzenie zamówienia

Po zatwierdzeniu zamówienia (przycisk „Order”) klient zostanie przekierowany do nowej podstrony, na której widnieje podsumowanie zamówienia wraz z informacją dotyczącą płatności. W tym momencie, w bazie danych zostaje dodane nowe zamówienie automatycznie kasowana jest zawartość koszyka użytkownika (patrz rys. 25). Metoda odpowiedzialna za wyżej wymienione czynności została przedstawiona w prog. 20 .

Menu Search **Cart (empty)** Paweł

IMG	Title	Author	Price	Quantity
	Angular 2. Tworzenie interaktywnych aplikacji internetowych	Gion Kunz	\$10.00	1

Total price: **\$10.00**

Your order has been placed

Please make a transfer to the following data:

Name: ShopOnline

Account number: 74 1140 2004 0000 3802 7477 8008

Title: -L-3VK9XOrxfcSmSUJKa

Amount: \$10.00

*make sure the title is the same

Your order will be sent immediately after the payment has been posted.

If You have any question please contact us. [Contact](#)

Thank you for using our services,
ShopOnline team

[Go back](#) [See my orders](#)

Rys. 25. Podsumowanie złożonego zamówienia

Prog. 20. Złożenie zamówienia, dodanie pola w bazie danych

```
makeOrder() {
  const orderDTO: IOrderDTO = {
    userId: this._authService.currentUser.uid,
    list: this.orders,
    totalPrice: this.totalPrice,
    status: 'waiting for payment',
    orderDate: Date.now()
  };
  this._db.list('orders').push(orderDTO)
    .then((order) => {
      this._router.navigate(['/order', order.key]);
      this.clearOrders();
    });
}
```

Dodatkowo w platformie Firebase została zaimplementowana *Cloud function*¹¹ (patrz prog. 21), która po wykryciu dodania zamówienia, najpierw sprawdza czy liczba zamówionych egzemplarzy jest dostępna. Dzięki temu unikamy konfliktu, gdy kilku użytkowników jednocześnie chce kupić jeden produkt.

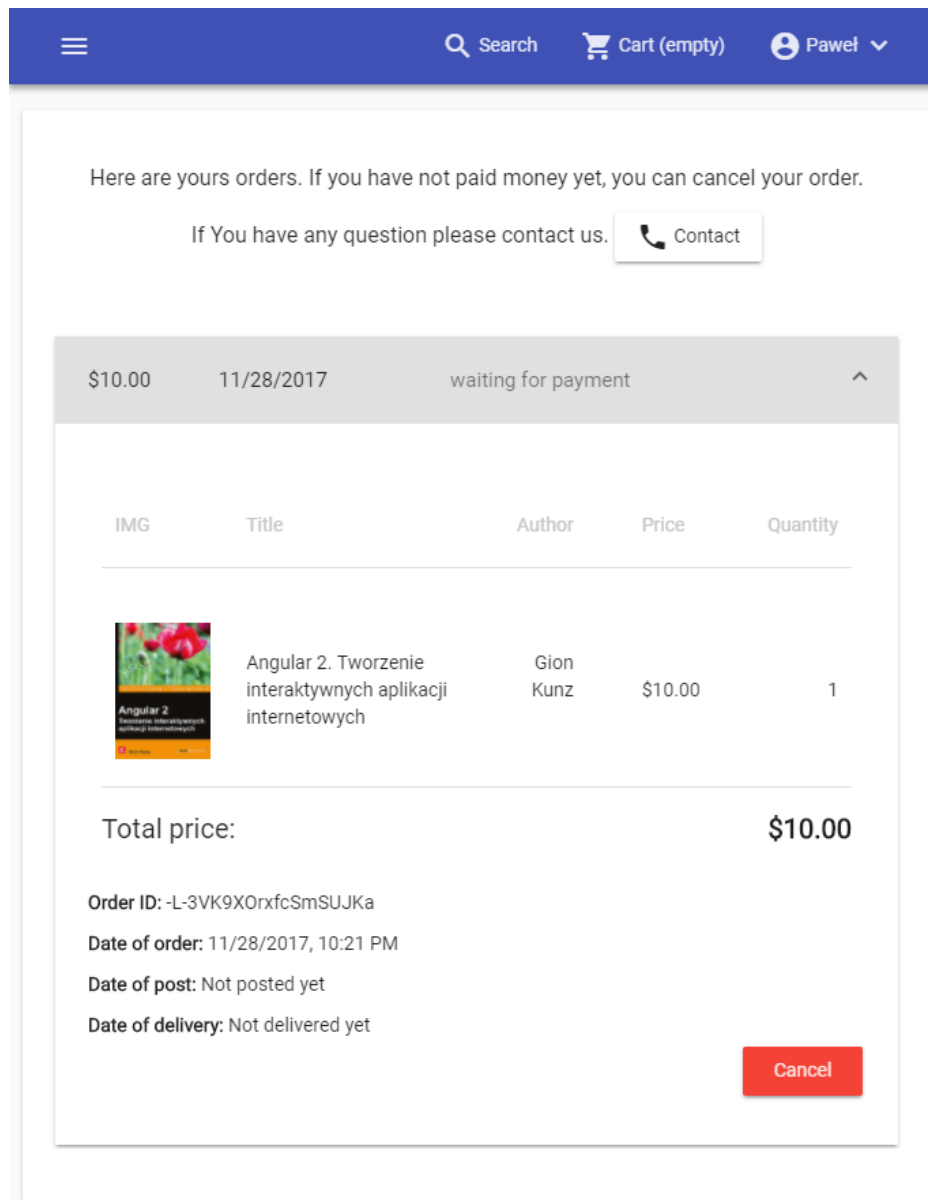
Prog. 21. Cloud Function - tworzenie zamówienia

```
exports.makeOrderAndChangeBookQuantity = functions.database
  .ref('/orders/{pushId}')
  .onWrite(event => {
    if(event.data.val() != null) {
      const listOfBooks = event.data.val().list
      const root = event.data.ref.root
      for(let i = 0; i < listOfBooks.length; i++){
        const path = '/books/' + listOfBooks[i].book.key
        root.child(path).once('value').then(snap => {
          const quantity = snap.val().quantity
          const newQuantity = quantity -
            listOfBooks[i].quantity
          if(newQuantity < 0) {
            throw new Error('Quantity error.');
```

¹¹ Więcej informacji na stronie: <https://firebase.google.com/docs/functions/>

3.3.8. Anulowanie zamówienia

Na podstronie „*Order History*” (patrz rys. 26) klient ma wgląd do historii złożonych przez niego zamówień oraz możliwość anulowania zamówienia w przypadku, gdy status zamówienia jest ustawiony na „*waiting for payment*” – czyli przed dokonaniem płatności. Dzięki temu rozwiązaniu, w razie pomyłki można w łatwy sposób anulować zamówienie nie ingerując w to pracowników sklepu.



Rys. 26. Zamówienia użytkownika.

Podczas anulowania zamówienie zostaje wywołana funkcja z poziomu platformy Firebase (Cloud Function) przedstawiona w prog. 22. Metoda ta automatycznie zwiększa liczbę egzemplarzy produktów znajdujących się w anulowanym zamówieniu.

Prog. 22. Cloud Function - anulowanie zamówienia

```
exports.deleteOrderAndChangeBookQuantity = functions.database
  .ref('/orders/{pushId}')
  .onDelete(event => {

    if(event.data.previous.val() != null) {

      const listOfBooks = event.data.previous.val().list
      const root = event.data.ref.root

      for (let i = 0; i < listOfBooks.length; i++) {

        const path = '/books/' + listOfBooks[i].book.key

        root.child(path).once('value').then(snap => {
          const quantity = snap.val().quantity
          const newQuantity = quantity +
            listOfBooks[i].quantity
          snap.ref.update({quantity: newQuantity})
        })

      }

    }

    return null;
  })
```

3.3.9. Zmiana danych użytkownika

Zmiana danych użytkownika polega na edycji pól formularza przedstawionego na rys. 27. Każde z pól jest wymagane, system wywoła metodę edytującą informacje w bazie danych (patrz prog. 23) zostanie wywołana tylko w przypadku gdy formularz przejdzie walidację.

Prog. 23. Zmiana danych użytkownika

```
updateUserData(user: IUser, uid: string, email: string) {
  const updateObject = this._db.object(`/users/${uid}`).update(user)
  .catch((error) => {
    throw new Error(error.message);
  });
  const updateEmail = this.updateEmail(email)
  .catch((error) => {
    throw new Error(error.message);
  });
}
```

```
const updatePersonal = this.updatePersonal(user.name)
  .catch((error) => {
    throw new Error(error.message);
  });

const dataChanged = [
  updateObject,
  updateEmail,
  updatePersonal
];
return Observable.merge(dataChanged);
}
```

The screenshot shows a web application interface. At the top is a blue navigation bar with a hamburger menu icon, a search icon labeled 'Search', a shopping cart icon labeled 'Cart (empty)', and a user profile icon labeled 'Paweł' with a dropdown arrow. Below the navigation bar is a light gray container. Inside this container, there is a green success message banner that says 'The data has been changed.' with a close 'X' button. Below the banner is a user profile form. The form contains the following fields: 'Name *' with the value 'Paweł', 'Surname *' with the value 'Idziak', 'Email *' with the value 'pa.idziak@gmail.com', 'Telephone *' with the value '123 456 789', 'Street *' with the value 'Plac Grunwaldu 1', 'Zip-code *' with the value '50-377', and 'City *' with the value 'Wrocław'. At the bottom of the form is a blue button labeled 'UPDATE'.

Rys. 27. Pomyślna zmiana danych użytkownika

3.4. Instrukcja instalacji

Przed przystąpieniem do uruchomienia projektu należy zainstalować następujące narzędzia:

1. **Node.js** – środowisko uruchomieniowe dla *JavaScript*'u.

Preferowana wersja: 6.11.2

Plik instalacji jak i instrukcja dostępna na oficjalnej stronie: <https://nodejs.org/en/>

2. **NPM** (ang. *Node Packaged Module*) – menedżer pakietów, system paczek.

Preferowana wersja: 3.10.10

Menedżer pakietów jest instalowany wraz z *Node.js* z pkt. 1.

3. **Angular CLI** (ang. *Angular Command Line Interface*) – linia komend dla platformy *Angular*

Preferowana wersja: 1.4.5

Instalacja: `npm install -g @angular/cli`

Więcej informacji odnośnie instalacji znajduje się na oficjalnej stronie: <https://cli.angular.io/>

4. **Przeglądarka internetowa**, np. *Chrome*

Preferowana wersja: 62.0.3202.94

Plik instalacyjny dostępny na oficjalnej stronie: <https://www.google.pl/chrome/browser/desktop/index.html>

Lista kroków potrzebnych do uruchomienia projektu:

1. Przejście do folderu projektu „*e-commerce-app*”:

```
cd e-commerce-app
```

2. Uruchomienie servera:

```
ng serve – domyślny port 4200
```

lub

```
ng serve --port XXXX – gdzie XXXX to numer portu
```

3. Uruchomienie strony w przeglądarce, jako adres podejmy adres lokalny raz ustalony w w pkt. 2 port:

```
localhost:4200 – jeśli nie zdefiniowaliśmy portu
```

```
localhost:XXXX – jeśli zdefiniowaliśmy port, gdzie XXXX to numer portu
```

4. Podsumowanie

Wynikiem pracy jest zaimplementowana aplikacja spełniająca podstawowe funkcjonalności sklepu internetowego czyli: rejestrację, logowanie, wyszukiwanie, przeglądanie i kupowanie produktów, zmiana danych użytkownika oraz przegląd zamówień jak możliwość ich anulowania. Aplikacja jest nowoczesna, elementy na stronie są elastyczne dzięki czemu cała witryna jest responsywna i można z niej korzystać na różnych urządzeniach o dowolnym rozmiarze ekranu i systemie. Zadbano o to by kod był przejrzysty i elastyczny, dlatego można wprowadzać dowolne typy produktów jak i kategorie.

Aplikacja jest typu SPA (ang. *Single Page Application*) co w połączeniu z bazą danych platformy Firebase daje nam rozwiązanie bazujące na zmianie jakichkolwiek danych na stronie bez jej przeładowania. Dzięki temu gdy stan danego produktu się zmieni, użytkownik natychmiast zauważy zmianę w aplikacji. Dodatkowo stanowi to duży plus w przypadku, gdy użytkownik posiada wolne łącze internetowe – aplikacja zostanie załadowana raz, podczas jej wczytywania, następnie każda akcja wykona się bez jej przeładowania.

Wszystkie elementy wizualne strony wraz z ich kolorystyką oraz rozłożeniem bazują na standardzie *Material Design* dzięki czemu interfejs aplikacji jest prosty i użyteczny. Użytkownik ma dostęp do wszystkich potrzebnych funkcjonalności oraz informacji poprzez pasek narzędzi i pasek boczny, które są dostępne z każdego miejsca aplikacji.

System został zaimplementowany również z uwzględnieniem zasad bezpieczeństwa transakcji. Każdy z dostępnych formularzy w aplikacji jest odpowiednio walidowany, dzięki czemu mamy pewność, że do bazy danych zostały dodane poprawne informacje. Zdefiniowano również reguły bezpieczeństwa (patrz rys. 5) w platformie Firebase przez co informacje z oraz do bazy danych są dostępne dla nieupoważnionych użytkowników.

Projekt jest na tyle elastyczny by można rozbudowywać go o dowolne funkcjonalności, takie jak dodatkowe metody płatności, opinie klientów czy rozbudowany panel administratora. Niniejszy projekt można wdrożyć do użytku produkcyjnego przy bardzo małych kosztach finansowych. Wyjątek stanowi przypadek, gdy aplikacja rozrośnie się do dużych rozmiarów. Platforma Firebase jest darmowa do pewnego stopnia¹². Gdy baza produktów lub liczba użytkowników przekroczy dany limit, należy wykupić odpowiedni pakiet.

¹² Ceny pakietów dostępne na stronie: <https://firebase.google.com/pricing/>

Literatura

- [1] Adam Trachtenberg, D. S. (2007). PHP. Receptury. Wydanie II .
- [2] Beighley, L. (2012). SQL. Rusz głową!
- [3] Delaney, J. (2017). The Angular Firebase Survival Guide: Build Angular Apps on a Solid Foundation with Firebase.
- [4] Duckett, J. (2014). HTML i CSS. Zaprojektuj i zbuduj witrynę WWW. Helion.
- [5] Eric T Freeman, E. R. (2012). HTML5. Rusz głową! Helion.
- [6] Eric T. Freeman, E. R. (2015). Programowanie w JavaScript. Rusz głową! Helion.
- [7] Gajda, W. (2013). Git. Rozproszony system kontroli wersji
- [8] Kunz, G. (2017). Angular 2. Tworzenie interaktywnych aplikacji internetowych. Helion.
- [9] Lavin, P. (2006). Object-PHP. Programowanie obiektowePHP. Helion.
- [10] MacGregor, A. (2014). Magento. Przewodnik dla programistów PHP.
- [11] Mariusz Chudzik, A. F. (2005). Prawo handlu elektronicznego.
- [12] Marrs, T. (2017). JSON at Work. Practical Data Integration for the Web. O'Reilly Media.
- [13] McFarland, D. S. (2013). CSS3. Nieoficjalny podręcznik. Wydanie III. Helion.
- [14] McFarland, D. S. (2016). CSS. Nieoficjalny podręcznik. Wydanie IV. Helion.
- [15] Mew, K. (2015). Learning Material Design.
- [16] Robbert, R. (brak daty). Building E-Commerce Solutions with WooCommerce - Second Edition. Packt Publishing.
- [17] Rozentals, N. (brak daty). Język TypeScript. Tajniki kodu. Wydanie II.
- [18] Sullivan, D. (2016). NoSQL. Przyjazny przewodnik.

Dodatek A

Poniżej przedstawiono spisy rysunków, tabel oraz programów prezentowanych w omawianej pracy inżynierskiej.

Spis rysunków

Rys. 1. Statystyki technologii sklepów internetowych – sierpień 2016	7
Rys. 2. Zawartość folderu stworzonego komponentu	13
Rys. 3. Panel administracyjny Firebase	17
Rys. 4. Dostępne metody logowania w Firebase	18
Rys. 5. Baza danych Firebase.....	19
Rys. 6. Reguły zabezpieczeń bazy danych Firebase	20
Rys. 7. Widok aplikacji dla większych urządzeń	22
Rys. 8. Widok aplikacji dla mniejszych urządzeń	23
Rys. 9. Diagram przypadków użycia	25
Rys. 10. Formularz rejestracji	26
Rys. 11. Komunikat błędu rejestracji	27
Rys. 12. Potwierdzenie rejestracji	28
Rys. 13. Formularz logowania, dane przykładowe	29
Rys. 14. Logowanie, błąd - niezweryfikowany email.....	30
Rys. 15. Widok produktów – miniatury	31
Rys. 16. Widok produktów - lista	32
Rys. 17. Szczegóły produktu.....	34
Rys. 18. Wyszukiwanie produktów.....	36
Rys. 19. Przykład dodania produktu do koszyka	38
Rys. 20. Widok koszyka.....	39
Rys. 21. Sprawdzenie zamówienia.....	43
Rys. 22. Adres wysyłki	43
Rys. 23. Potwierdzenie zamówienia	44
Rys. 24. Potwierdzenie zamówienia	45
Rys. 25. Podsumowanie złożonego zamówienia	46
Rys. 26. Zamówienia użytkownika.	48
Rys. 27. Pomyślna zmiana danych użytkownika	50

Spis tabel

Tab. 1. Porównanie języka TypeScript i JavaScript.....	14
Tab. 2. Przykład użycia HTML wraz z CSS	15
Tab. 3. Porównanie CSS z SCSS	16
Tab. 4. Przykład deklaracji domieszki i jej użycie:.....	16
Tab. 5. Przykład deklaracji funkcji i jej użycie:.....	16
Tab. 6. Walidacja pól formularza rejestracji	27

Spis programów

Prog. 1. Przykład wstrzykiwania zależności w Angular 4	12
Prog. 2. Dodawanie kategorii i książek do bazy danych.....	24
Prog. 3. Kod odpowiedzialny za rejestrację	25
Prog. 4. Kod odpowiedzialny za logowanie	29
Prog. 5. Utrzymanie sesji	30
Prog. 6. Wylogowanie	30
Prog. 7. Pobieranie wszystkich produktów	31
Prog. 8. Pobieranie produktów z danej kategorii	32
Prog. 9. Sortowanie produktów	33
Prog. 10. Pobieranie pojedynczego produktu z bazy danych.....	34
Prog. 11. Wyszukiwanie produktów	35
Prog. 12. Dodawanie produktu do koszyka.....	37
Prog. 13. Zapisywanie pozycji w koszyku do pamięci podręcznej przeglądarki.....	40
Prog. 14. Odczytywanie listy produktów z pamięci podręcznej przeglądarki	40
Prog. 15. Sprawdzanie zawartości koszyka	41
Prog. 16. Obliczanie ceny całkowitej koszyka.....	41
Prog. 17. Usuwanie pozycji w koszyku	41
Prog. 18. Zmiana liczby egzemplarzy	42
Prog. 19. Pobieranie danych użytkownika	44
Prog. 20. Złożenie zamówienia, dodanie pola w bazie danych.....	47
Prog. 21. Cloud Function - tworzenie zamówienia	47
Prog. 22. Cloud Function - anulowanie zamówienia	49
Prog. 23. Zmiana danych użytkownika	49