Jakub Bogusz

Informatyka Techniczna 2 stopnia

Zaawansowane Techniki Internetowe

Sprawozdanie z projektu

Temat projektu: „Internetowy sklep koszykarski” w oparciu o technologie ASP.NET Core 5.0,  
 Angular 13, SQLite oraz Redis.

1. Backend aplikacji – technologia Microsoft ASP.NET Core 5.0

W pierwszym kroku utworzone zostało Web API wykorzystujące architekturę REST oraz po stronie backend’u zaimplementowano logikę biznesową. Stworzone zostały modele encji bazodanowych, a przy użyciu MS Entity Framework Core wygenerowano migracje, które utworzyły bazę danych „basketstore.db” w technologii Sqlite. Baza ta przechowuje dane związane min. z produktami dostępnymi w sklepie, typami produktów, markami oraz zamówieniami. Kolejną bazą jest „identity.db”, która wygenerowała została przy pomocy frameworka ASP.NET Core i jest odpowiedzialna za bezpieczne przechowywanie danych, autoryzację użytkowników oraz rejestrację nowych kont. Poniżej zamieszczony został zrzut ekranu przedstawiający budowę tabel obu baz danych:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Web API napisane zostało w języku C# z wykorzystaniem wzorców projektowych takich jak Generic Repository Pattern, Unit Of Work, Dependency Injection. Dla rozdzielenia obowiązków i większej przejrzystości projektu zastosowano podział na kilka warstw umieszczonych w osobnych projektach w ramach jednej solucji. Taki podział pozwala na mniejsze powiązanie ze sobą poszczególnych modułów aplikacji, które nie koniecznie powinny mieć do siebie referencje (nie muszą posiadać odniesień). Tymi warstwami są:

1. Warstwa dostępu do danych w projektcie Core - gdzie zdefiniowane zostały modele encji bazodanowych służące do wyciągania danych oraz zapisu do bazy danych, a także interfejsy. Poniżej zamieszczam przykładową definicję encji używanych w projekcie:

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Warstwa genrycznego repozytorium w projekcie Infrastructure – tu zdefiniowana została logika biznesowa oraz kod odpowiedzialny za operacje CRUD (Create, Read, Update, Delete), które są podstawą działania każdego Web API. W projekcie tym umieszczono również migracje wygenerowane przy pomocy Entity Frameworka. Zdefiniowana została tu również klasa StoreContext.cs, która umożliwia konfigurację połączenia aplikacji z bazą danych w metodzie OnModelCreating, a także do rejestracji instancji DbSet dla modeli z projektu Core.cs.

Obraz zawierający tekst, ekran, zrzut ekranu

Opis wygenerowany automatycznie

Implementacja generycznego repozytorium pozwala na używanie jednej bazy kodu definiującej generyczne metody, które mogą być wykorzystywane dla różnych typów danych. Oznacza to, że typy danych wcale nie muszą być znane przed kompilacją, a wręcz przeciwnie repozytorium dowiaduje się o danych typie w trakcie działania programu. Dodatkowo implementacja wzorca projektowego Generic Repository pozwala nam na spełnienie jednej z najważniejszych zasad pisania „czystego kodu”, tj. „Don’t repeat yourself”. Nasz kod został zdefiniowany jeden raz i może zostać wykorzystany w wielu kontrolerach, a nawet w kolejnych projektach. Jako, że implementacja tego wzorca jest dość skomplikowana, poniżej zamieszczono jedynie kluczowe dla jego działania fragmenty:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Dodatkowo w API wykorzystany został wzorzec Unit Of Work, służący do grupowania jednej lub więcej operacji (zwykle CRUD na bazie danych) w pojedynczą transakcję lub „jednostkę pracy”, tak aby wszystkie operacje przychodziły lub kończyły się niepowodzeniem jako jedna jednostka. Implementacja w projekcie przedstawia się następująco:

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Warstwa API – są tu zdefiniowane kontrolery służące do przesyłania żądań Http i komunikację z Frontendem. Kontrolery opierają się na frameworku ASP.NET Core MVC   
   i implementują zachowania bazowego ControllerBase.cs. Przykładowa metoda GET w kontrolerze ProductController korzystająca z generycznego repozytorium:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Dodatkowo projekt API wykorzytuje narzędzie Automapera, którego zadaniem jest przekształcenie danych wyciąganych z bazy danych do bardziej przejrzystego obiektu, który w programowaniu nazywany jest DTO (data transfer object). Obiekty tego typu często używane są w kontrolerach po to, aby przekazywać do warstwy widoku mniej skomplikowane obiekty pozbawione wszelkich relacji pomiędzy encjami, a także dzięki nim dane z widoku zapisywane są do prostszego obiektu. W tym miejscu zadaniem Automapera jest zmapowanie DTO na model bazodanowy lub odwrotnie. Przykładowy profil mapujący zamieszono poniżej:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Dla właściwości PictureUrl możemy zauważyć wykorzystanie klasy OrderItemUrlResolver. Jest to specjalna klasa, która pozwala przekształcać bardziej skomplikowane obiekty – w zależności jaki typ destynacji jest oczekiwany.

Warto wspomnieć również o logowaniu i rejestracji użytkowników. Kod odpowiedzialny za te funkcjonlności korzysta ze standardowej biblioteki Microsoft.AspNetCore.Identity, która umożliwia bezpieczne dodawanie oraz przechowywanie haseł oraz danych użytkowników. Bibliotekę tą można bezpłatnie zainstalować z poziomu Managera Pakietów Nuget. W poniższym przykładzie metody odpowiedzialnej za logowanie użytkownika możemy zobaczyć wykorzystanie jednej z funkcji dostępnych w tej bibliotece, tj: CheckPasswordSignInAsync().

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Frontend aplikacji – Angular 13, Boostrap 5

Frontend projektu został napisany technologii Angular z wykorzystaniem języka Typescript.

Angular to kompleksowy framework do tworzenia aplikacji typu SPA (Single Page Application). SPA to strony, które są ładowane w przeglądarce dynamicznie. Nie mamy tutaj dociągania strony z serwera gdy przechodzimy na poszczególne podstrony. Angular został stworzony przez Google i społeczność, ponieważ jest to technologia o otwartych źródłach. Framework ten w sobie dostarcza wszystko, co jest niezbędne do stworzenia aplikacji internetowej. W porównaniu do innych frameworków, gdzie często musimy skorzystać z dodatkowych rozwiązań na przykład biblioteki HTTP.

Angular napisany jest z użyciem języka TypeScript, który ostatecznie kompiluje się do języka JavaScript. Angular jest następcą po AngularJS, z którym nie jest kompatybilny.

W projekcie zdefiniowanych zostało 5 podstawowych komponentów:

- basket,

- checkout,

- home,

- orders,

- shop

Każdy komponent składa się z plików: html, scss oraz typescript. Omówię działanie tych komponentów w Angularze na podstawie „basket component”, ponieważ pozostałe komponenty opierają się na podobnych schemacie, wprowadzając niekiedy modyfikacje i dodając kolejne rozwiązania typowe dla tej technologii.

Na początku powinniśmy zdefiniować basket-routing.module.ts:

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Wygląd komponentów języku html: Obraz zawierający tekst

   Opis wygenerowany automatycznie

Zdefiniować należy również basket.module.ts:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Za obsługę koszyka odpowiedzialne są metody zdefiniowane w pliku basket.component.ts, którego implementacje zamieszczono poniżej (większość operacji na koszyku zawiera basket.service):

Obraz zawierający tekst

Opis wygenerowany automatycznie

Główna logika obsługi koszyka zakupów jest zawarta w pliku basket.service.ts. Poniższa metoda calculateTotals przedstawia obliczanie końcowej wartości koszyka, która zostanie wyświetlona na ekranie:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Z kolei metoda removeItemFromBasket definiuje zachowanie usuwania przedmiotu po kliknięciu odpowiedniego przycisku przez użytkownika:

Obraz zawierający tekst

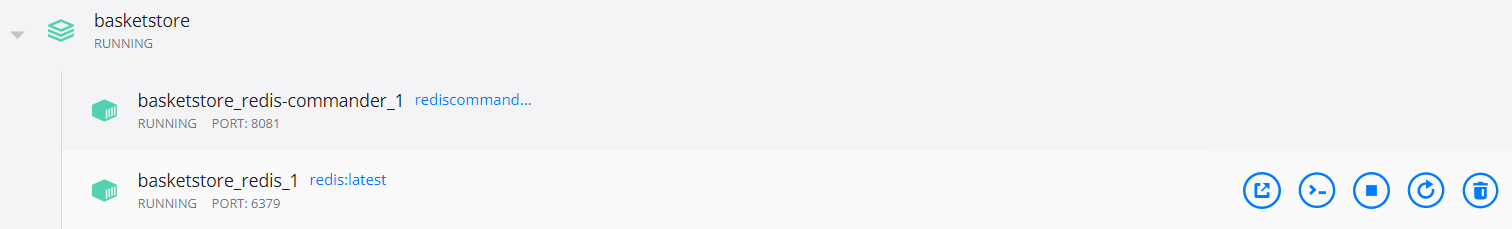
Opis wygenerowany automatycznie

Wygląd koszyka po dodaniu produktów przedstawia się następująco:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Koszyk w naszej aplikacji wykorzystuje jeszcze jedną technologię, a mianowicie bazę danych Redis. Jest to oprogramowanie działające jako nierelacyjna baza danych, przechowująca dane w strukturze klucz-wartość w pamięci operacyjnej serwera, przeznaczona do działania jako klasyczna baza danych, miejsce przechowywania pamięci podręcznej oraz broker wiadomości. Redis został uruchomiony w kontenerze Dockera i przed uruchomieniem aplikacji e-commerce należy zadbać żeby Redis został uruchomiony w tle:



Została ona obsłużona na backendzie w WebApi z wykorzystaniem biblioteki StackExchange.Redis. Za obsługę koszyka właśnie z poziomu backendu odpowiada BasketRepository oparte na interfejsie IBasketRepository.

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Omówienie aplikacji i przedstawienie interfejsu użytkownika.

Aplikacja nie została opublikowana na serwer hostingowy, dlatego nie jest ona publiczna (być może w przyszłości się to zmieni). W związku z tym aby uruchomić aplikację lokalnie, należy z poziomu Visual Studio Code uruchomić projekty:

- WebApi w projekcie API - np. w terminalu za pomocą komendy „dotnet watch run” ,

Obraz zawierający tekst

Opis wygenerowany automatycznie

- Angular w projekcie „client” przy pomocy komendy „ng serve”.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Ekran początkowy widoczny po otwarciu strony:

Obraz zawierający tekst, zawody lekkoatletyczne, sport, koszykówka

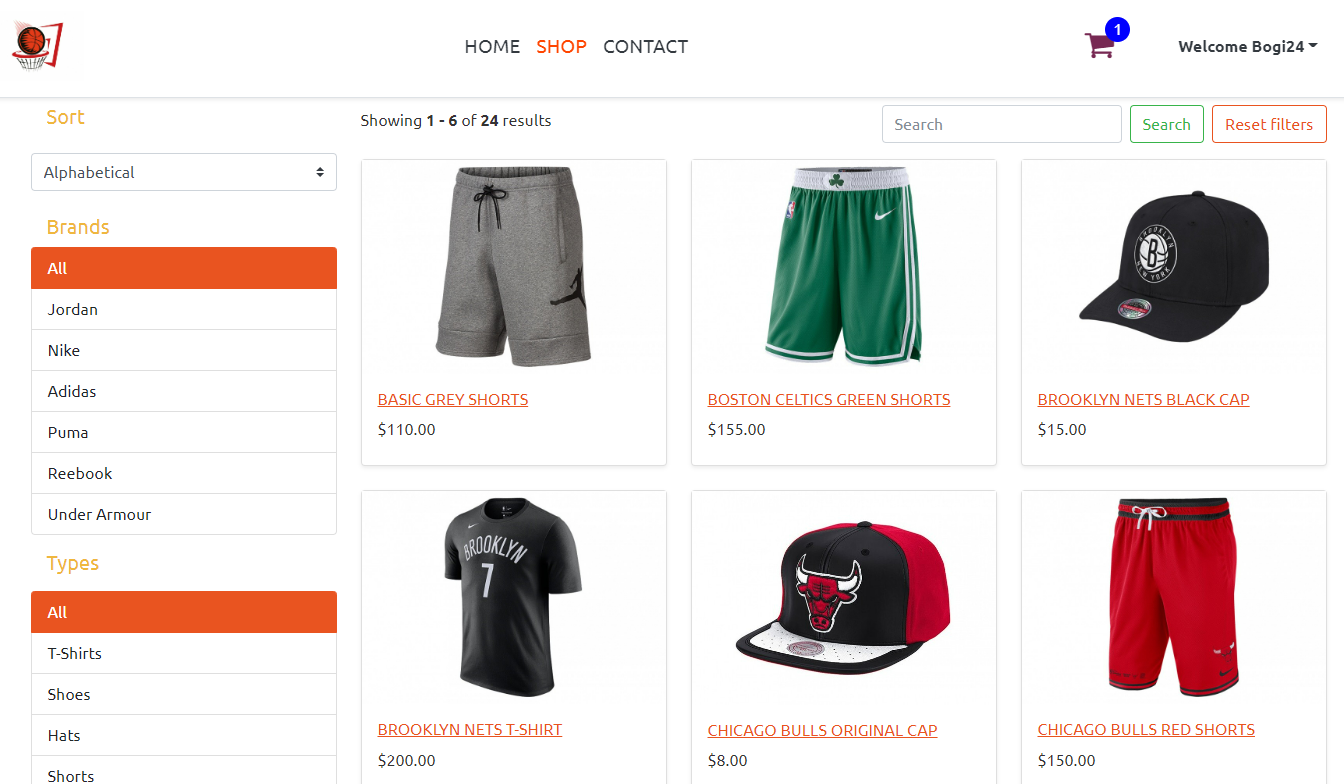
Opis wygenerowany automatycznie

Górny panel nawigacji:

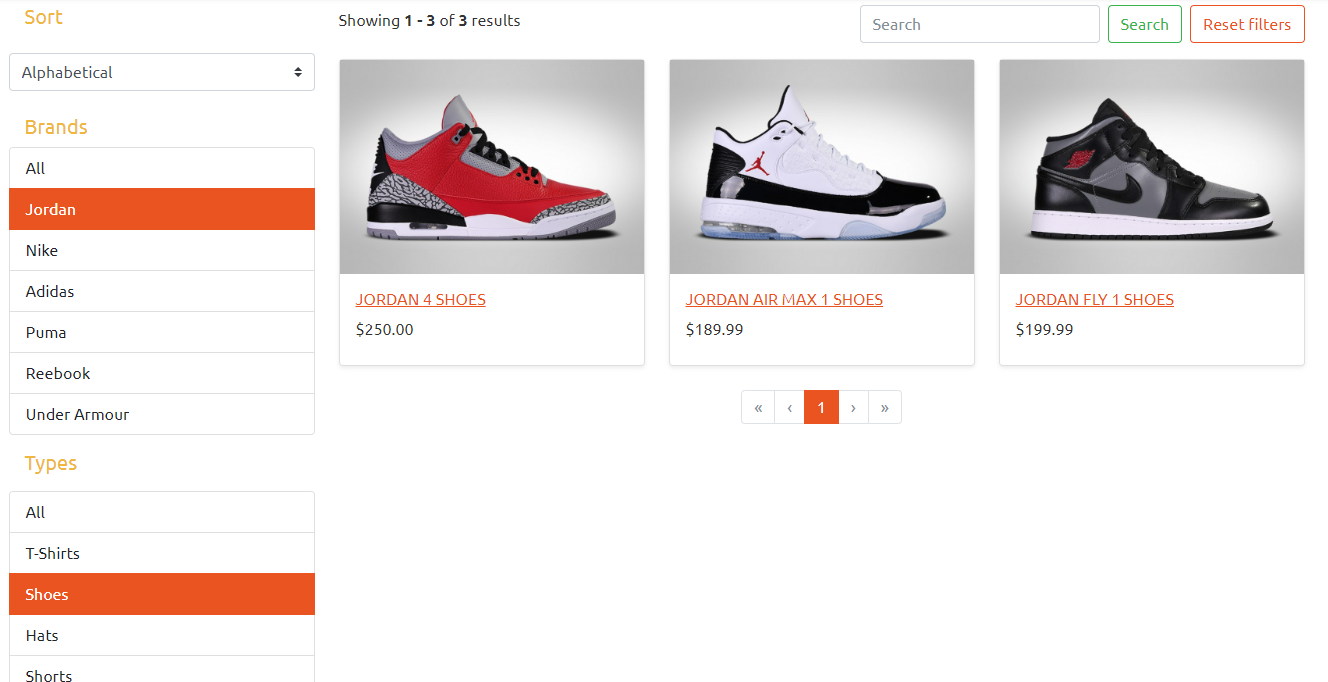
Obraz zawierający tekst

Opis wygenerowany automatycznie

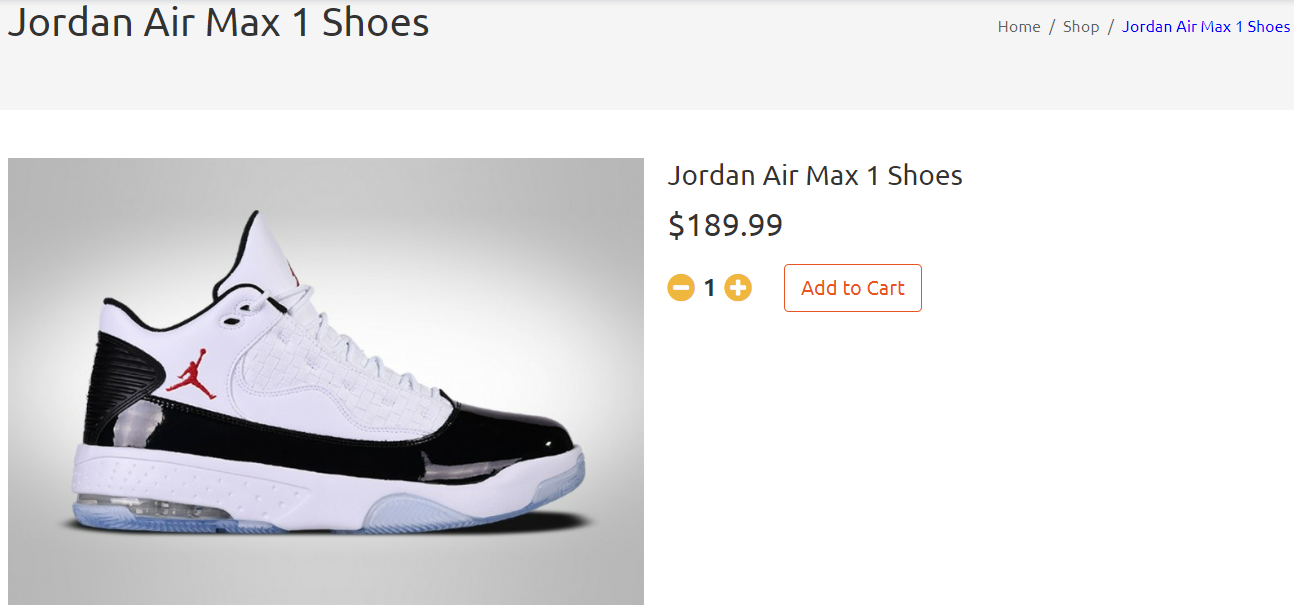
Widok sklepu wraz z produktami:



Zastosowanie filtrów: Marka – Jordan, Typ – buty



Szczegóły przedmiotu:

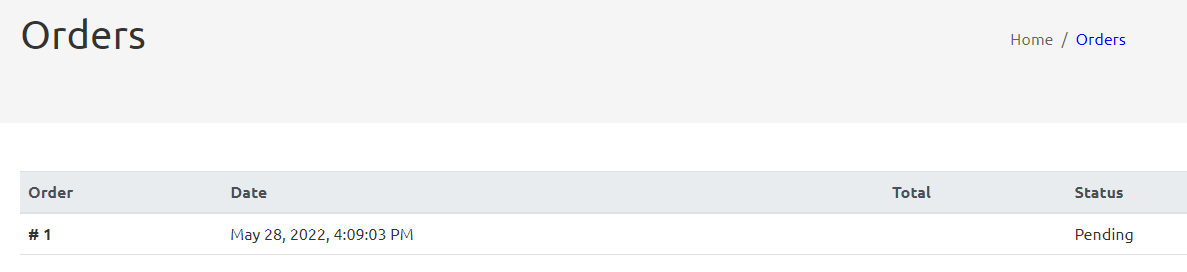


Widok koszyka zakupów:

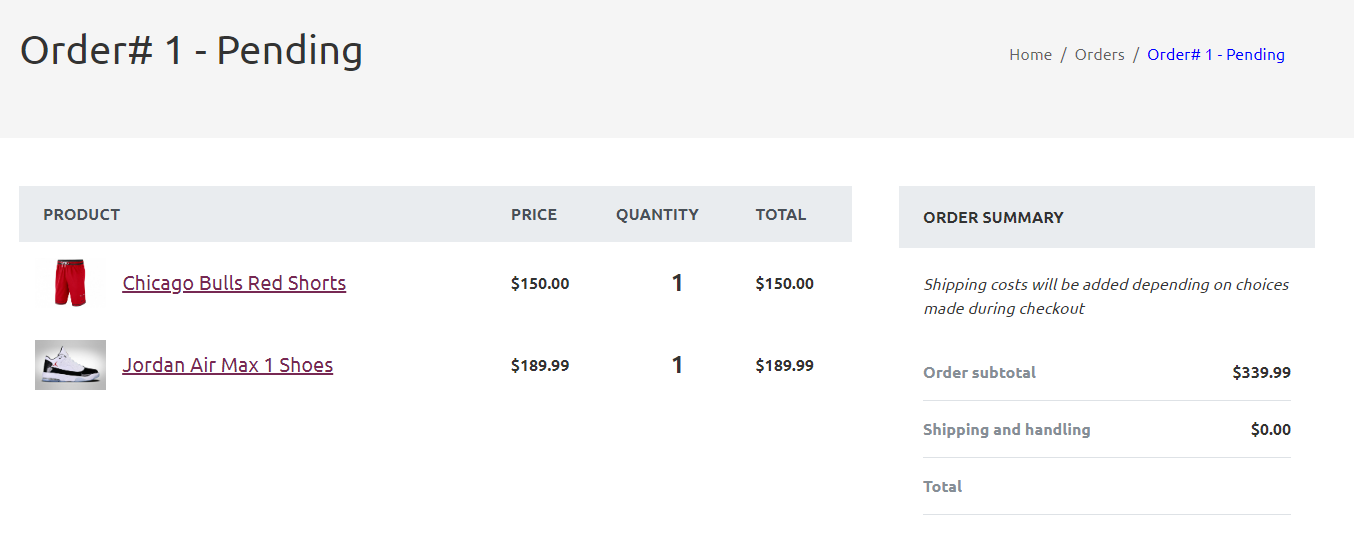
Obraz zawierający tekst

Opis wygenerowany automatycznie

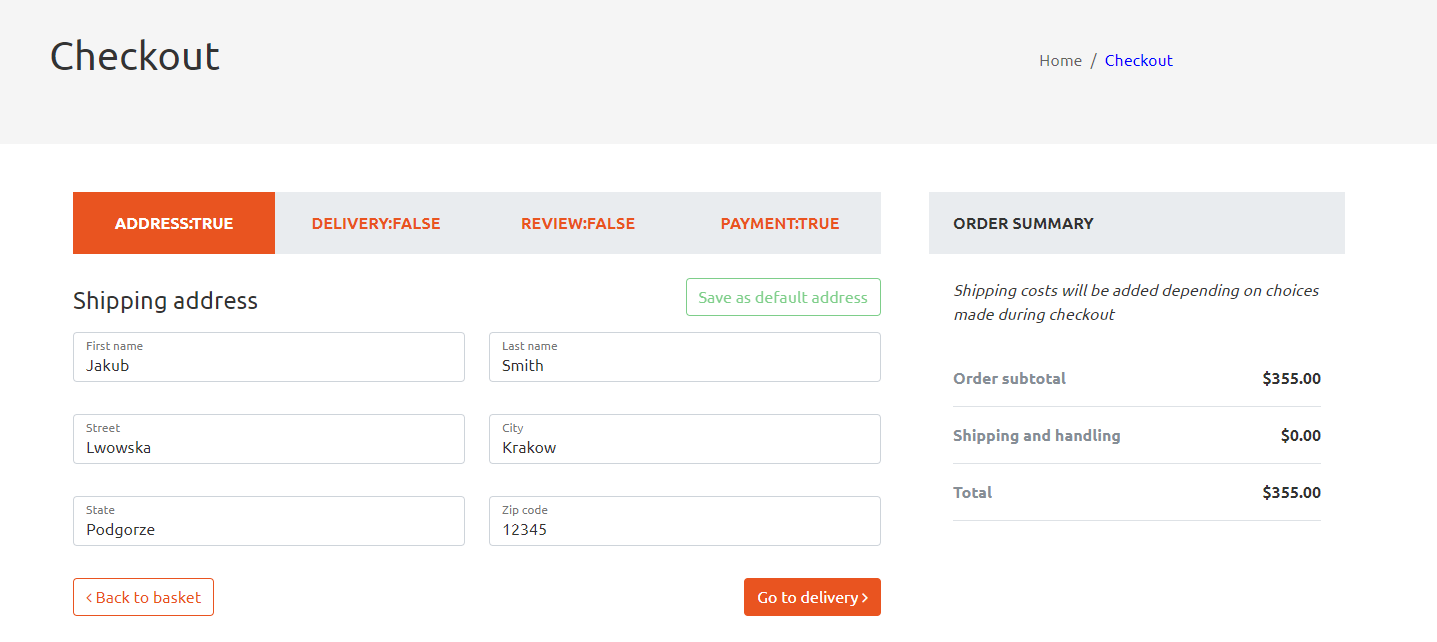
Widok nr 1 w historii zamówień:



Widok nr 2 w historii zamówień:



Widok podsumowania oraz adresu:



Formatki logowania oraz rejestracji użytkownika:

