

Wydział Inżynierii Elektrycznej i Komputerowej

Kierunek Informatyka

Praca inżynierska

Aplikacja mobilna do grupowego zamawiania posiłków dla pracowników firmy

A mobile application for group food ordering from restaurants for company employees

Dyplomant:

**Jakub Gros**

Promotor:

**dr hab. inż. Mieczyslaw Zajac, prof. PK**

Konsultant:

**Mgr Grzegorz Nowakowski**

Kraków 2019

Spis treści

[1. Słownik 3](#_Toc26049013)

[2. Wstęp 4](#_Toc26049014)

[3. Cel, zakres pracy i efekt końcowy 5](#_Toc26049015)

[3.1. Cel 5](#_Toc26049016)

[3.2. Zakres 5](#_Toc26049017)

[3.3. Efekt końcowy 5](#_Toc26049018)

[4. Podejście do problemu 6](#_Toc26049019)

[4.1. Analiza aktualnego podejścia 6](#_Toc26049020)

[4.1.1. Opis realizacji procesu 6](#_Toc26049021)

[4.1.2. Problematyczność aktualnego procesu 6](#_Toc26049022)

[4.2. Koncepcja rozwiązania 7](#_Toc26049023)

[5. Wymagania aplikacji 9](#_Toc26049024)

[5.1. Wymagania funkcjonalne 9](#_Toc26049025)

[5.2. Wymagania niefunkcjonalne 10](#_Toc26049026)

[6. Diagram przepływu danych 12](#_Toc26049027)

[7. Wybór technologii 13](#_Toc26049028)

[7.1. Klient mobilny 13](#_Toc26049029)

[7.1.1. Analiza 13](#_Toc26049030)

[7.1.2. Opis wybranej technologii 14](#_Toc26049031)

[7.2. Serwer i silnik logiczny 14](#_Toc26049032)

[7.3. Baza danych 15](#_Toc26049033)

[7.4. Klient webowy 15](#_Toc26049034)

[8. Implementacja 16](#_Toc26049035)

[8.1. Aplikacja mobilna 16](#_Toc26049036)

[8.1.1. Dostęp 16](#_Toc26049037)

[8.1.2. Przeglądanie zleceń złożenia zamówienia jednostkowego 19](#_Toc26049038)

[8.1.3. Wyświetlanie złożonego zamówienia 22](#_Toc26049039)

[8.1.4. Składanie zamówień 22](#_Toc26049040)

[8.2. Aplikacja webowa 33](#_Toc26049041)

[8.2.1. Tworzenie nowego zlecenia 33](#_Toc26049042)

[8.2.2. Wyświetlanie listy zleceń 34](#_Toc26049043)

[8.2.3. Wyświetlanie zamówienia grupowego 36](#_Toc26049044)

[8.3. Serwer - dokumentacja zasobów 37](#_Toc26049045)

[8.3.1. Autentykacja 38](#_Toc26049046)

[8.3.2. Tworzenie nowego użytkownika 38](#_Toc26049047)

[8.3.3. Grupowe zamówienie 39](#_Toc26049048)

[8.3.4. Sprawdzanie czy użytkownik złożył zamówienie 40](#_Toc26049049)

[8.3.5. Odczytywanie tekstu z obrazka 41](#_Toc26049050)

[8.3.6. Składanie zamówienia jednostkowego 42](#_Toc26049051)

[8.3.7. Sprawdzanie zamówienia jednostkowego 43](#_Toc26049052)

[8.3.8. Utworzenie nowego zlecenia 44](#_Toc26049053)

[8.3.9. Wszystkie zlecenia 45](#_Toc26049054)

[8.3.10. Wszystkie zlecenia złożenia zamówień jednostkowych wraz z identyfikatorem złożonego zamówienia 46](#_Toc26049055)

[8.4. Kody zapytań oraz obsługa wyjątków 47](#_Toc26049056)

[9. Baza danych 48](#_Toc26049057)

[9.1. Analiza przechowywanych danych 48](#_Toc26049058)

[9.2. Schemat tabel 50](#_Toc26049059)

[10. Bezpieczeństwo 50](#_Toc26049060)

[10.1. SQL injection 50](#_Toc26049061)

[10.2. Szyfrowanie danych 52](#_Toc26049062)

[10.3. Autoryzacja zasobów 52](#_Toc26049063)

[11. Architektura aplikacji 54](#_Toc26049064)

[11.1. BLoC 54](#_Toc26049065)

[11.2. Klient - serwer 55](#_Toc26049066)

[12. Podsumowanie 56](#_Toc26049067)

[12.1. Perspektywa dalszego rozwoju aplikacji 56](#_Toc26049068)

[12.1.1. Propozycje zamawiania posiłków 56](#_Toc26049069)

[12.1.2. Usprawnienie OCR 56](#_Toc26049070)

[12.1.3. Powiadomienia 56](#_Toc26049071)

[12.2. Wnioski 56](#_Toc26049072)

[13. Wykaz rysunków 57](#_Toc26049073)

[14. Bibliografia 59](#_Toc26049074)

# Słownik

**administrator** - osoba tworząca zlecenie zamówienia i składająca zamówienie grupowe, będąca łącznikiem pomiędzy interfejsem aplikacji, a restauracją, której jedynym interfejsem do składania zamówień na dowóz jest połączenie telefoniczne.

**zamówienie grupowe** - zamówienie składające się z zamówień jednostkowych.

**zlecenie zamówienia** - polecenie dla zamawiających o konieczności złożenia zamówień jednostkowych.

**zamawiający** - osoba składająca zamówienie jednostkowe

**zamówienie jednostkowe** - zamówienie, które składane jest przez każdego z pracowników firm.

# Wstęp

W obecnych czasach, gdy ludzie żyją w dużym pośpiechu, mocno spopularyzowało się zamawianie jedzenia na dowóz. Podyktowane jest to wielką wygodą, którą umożliwiają nam interfejsy udostępnione przez dedykowane do tego aplikacje.

W ostatnich latach w Polsce mocną popularność zyskały takie serwisy jak Pyszne.pl czy też Pizza Portal, które tworzą wielką sieć restauracji dostarczając dogodny interfejs dla końcowego użytkownika, umożliwiający przy pomocy paru kliknięć zamówienie jedzenia oraz jego opłacenie. Niestety, ale uczestnictwo w takiej sieci restauracji nie jest darmowe dla właścicieli restauracji, co powoduje, że nie wszyscy decydują się na dołączenie się do programu, ze względów finansowych. Odbija się to na osobie zamawiającej, która w przypadku chęci złożenia zamówienia na dowóz, zmuszona jest do skorzystania z mniej dogodnej opcji - zamówienia przez telefon i płatności przy odbiorze.

Wspomniane wcześniej wysokie tempo życia oraz duża dostępność gotowego jedzenia na dowóz doprowadziła do tego, że niektóre fimry zdecydowały się na wynagradzanie swoich pracowników posiłkami w pracy, które spożywają podczas przerw w swoich codziennych obowiązkach. Wpływa to pozytywnie na ich produktywność, ponieważ będąc głodnym spada zdolność skupienia się, gdyż nasze myśli błądzą wokół jedzenia. Dodatkowym atutem tego rozwiązania jest wspólne spożywanie posiłków, co integruje ze sobą pracowników firmy, a często też prowadzi do sytuacji, gdy po kontakcie z osobą spoza swojego wąskiego otoczenia (pracownicy z którymi ściśle współpracujemy i z którymi mamy jedyny kontakt podczas pracy) dostajemy olśnienia, ponieważ ktoś rzuca nowe światło na nasz problem.

Podobne rozwiązania stosuje wszystkim nam znana firma Google. Projektuje ona swoje biura tak aby zmaksymalizować “casual collisions”, co oznacza, że toalety i inne pomieszczenia, z których korzysta każdy z pracowników rozmieszczone są w taki sposób, aby zmusić ich do wyjścia ze swojego hermetycznego otoczenia, z którym ściśle współpracują i nieświadomie zmusić ich do interakcji z pracownikami z innych projektów, dziedzin itp.

W jednej z Krakowskich firm IT pracownicy w ramach dodatkowego wynagrodzenia otrzymują posiłki w pracy. Zamawiane one są z różnych lokalnych restauracji, które często nie posiadają wygodnego interfejsu do składania zamówień poprzez internet i jedyną opcją złożenia zamówienia na dowóz jest kontakt telefoniczny.

Firma cały czas się rozrasta i proces zamawiania posiłków zaczyna stawać się problematyczny ze względu na to, że zajmuje on coraz więcej czasu, w szczególności osobie odpowiedzialnej za zamawianie obiadów. Ponadto osoba ta, mając dużą ilość zamówień może pomylić się podczas sumowania zamówień jednostkowych. Przypadkowo pominięte zamówienia są “domawiane”, lecz często wiążę się to z dodatkowymi kosztami za dowóz, ponieważ jeden posiłek nie przekracza minimalnej kwoty zamówienia potrzebnej do darmowej dostawy.

# Cel, zakres pracy i efekt końcowy

## Cel

Celem pracy jest zaproponowanie rozwiązania problemu czasochłonnego zamawiania posiłków, które usprawni ten proces.

Rozwiązanie to jest dedykowanym rozwiązaniem opartym na własnych doświadczeniach podczas uczestniczenia w procesie zamawiania jedzenia w jednej z Krakowskich firm IT. Dodatkowym celem jest osiągnięcie rozwiązania o niskich kosztach utrzymania i wdrożenia.

## Zakres

Zakresem pracy jest opracowanie systemu składania zamówień, w którym zarówno osoby składające zamówienia jednostkowe (pracownicy firmy) oraz jedna osoba administrująca całym procesem(łącznik między osobami składającymi a restauracją) zaoszczędzą czas poświęcony na całą aktywność.

Ponadto do zakresu należy również utworzenie prototypu aplikacji dostępnej dla wszystkich pracowników bez względu na posiadany system na swoim urządzeniu mobilnym i zaproponowanie rozwiązania opartego na darmowych narzędziach.

Całość jako system oraz aplikacja udostępniona dla użytkownika powinny wpłynąć pozytywnie na wygodę składania zamówień zarówno jednostkowych poprzez pracowników firmy jak i składania zamówień grupowych poprzez administratora

## Efekt końcowy

Efektem końcowym będzie aplikacja, która zaoszczędzi dużo czasu wszystkich pracowników, lecz głównie osoby odpowiedzialnej za składanie zamówień grupowych, co pozwoli jej na zużytkowanie go na inne potrzeby firmy, co za sobą pociąga usprawnienie jej działania.

# Podejście do problemu

## Analiza aktualnego podejścia

### Opis realizacji procesu

Aktualnie proces wygląda następująco. W godzinach przedpołudniowych, osoba odpowiedzialna za zamawianie posiłków wysyła do reszty pracowników maila z informacją o tym, z jakiej restauracji danego dnia zamawiane jest jedzenie, do której godziny mają złożyć zamówienie, limit cenowy oraz menu.

Zamówienia jednostkowe od pracowników są odsyłane w odpowiedzi na ten email poprzez wysłanie nazw posiłków które chcą oni otrzymać. Po otrzymniu wszystkich zamówień jednostkowych, osoba składająca zamówienie w restauracji musi przejrzeć wszystkie maile i uporządkować je w formie dogodnej do podyktowania przez telefon tj. zsumować powtarzające się zamówienia. Następnie zamówienie grupowe jest zamawiane poprzez kontakt telefoniczny z resturacją. Etapem finalizującym cały proces jest otrzymanie dostawy i przydział posiłków, do osób które je zamawiały.

### Problematyczność aktualnego procesu

Aktualnie proces zamawiania jest nieoptymalny i zajmuje stosunkowo dużo czasu. Ponadto jest on niewygodny i zaczyna się stawać problematyczny wraz z rozrastaniem się firmy.

Napotkane problemy to:

* niewygoda w sprawdzaniu czy przygotowano już zlecenie zamówienia - trzeba szukać wiadomości pomiędzy innymi mailami,
* ręczna kontrola tego czy wybrane posiłki nie przekraczają limitu cenowego,
* niewygoda przy przepisywaniu nazw posiłków podczas zamawiania z menu w formie obrazu,
* konieczność “przeskakiwania” pomiędzy menu a wiadomością email,
* ręczne zbieranie zamówień jednostkowych poprzez przeszukiwanie wiadomości email i scalanie ich w zamówienie grupowe,
* możliwość pomylenia się przy scalaniu zamówienia grupowego - zwiększenie kosztów,
* ludzie często zapominali o tym co zamówili i musieli sprawdzać w swoim zamówieniu, co było problematyczne, ponieważ trzeba było przeszukiwać skrzynkę email, co zajmowało czas.

## Koncepcja rozwiązania

Program do zamawiania posiłków postanowiono dostarczyć w formie aplikacji mobilnej. Podyktowane jest to wygodą możliwości zamówienia posiłku w dowolnej chwili oraz w dowolnym miejscu bez dostępu do komputera.

Dodatkowo cały system jest wspierany aplikacja webową, która służy do utworzenia zleceń zamówień. W tym przypadku zdecydowano się na taką formę, ponieważ aplikacja mobilna byłaby niewygodna w momencie, gdy trzeba wypełniać formularz zawierający dużo danych, w tym często takie, które są kopiowane z innych okien (link do menu). Mobilność w tym przypadku nie jest konieczna, ponieważ zlecenia są tworzone przez pracownika za to odpowiedzialnego w ramach jego codziennych obowiązków, a więc z miejsca pracy z dostępem do komputera.

Cały proces rozpoczyna się od stworzenia zlecenia zamówienia.

W aplikacji będzie możliwe utworzenie wielu zleceń zamówień. Dzięki temu osoba odpowiedzialna za zamawianie jedzenia w restauracji będzie mogła na początku tygodnia utworzyć zlecenia na cały nadchodzący tydzień, a następnie o wcześniej ustalonych porach wykonać tylko połączenie telefoniczne dyktując wygenerowane zamówienie grupowe, co znacząco skróci cały proces z jej punktu widzenia.

Po utworzeniu zlecenia zamówienia, pojawia się ono w głównym widoku aplikacji mobilnej. Przy obiekcie reprezentującym zlecenie zamówienia widoczne są jego wszystkie atrybuty oraz informacja o tym, czy dany użytkownik złożył już zamówienie jednostkowe.

W przypadku gdy już to zrobił, po kliknięciu w obiekt reprezentujący zlecenie zamówienia przenoszony jest do widoku, w którym może zobaczyć co zamówił.

W przypadku gdy zamówienie jednostkowe nie zostało złożone, użytkownik przenoszony jest do widoku prezentującego menu.

W widoku tym ma on możliwość wyboru posiłków bez konieczności przepisywania ich nazw. Odbywa się to poprzez zaznaczanie nazw posiłków, które następnie przetwarzane są na tekst. Dodatkowo widoczna jest dla niego informacja o tym, ile aktualnie wynosi sumaryczny koszt jego zamówienia.

Przekroczenie limitu nie powoduje braku możliwości złożenia zamówienia, lecz wyświetla dobrze widoczną informację o zaistniałym fakcie. Podyktowane jest to tym, że dozwolone jest przekroczenie limitu niewielką kwotę oraz tym, że niektórzy pracownicy umawiają się, że zamówią razem jakiś większy zestaw, a następnie się nim podzielą - wtedy zamówienie składa tylko jedna osoba. Ze względu na zaufanie do pracowników, nie ma konieczności kontrolowania ich wydatków i przyjmuje się, że będą oni przestrzegać limitu cenowego, więc informacja o sumarycznym koszcie zamówienia ma jedynie charakter informacyjny.

Do każdego zamówionego posiłku pracownik ma możliwość dodania komentarza, który może służyć np. do zmiany składników albo sposobu przygotowania posiłku.

Dodatkowo istnieje możliwość podglądu wszystkich pozycji w aktualnym zamówieniu poprzez przejście do koszyka, z poziomu którego widzimy informacje o poszczególnych posiłkach oraz mamy możliwość zwiększenia/zmniejszenia liczby sztuk lub całkowitego usunięcia pozycji z zamówienia. Po skompletowaniu zamówienia istnieje możliwość sfinalizowania go poprzez kliknięcie na przycisk składający zamówienie. W tym momencie interakcja z aplikacją chwilowo się kończy i następuje oczekiwanie na dostawę jedzenia.

Gdy minie wcześniej ustalony ostateczny termin złożenia zamówienia jednostkowego, osoba odpowiedzialna za składanie zamówienia grupowego wchodzi na stronę, która zawiera wszystkie zlecenia. Są one posortowane malejąco względem tego terminu, co znacząco ułatwia odnalezienie szukanej pozycji. Po odnalezieniu zlecenia zamówienia, które należy złożyć, używa opcji umożliwiającej wyświetlenia wygenerowanego zamówienie grupowego, które jest gotowe do podyktowania przez telefon podczas połączenia z restauracją.

Wygenerowane zamówienie grupowe składa się z wielu zamówień jednostkowych, które są sumowane w przypadku gdy wystąpią powtórzenia. Dzięki temu mając 2 zamówienia jednostkowe z tym samym posiłkiem, na zamówieniu grupowym pozycja taka pojawi się tylko raz, a liczba wskazująca jej ilość zostanie odpowiednio zwiększona.

Po otrzymaniu dostawy, każdy z pracowników, który zapomni co zamówił ma możliwość podejrzenia swojego zamówienia z poziomu aplikacji mobilnej (po to aby nie musiał się wracać z jadalni do swojego stanowiska pracy) poprzez kliknięcie na obiekt przedstawiający zlecenie zamówienia.

W momencie gdy każdy z pracowników otrzymał swoje zamówienie, cały proces zamawiania uznany jest za zakończony, a aplikacja może zostać ponownie użyta przy kolejnym zamawianiu posiłków.

# Wymagania aplikacji

W celu większej kontroli nad implementacją aplikacji i bardziej precyzyjnym określeniu tego, co ma robić, przeanalizowano problem zamawiania posiłków grupowych z restauracji i wyszczególniono w kolejnych podrozdziałach wymagania funkcjonalne i niefunkcjonalne, które program ma spełniać.

## Wymagania funkcjonalne

Wymagania funkcjonalne opisują jakie usługi dostarcza nam program, co powinien on realizować oraz jak zachowuje się on w określonych sytuacjach. Są one następujące:

**Aplikacja mobilna:**

* wybieranie posiłków bez potrzeby przepisywania ich nazw,
* automatyczne sumowanie cen wszystkich wybranych posiłków w celu możliwości kontrolowania kosztów zamówienia (limit cenowy),
* możliwość podejrzenia złożonego zamówienia,
* dodawanie komentarzy do zamawianych posiłków (np. w celu zmiany składników),
* dostęp tylko dla zalogowanych użytkowników,
* możliwość utworzenia konta,
* możliwość zapamiętania hasła w aplikacji,
* w przypadku wprowadzenia nieprawidłowych danych, aplikacja wyświetla odpowiednią informację,
* wyświetlanie wszystkich przeszłych i aktualnych zleceń zamówienia,
* możliwość złożenia zamówienia jednostkowego,

**Serwer:**

* autoryzowany dostęp do niektórych zasobów,
* możliwość pobrania wszystkich utworzonych zleceń zamówień,
* możliwość pobrania konkretnego utworzonego zlecenia zamówień na podstawie jego ID,
* możliwość pobrania informacji o tym, czy użytkownik złożył zamówienie jednostkowe dla danego zlecenia zamówienia,
* możliwość utworzenia nowego konta użytkownika na podstawie podanego adresu email oraz hasła,
* przetwarzanie obrazu na tekst (OCR),
* sprawdzanie czy użytkownik złożył zamówienie jednostkowe dla danego zlecenia zamówienia,
* możliwość złożenia zamówienie jednostkowego,
* możliwość sprawdzenia posiłków znajdujących się w zamówieniu jednostkowym na podstawie jego ID,
* możliwość stworzenia zlecenia zamówienia posiadającego następujące atrybuty:
  + ostateczny termin złożenia zamówienia,
  + link do menu,
  + dodatkowa wiadomość,
  + nazwa,
  + limit cenowy

**Klient webowy:**

* możliwość wyświetlenia wszystkich zleceń zamówień wraz z ich szczegółowymi informacjami,
* możliwość utworzenia nowego zlecenia zamówień,
* ustalanie limitu cenowego dla zlecenia zamówienia,
* ustalanie limitu czasowego na złożenie zamówienia jednostkowego,
* możliwość dodania dodatkowych informacji do zlecenia zamówienia, które będa widoczne dla osoby zamawiającej(np. o możliwości dokonania dodatkowego wyboru, który nie jest wspomniany w menu),
* możliwość wyświetlenia listy zamówień sformatowanej w sposób wygodny do podyktowanie przez telefon składając zamówienie telefoniczne.

## Wymagania niefunkcjonalne

Wymagania niefunkcjonalne opisują jak program powinien realizować swoje zadania. Definiują one cechy oprogramowania oraz ograniczenia jakie program musi spełniać. Dla naszej aplikacji kształtują się one następująco:

**Serwer:**

* wydajność - odpowiada w czasie krótszym niż 10 sekund,
* poprawność danych - zwraca odpowiednie kody odpowiedzi HTTP,
* kompatybilność - może być uruchomiony na systemach Windows oraz Linux,
* niezawodność - wyjątki zgłaszane na poziomie silnika logicznego nie powodują zatrzymania pracy serwera,
* bezpieczeństwo**:**
  + zasoby o ograniczonym dostępie wymagają autoryzacji,
  + wrażliwe dane są przesyłane i przechowywane w postaci zaszyfrowanej

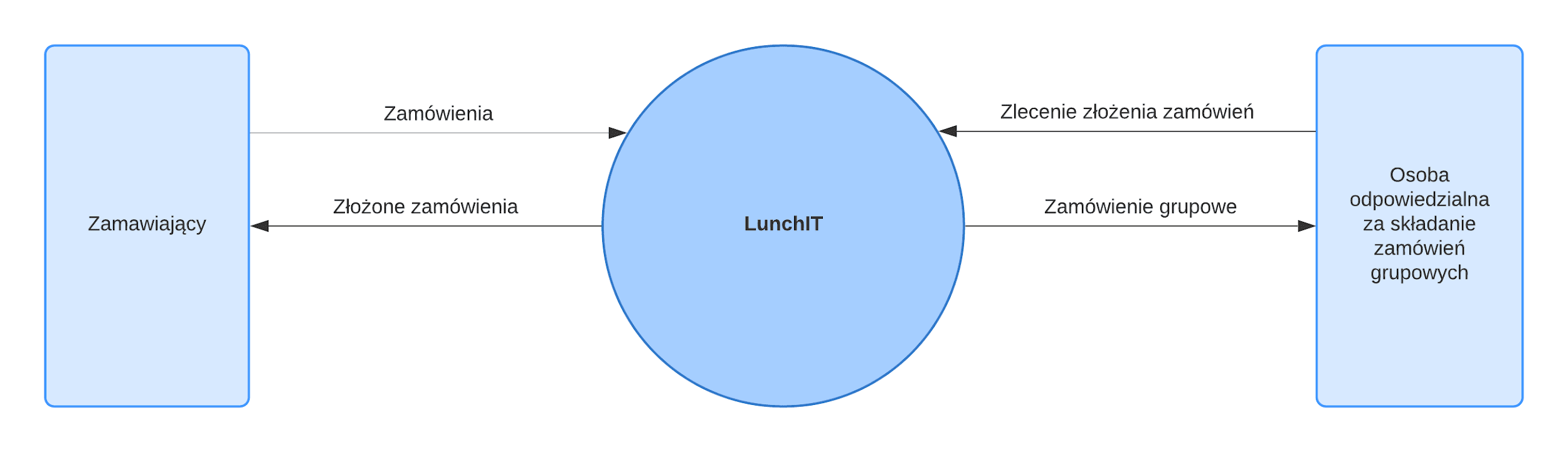
**Aplikacja mobilna:**

* wydajność - płynność działania i brak zawieszeń,
* kompatybilność - możliwość uruchomienia aplikacji na systemie Android Jelly Bean, v16, 4.1.x lub nowszym oraz na systemie iOS 8 lub nowszym,
* poprawność danych
  + weryfikuje nieprawidłowe dane wprowadzone przez użytkownika,
  + sprawdza czy wymagane pola zostały wypełnione.
* użyteczność - łatwy i intuicyjny interfejs,
* rozmiar - zajmuje mniej niż 100 MB pamięci,

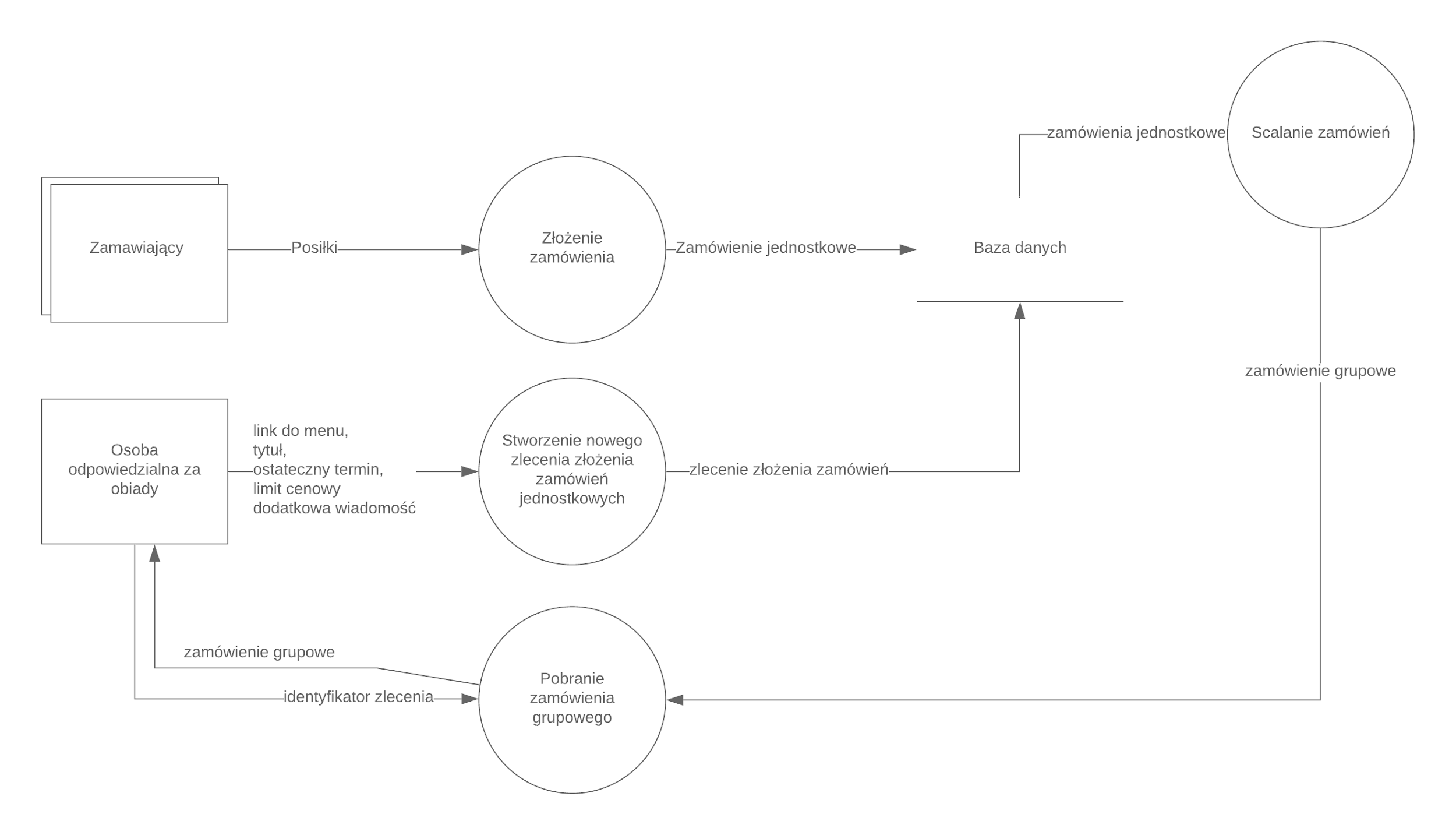
**System jako całość:**

* niski koszt wdrożenia aplikacji,
* niski koszt utrzymania,
* ukończenie przed końcem 2019 roku,
* odporność na popularne ataki hackerskie,
* możliwość złożenia zamówienia jednostkowego bez dostępu do komputera,
* stabilność działania

# Diagram przepływu danych



Rysunek 6.1 Diagram przepływu danych, poziom 0



Rysunek 6.2 Diagram przepływu danych, poziom 1

# Wybór technologii

Podstawowymi kryteriami podczas doboru technologii użytych w aplikacji był brak kosztów związanych z użyciem danej technologii oraz szybkość wytwarzania oprogramowania, przez co dokonano wyborów opisanych w kolejnych podrozdziałach

## Klient mobilny

### Analiza

Badając strukturę rynku smartphonów zauważa się znaczną dominację dwóch platform: Androida oraz iOS. W bieżącym roku (2019) 87% z wszystkich dostarczanych urządzeń mobilnych działa na systemie Android, natomiast 13% z nich obsługiwanych jest przez iOS. Powołując się na badania IDC tendencja rynku smartfonów nie zmieni się w najbliższych latach w znaczny sposób.



Rysunek 7.1 Prognoza udziałów rynkowych systemów na smartfony źródło: <https://www.idc.com/promo/smartphone-market-share/os>

Dodatkowo zauważyć można że tendencja ta utrzymuje się ona na podobnym poziomie od 2017 roku. Ze względu na brak innych znaczących udziałowców rynkowych, znaczna większość aplikacji wytwarzana jest tylko na te 2 platformy.

Wytwarzanie aplikacji na oba środowiska wymaga napisania oddzielnych baz kodów źródłowych co znacząco zwiększa koszty wytworzenia oraz utrzymania aplikacji. W celu obniżenia nakładów pieniężnych potrzebnych do stworzenia programu mobilnego warto zainteresować się językami, które pozwalają na cross-platform development, czyli budowanie aplikacji na wiele platform przy pomocy jednego stacku technologicznego. Możliwości takie daje nam framework Flutter.

### Opis wybranej technologii

Flutter jest otwarto źródłowym i wieloplatformowym frameworkiem do rozwoju aplikacji mobilnych stworzonym przez Google w 2017 roku. Framework jest używany przy pomocy języka Dart.

Do jego zalet należy funkcja hot-reload, która pozwala na szybsze naprawianie błędów w aplikacji oraz wygodniejsze projektowanie jej interfejsu. Funkcja ta pozwala na wyświetlanie zmian zastosowanych w kodzie bez potrzeby przebudowywania aplikacji.

Flutter udostępnia wiele predefiniowanych widgetów, których możemy używać do budowy aplikacji. Ponadto gdy stwierdzimy, że czegoś nam brakuje możemy z łatwością utworzyć własne widgety wedle naszych potrzeb, a następnie z łatwością stosować w kodzie unikając powtórzeń.

Ponadto Flutter udostępnia API, dzięki któremu możemy w łatwy sposób napisać i podłączyć kod napisany w natywnych językach na poszczególne systemy. Dzięki takiej możliwości możemy zoptymalizować pewne fragmenty kodu w zależności od systemu operacyjnego urządzenia, na którym uruchamiana jest aplikacja.

Ze względu na to że Flutter jest nową technologią, niektóre elementy mogą nie działać poprawnie. Jednakże dzięki wcześniej wspomnianej funkcji, mamy możliwość zaimplementowania tych rzeczy w natywnym języku, co zapewnia nas, że podczas pisania aplikacji nie dojdziemy do momentu, w którym musimy zmienić technologię ze względu na błędy w niej występujące, które nas blokują.

## Serwer i silnik logiczny

Jako że jedną z kluczowych funkcji przy zaproponowanym rozwiązaniu jest przetwarzanie obrazów na tekst oraz wymogiem jest szybkość pisania aplikacji, postanowiono wybrać wieloplatformowy język skryptowy Python. Został on utworzony w 1991 roku. Jego głównym założeniem jest możliwość pisania aplikacji na wysokim poziomie abstrakcji. Do jego zalet należy wysoce rozbudowany pakiet bibliotek standardowych, czyli takich, które dostarczane są nam razem z interpreterem języka. Ponadto istnieje bardzo dużo bibliotek stworzonych przez społeczność, które możemy w kilka sekund pobrać i skonfigurować do działania z naszym projektem dzięki menadżerowi pakietów o nazwie pip.

Najważniejsze z użytych bibliotek to:

* **pytesseract** - umożliwia optyczne rozpoznawanie znaków. Oznacza to, że potrafi ona “odczytywać” tekst osadzony w obrazach,
* **psycopg2** - umożliwia operowanie bazami danych PostgreSQL,
* **flask** - framework do tworzenia aplikacji webowych. Umożliwia on m.in. udostępnienie funkcjonalności backendu poprzez API, którego będzie używał klient mobilny oraz webowy,
* **simplejson** - wygodny dekoder i enkoder formatu JSON

## Baza danych

Podczas wyboru bazy głównymi czynnikami decyzyjnymi była licencja pozwalająca na darmowe używanie bazy danych w projektach komercyjnych oraz popularność, po to aby z łatwością można było znaleźć materiały edukacyjne oraz pomoc społeczności.

Wybrano PostgreSQL, który jest systemem zarządzania relacyjnymi bazami danych. Jest on darmowy i otwartoźródłowy. System ten implementuje większość standardu SQL:2011. PostgresSQL można uruchomić na wielu systemach operacyjnych takich jak FreeBSD, OS X, Linux, Windows czy UNIX (AIX, BSD, HP-UX, SGI IRIX, Solaris, Tru64).

## Klient webowy

Do stworzenia klienta webowego wykorzystano jedne z podstawowych a zarazem najpopularniejszych narzędzi:  
  
**HTML** - jest to język znaczników wykorzystywany do tworzenia dokumentów HTML. Pozwala on określić strukturę strony internetowej.

**CSS** - język służący do opisywania wyglądu stron internetowych. Definiuje on listę reguł według których przeglądarka powinna wyświetlać zawartość wcześniej zadeklarowanego w języku HTML elementu. Dzięki niemu możemy określać takie cechy jak położenie elementów, ich kolor, margines, a nawet animacje.

**JavaScript** - jest to język programowania wykorzystywany przy tworzeniu stron internetowych. Umożliwia on nam między innymi na dynamiczną zmianę treści lub jej wyglądu w zależności od sytuacji. Dzięki niemu jesteśmy w stanie dodać logikę do naszej strony WWW.

**jQuery** - biblioteka języka JavaScript znacznie ułatwiająca pracę przy tworzeniu stron. Udostępnia szereg metod pozwalających na manipulację drzewem DOM, tworzenie animacji czy też wysyłanie zapytań do serwera i odbieraniu odpowiedzi.

**Bootstrap** - biblioteka języka CSS udostępniająca narzędzia do projektowania interfejsu graficznego stron internetowych. Dzięki niej, niewielkim nakładem pracy możemy uzyskać stronę o atrakcyjnym wyglądzie.

# Implementacja

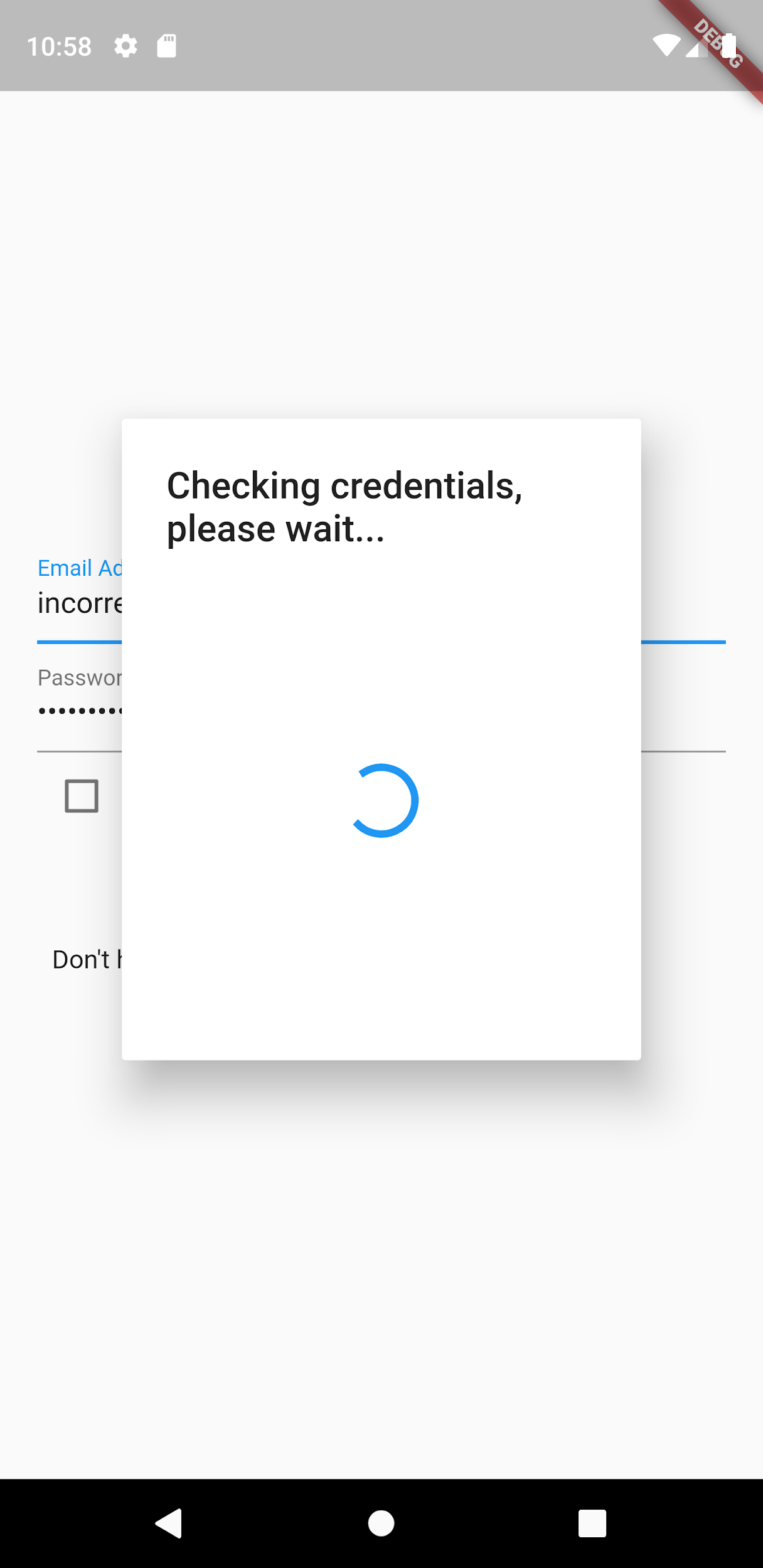
## Aplikacja mobilna

### Dostęp

Rysunek 8.1 Widok logowania

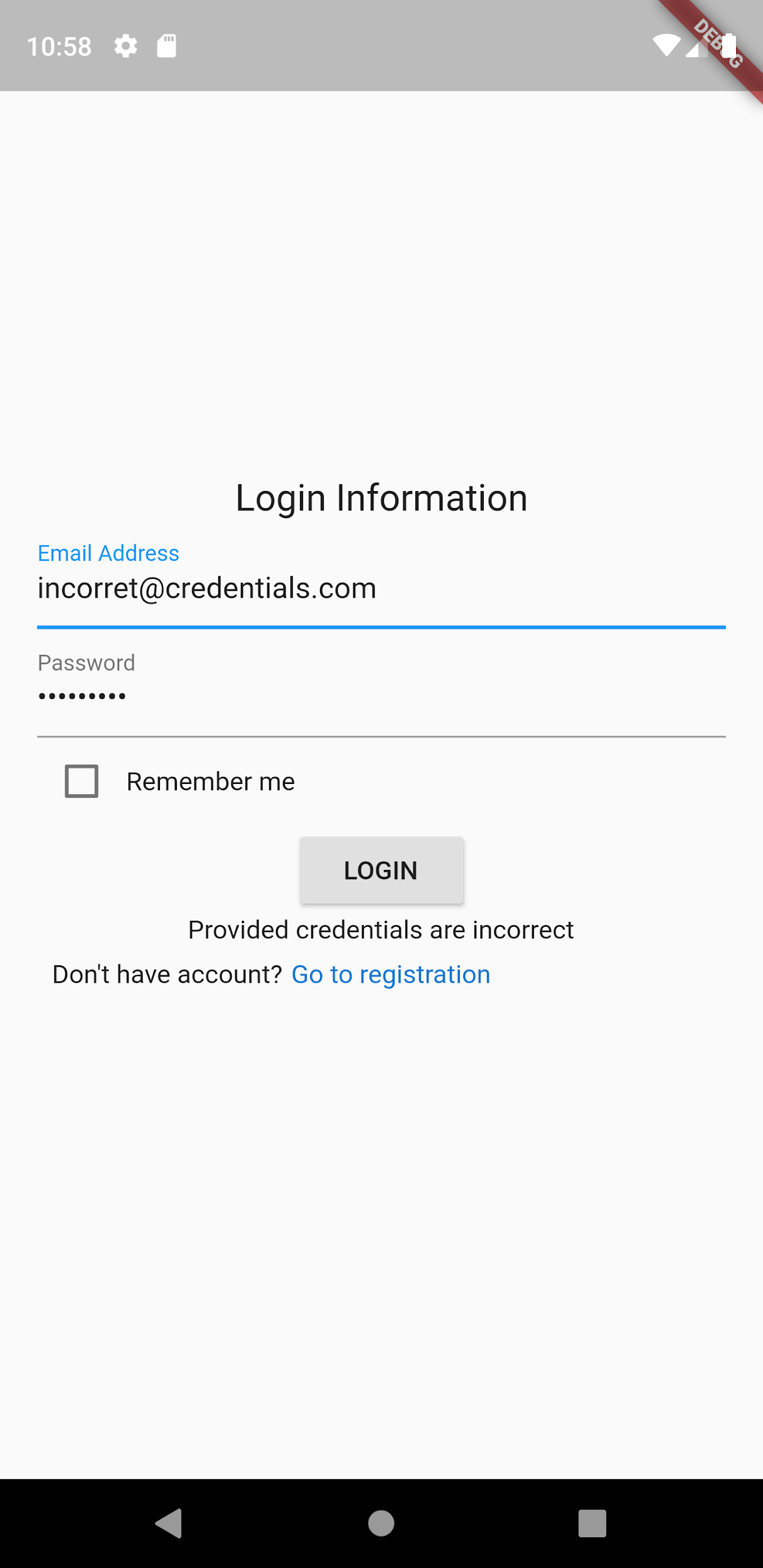
Z aplikacji mogą skorzystać tylko zalogowani użytkownicy. Ekranem, który pojawia się po pierwszym uruchomieniu aplikacji jest ekran logowania.

Użytkownik ma możliwość zapamiętania hasła, po to aby nie musiał go wpisywać przy ponownym uruchomieniu aplikacji.



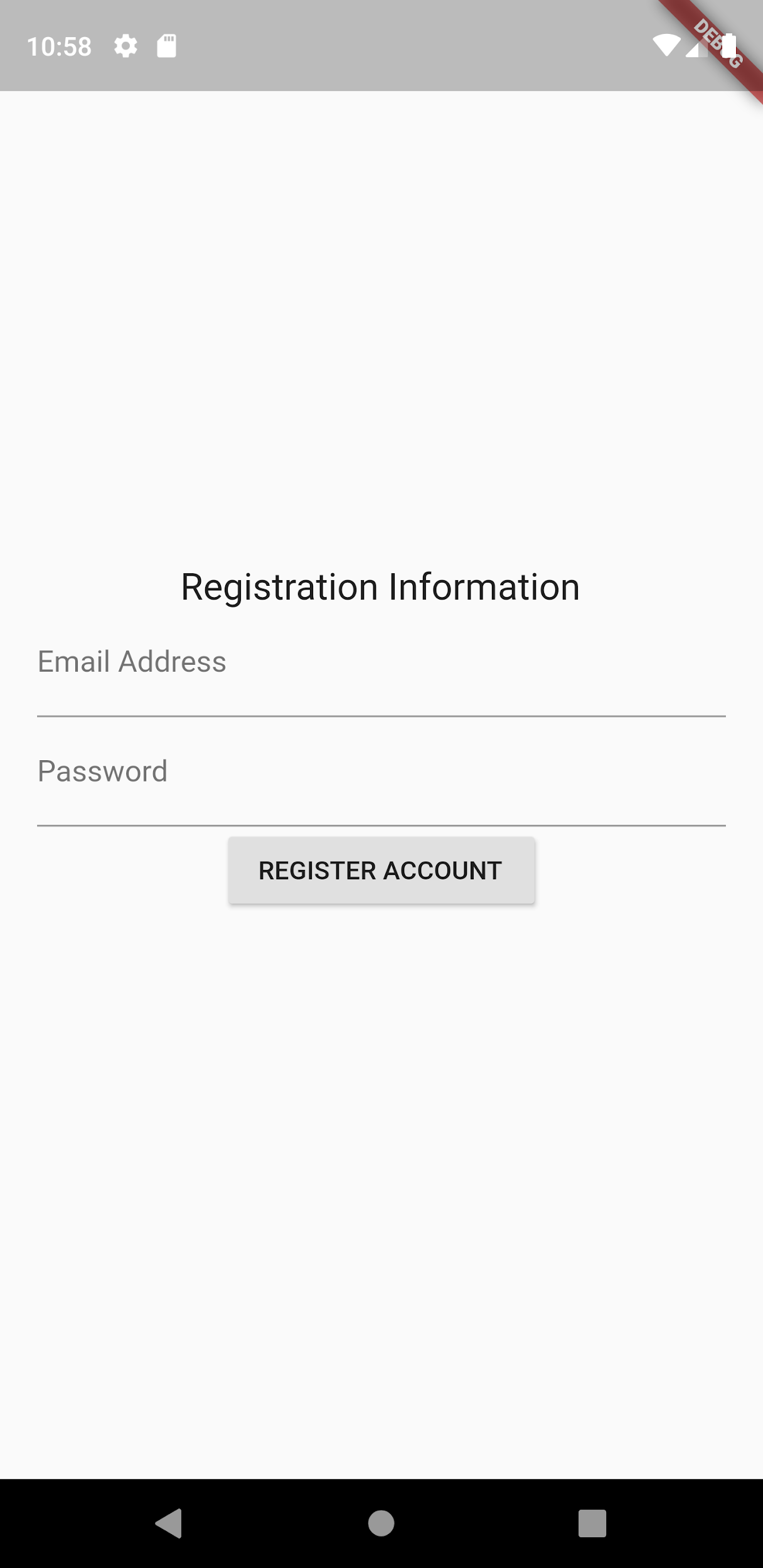
Rysunek 8.2 sprawdzanie poprawności danych logowania

Po kliknięciu w przycisk *Zaloguj* użytkownik informowany jest o trwającym procesie sprawdzania poprawności danych logowania. Wykonywane w tym czasie jest zapytanie do serwera, który sprawdza użytkownika w bazie danych.



Rysunek 8.3 Widok logowania po podaniu nieprawidłowych danych logowania

W przypadku gdy nie udało odnaleźć się użytkownika o podanym adresie e-mail lub podane hasło jest nieprawidłowe, pod przyciskiem *Login* wyświetlana jest stosowna informacja

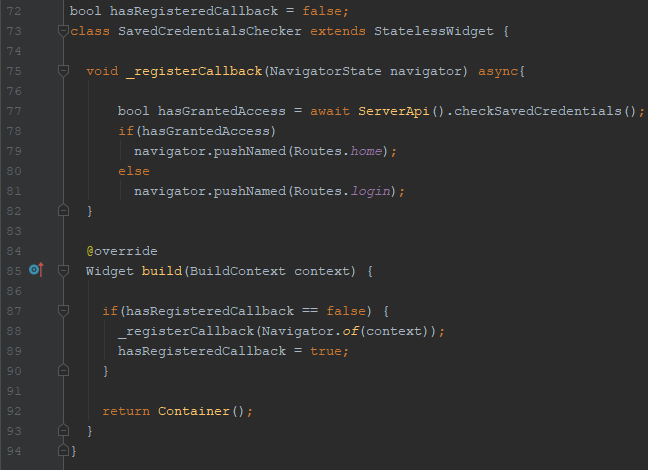


Jeżeli użytkownik nie posiada konta, ma on możliwość utworzenia go z poziomu programu.

Rysunek 8.4 Widok rejestracji nowego użytkownika

Dostęp dla tylko zalogowanych użytkowników zaimplementowany jest w oparciu o wbudowany moduł o nazwie Navigator, który jest odpowiedzialny za nawigowanie pomiędzy wyświetlanymi widokami czyli za wyświetlanie odpowiednich treści na ekranie i wykonywanie odpowiedniej logiki powiązanej z treścią.

Pierwszym widgetem, który jest budowany po uruchomieniu aplikacji jest *SavedCredentialsChecker*, którego zadaniem jest sprawdzenie czy użytkownik zapamiętał swoje hasło podczas poprzedniej sesji. Następuje to poprzez sprawdzenie faktu istnienia pliku z zahaszowanymi danymi logowania. Gdy owy plik istnieje, odczytywana jest jego zawartość i wykonywane jest zapytanie do serwera. Serwer zwraca odpowiedź z informacją czy dane logowania są prawidłowe. W przypadku gdy odnaleziono użytkownika w bazie danych i podane hasło jest prawidłowe, następuje zalogowanie i przekierowanie do strony głównej, natomiast gdy dane nie zostały zapamiętane lub nie są już ważne, użytkownik zostaje przekierowany do strony logowania.

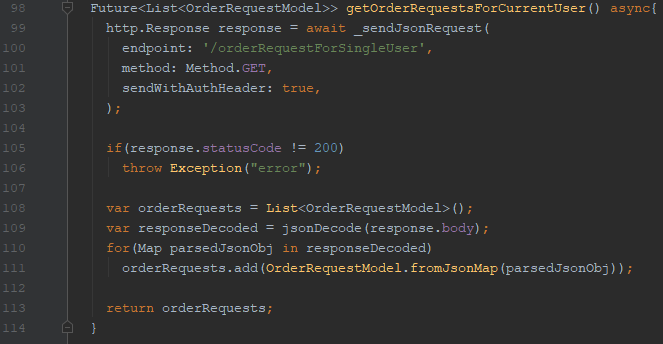
**

Rysunek 8.5 Implementacja modułu odpowiedzialnego za sprawdzanie zapamiętanych danych

Na stronie logowania użytkownik ma możliwość podania swojego loginu i hasła, zapamiętania swoich danych logowania oraz rejestracji. Hasło wprowadzane podczas tworzenia nowego konta lub logowania się do aplikacji przesyłane jest do serwera w postaci zahaszowanej przy pomocy funkcji haszującej *PBKDF2*.

Po zalogowaniu login i zahaszowane hasło zapisywane są w pamięci programu, a następnie dołączane do zapytań HTTP do serwera, które wymagają autoryzacji w postaci dedykowanego nagłówka o nazwie *AUTHORIZATION*.

Do komunikacji z serwerem przygotowano specjalną metodę pomocniczą, która powoduje, że zapytania do serwera są dużo wygodniejsze. Poniżej widzimy przykład użycia tej metody w celu komunikacji z zasobem, który wymaga autoryzacji.

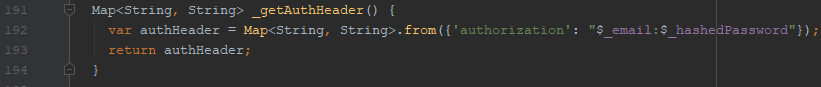


Rysunek 8.6 Przykład użycia metody pomocniczej służącej do komunikacji z serwerem

Na powyższym obrazku warto zwrócić uwagę na argument *sendWithAuthHeader*, który jest ustawiony na wartość *true* i powoduje dodanie nagłówka autoryzacji.

**

Rysunek 8.7 Miejsce użycia argumentu sendWithAuthHeader

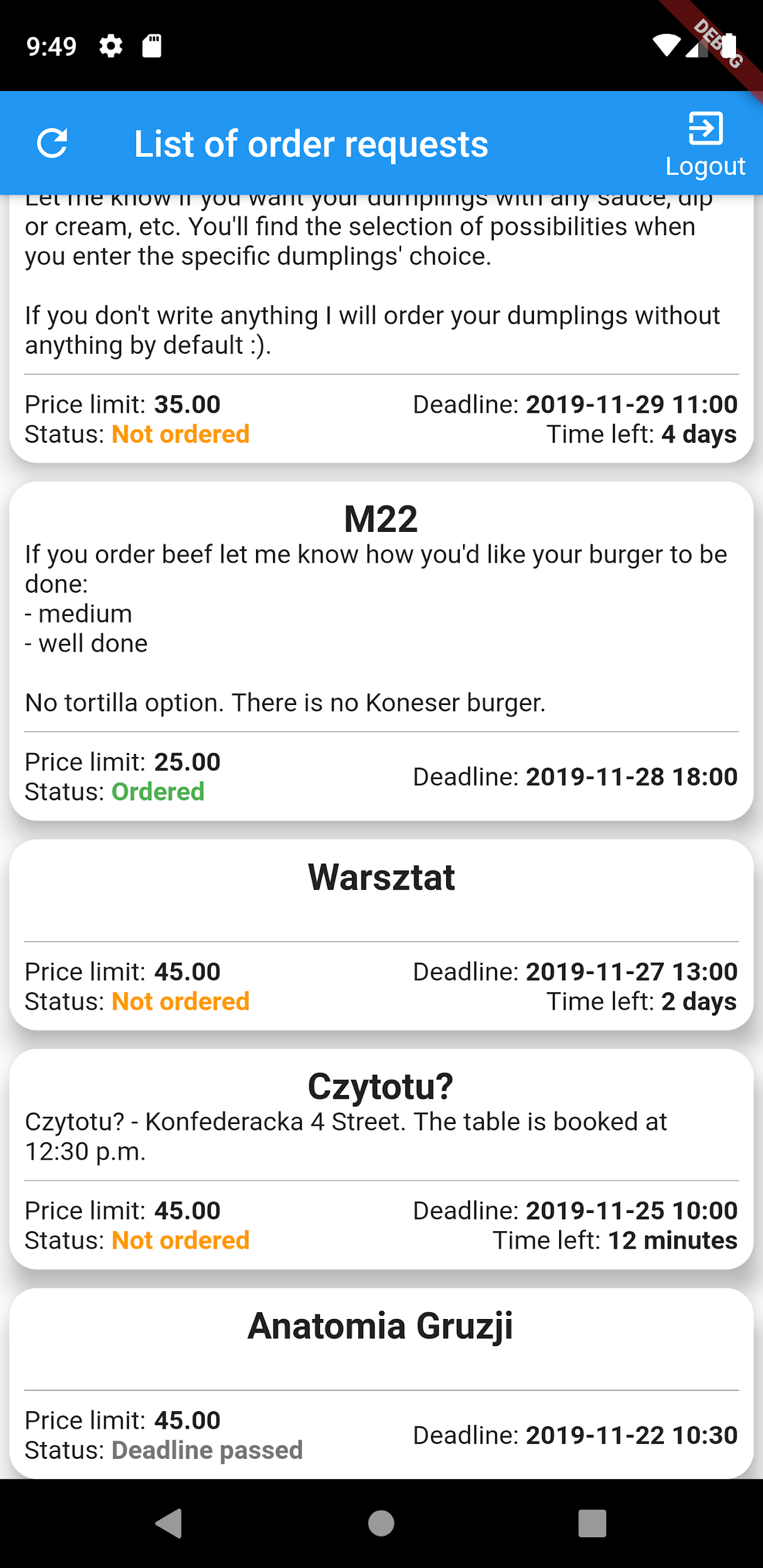
**

Rysunek 8.8 Przygotowanie nagłówka autoryzacji

### Przeglądanie zleceń złożenia zamówienia jednostkowego

Po pomyślnym zalogowaniu jesteśmy przenoszeni do głównego widoku, który wyświetla listę wszystkich zleceń złożenia zamówienia jednostkowego. Zaimplementowana jest ona w postaci przy pomocy *ListView*, którego elementy dostarczane są poprzez strumień o nazwie *orderRequest*. Strumień ten zarządzany jest przez obiekt *OrderRequestBloc*, który dostarcza do widoku wszystkie potrzebne dane. Dzięki wykorzystaniu wzorca BLoC, strumieni oraz widgetu o nazwie *StreamBuilder,* UI aplikacji reaguje natychmiastowo na zmiany wyświetlanych danych. Dodatkowo pozwala to na uzyskanie separacji kodu odpowiedzialnego za wygląd od tego odpowiedzialnego za logikę programu (w tym zapytań do serwera).

*OrderRequestBloc* wywołuje zapytanie do serwera z prośbą o dostarczenie wszystkich zleceń złożenia zamówienia jednostkowego dla aktualnego użytkownika, zwracana wartość ma typ *Future*. Poprzez zastosowanie *Future*, możliwe jest asynchroniczne wykonywanie aplikacji, dzięki czemu interfejs jest nadal responsywny podczas oczekiwania na odpowiedź. W przypadku gdyby nie zastosowano tego rozwiązania, użytkownik miałby wrażenie, że program zawiesił się, aż do momentu otrzymania odpowiedzi od serwera. Odpowiedź zostanie więc dostarczona w późniejszym czasie, a *OrderRequestBloc* zadba o to, by w momencie pojawienia się danych, zostały one przekazane do StreamSink, skąd powędrują prosto do strumienia który nasłuchiwany jest w UI. Interfejs użytkownika wyświetlający listę zleceń złożenia zamówienia nasłuchuje wspomniany strumień. Robi to przy pomocy widgetu *StreamBuilder*. Dzięki zastosowaniu tego komponentu, UI może reagować na zmiany w nasłuchiwanych strumieniach natychmiastowo aktualizując wyświetlaną zawartość.



Kliknięcie w obiekt reprezentujący zlecenie zamówienia, które zostało złożone powoduje przeniesienie do widoku wyświetlającego zawartość tego zamówienia. Jeżeli zamówienie nie zostało jeszcze złożone, kliknięcie to przenosi nas na do widoku zamawiania. Natomiast gdy czas na złożenie zamówienia minął, kliknięcie w owy obiekt nie powoduje żadnej reakcji.

Każdy z obiektów wyświetla takie informacje jak:

* nazwa zlecenia
* ostateczny termin złożenia zamówienia
* limit cenowy
* ile pozostało do ostatecznego terminu zamówienia (o ile on jeszcze i zamówienie nie zostało złożone)
* status
* komentarz do zlecenia

To jaki jest status danego zlecenia możemy stwierdzić po informacji wyświetlanej w lewym dolnym rogu każdego zlecenia. Został on pokolorowany w celu łatwiejszej nawigacji.

Rysunek 8.9 Główny widok prezentujący listę zleceń zamówienia.

### Wyświetlanie złożonego zamówienia



Rysunek 8.10 Widok prezentujący złożone zamówienie

Wyświetlanie złożonego zamówienia oparte jest o podobne mechanizmy, jak te opisane w poprzednim rozdziale. Obiekt BLoC wykonuje zapytanie do serwera o zwrócenie wszystkich pozycji w danym zamówieniu, a następnie przy pomocy strumienia zwraca odpowiednie dane, na co widok reaguje automatycznie aktualizując się i wyświetlając otrzymane informacje.

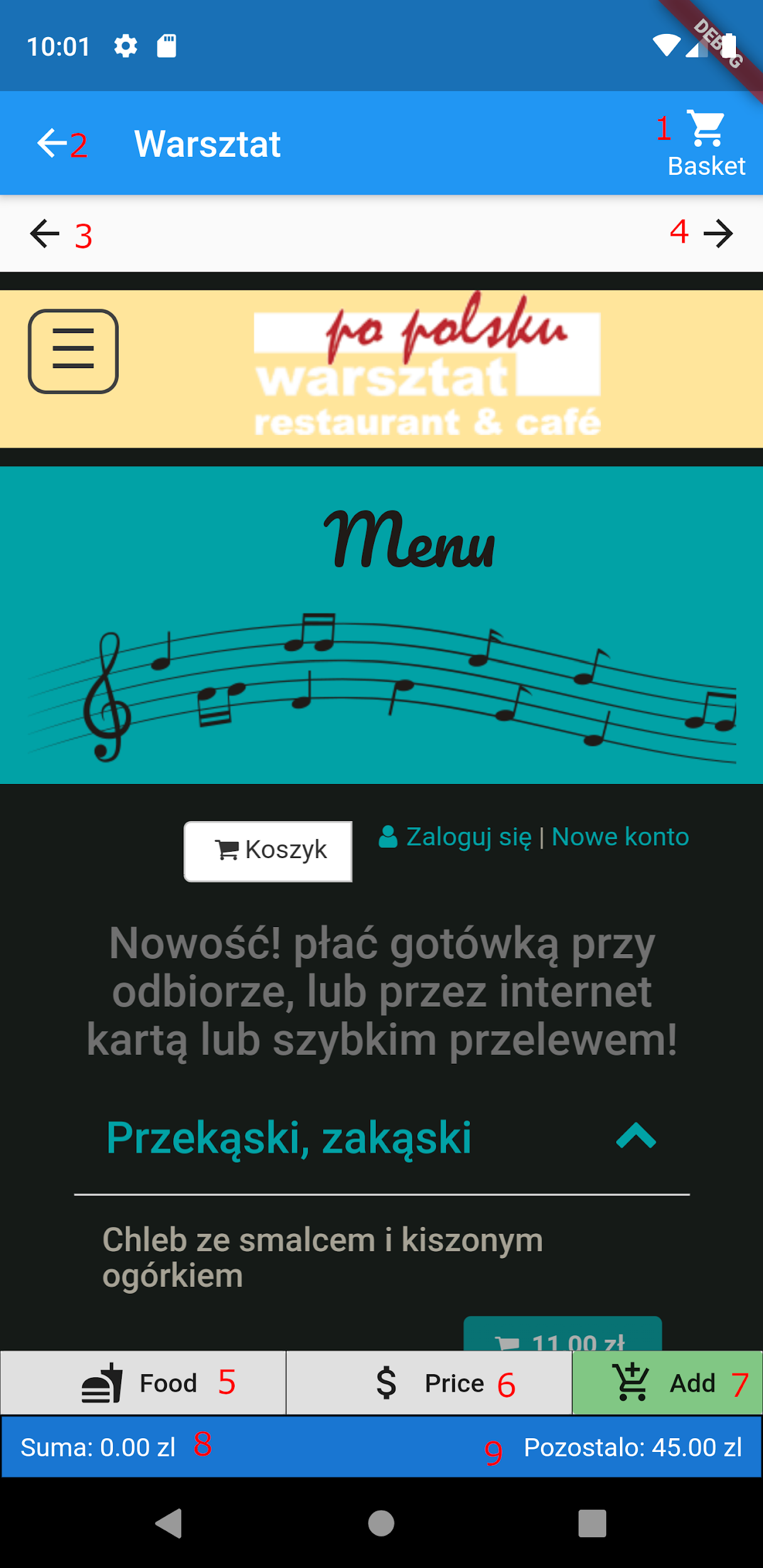
Komentarze do posiłków (o ile je dodano) wyświetlane są w nawiasach pod poszczególnymi pozycjami

Na dole widoku dodano belkę, na której widać sumaryczny koszt zamówienia

### Składanie zamówień

#### Wyświetlanie menu

Po kliknięciu w kartę prezentującą niezłożone zamówienie, jesteśmy przenoszeni do widoku zamawiania. Naszym oczom ukazują się następujące elementy:



Rysunek 8.11 Widok zamawiania posiłku

**1** - przycisk umożliwiający przejście do koszyka, gdzie widzimy wszystkie dodane posiłki,

**2** - przycisk umożliwiający powrót do widoku wyświetlającego wszystkie zlecenia,

**3** - nawigacja przeglądarki - wstecz,

**4** - nawigacja przeglądarki - dalej,

**5** - narzędzie zaznaczania posiłku,

**6** - narzędzie zaznaczania ceny,

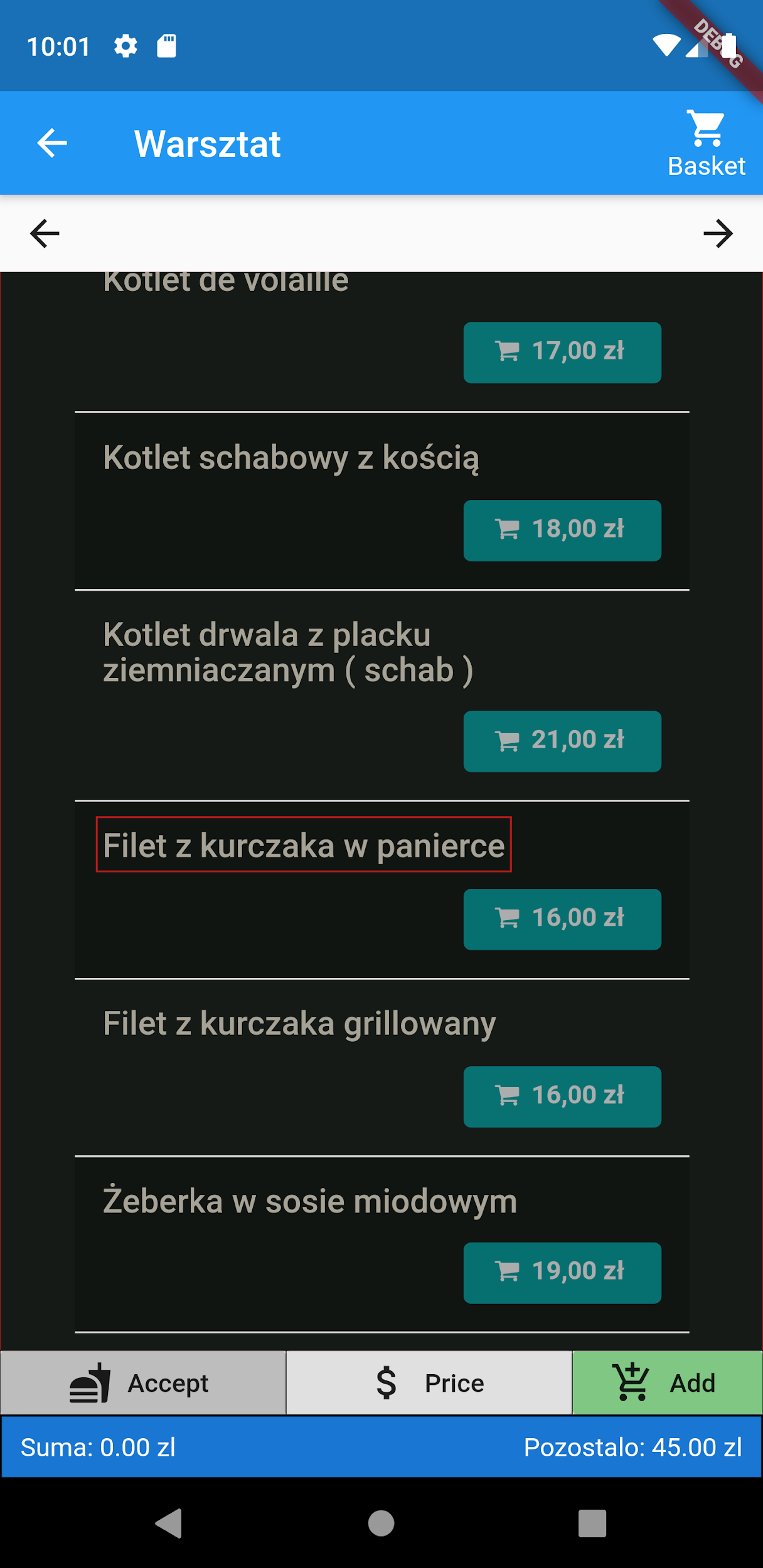
**7** - dodawanie zaznaczonego posiłku do koszyka, po kliknięciu mamy możliwość podania liczby sztuk lub dodania komentarza,

**8** - sumaryczna cena posiłków w koszyku,

**9** - kwota, która pozostała do wydania, obliczona na podstawie aktualnego kosztu koszyka i limitu dla zlecenia,

Menu wyświetlono przy pomocy pluginu *flutter\_inappbrowser*, który udostępnia nam klasę *InAppWebView*, umożliwiającą osadzenie zewnętrznej strony internetowej w naszej aplikacji.

#### Wybieranie posiłków



Wybieranie posiłków możliwe jest do zrealizowania przy pomocy narzędzi, które możemy uruchomić z dolnego paska narzędzi.

Po kliknięciu na przycisk *Food* uruchamia się narzędzie zaznaczania jedzenia. Wybrany posiłek zaznaczamy przeciągając palcem po ekranie tak, aby nazwa znalazła się wewnątrz prostokąta, po czym klikamy *Accept*. Analogicznie postępujemy dla ceny, używając przycisku *Price*. Ostatecznie wszystkie zaznaczone dane zatwierdzamy przyciskiem *Add*.

Rysunek 8.12 Przykład prezentujący moment zaznaczania posiłku

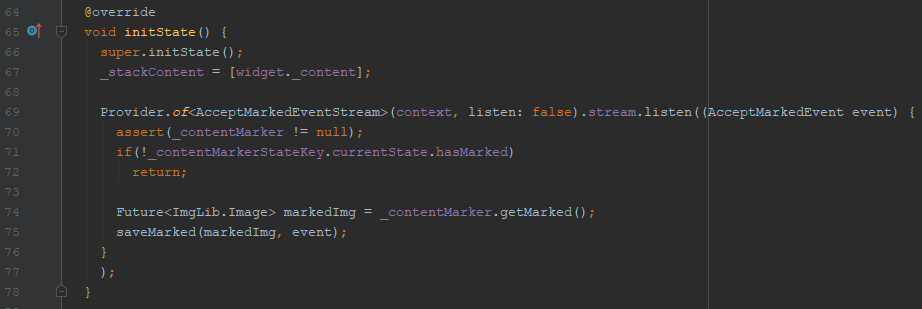
Narzędzie do zaznaczania nazwy posiłków oraz ich cen zostało zaimplementowane przy pomocy utworzonej klasy o nazwie *MarkingManager*



Rysunek 8.13 metoda build klasy MarkingManager

Przy implementacji wspomagano się wbudowanym widgetem o nazwie *Stack*, który umożliwia wyświetlanie widgetów jeden nad drugim. Posiada on zawsze co najmniej jeden element, którym jest widget wyświetlający zawartość strony internetowej prezentującej menu restauracji.

W momencie uruchomienia narzędzia zaznaczania posiłku lub ceny, lista definiująca elementy klasy Stack jest modyfikowana poprzez dodanie na jej koniec (a więc na górę stosu) widgetu o nazwie *ContentMarker*



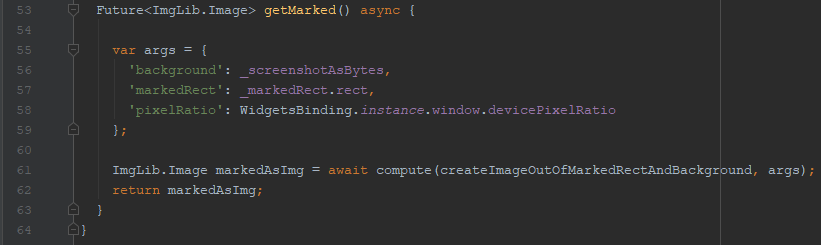
Rysunek 8.14 Metoda initState klasy MarkingManager

Widget wyświetlający menu ustawiany jest jako pierwszy element stosu, poprzez dodanie go do listy *\_stackContent*, która następnie przekazywana jest do widżetu *Stack*, co powoduje, że osadzony jest on najgłębiej.

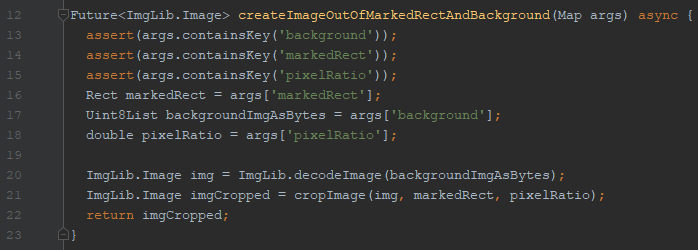
W momencie, gdy tworzony jest stan dla widżetu *MarkingManager*, robiony jest zrzut ekranu przedstawiający stronę internetową, który później jest wyświetlany zamiast prawdziwej strony internetowej. Dzięki temu zyskujemy efekt “zamrożenia”, tego co wyświetla przeglądarka. Skorzystano z takiego rozwiązania, ponieważ czasami strony internetowe zmieniają swój wygląd dynamicznie, co utrudniałoby zaznaczanie posiłków, gdy zawartość strony przesuwałaby się.

*ContentMarker* - jest widgetem, który otrzymuje wcześniej zrobiony zrzut ekranu i na nim wyświetla prostokąt symbolizujący zaznaczony ekran. Posiada on *GestureDetector*, który wykrywa dotknięcia ekranu, dzięki czemu możliwe jest aktualizowanie wielkości zaznaczonego obszaru.

W momencie, gdy użytkownik naciśnie przycisk *Accept* potwierdzający zakończenie zaznaczania. Wywoływana jest metoda *getMarked* widżetu *ContentMarker*. Metoda ta pobiera informacje o zaznaczonym obszarze oraz zrzut ekranu przedstawiający całą “zamrożoną” stronę z menu, a następnie wycina zaznaczony obszar i w celu późniejszego przetworzenia go. Całe zadanie jest dosyć zasobożerne i powodowało zacięcia interfejsu w trakcie jego wykonywania, więc wykonywane ono jest na osobnym *Isolate*, który jest czymś w rodzaju wszechobecnie znanego wszystkim programistom wątku. Umożliwia nam to metoda *compute*, która przyjmuje funkcję do wykonania oraz dokładnie jeden argument, który jest przekazywany do tej funkcji. Ze względu na ograniczenie argumentów, przekazywane one są jako mapa, a następnie przesyłane do funkcji pomocniczej *createImageOutOfMarkedRectAndBackground*, która przyjmuje tylko jeden argument.



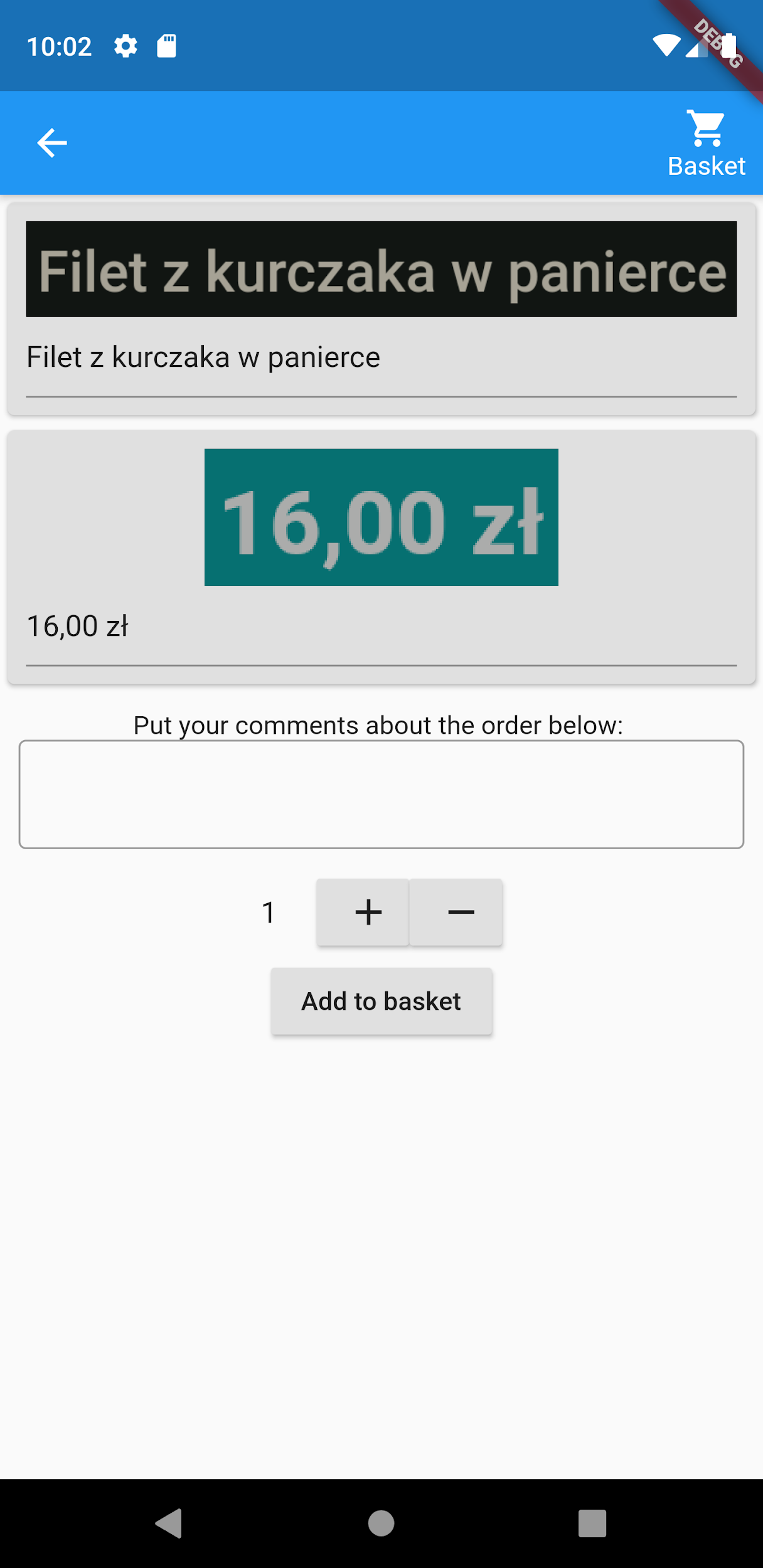
Rysunek 8.15 Metoda getMarked klasy ContentMarker

**

Rysunek 8.16 Metoda pomocnicza używana w metodzie getMarked()

Po otrzymaniu wyciętych części zrzutów ekranu przedstawiających tylko zaznaczone obszary, klient mobilny przesyła je do serwera, który przy pomocy biblioteki OCR o nazwie *PyTesseract* przetwarza je i zwraca rozpoznany tekst na każdym z nich.

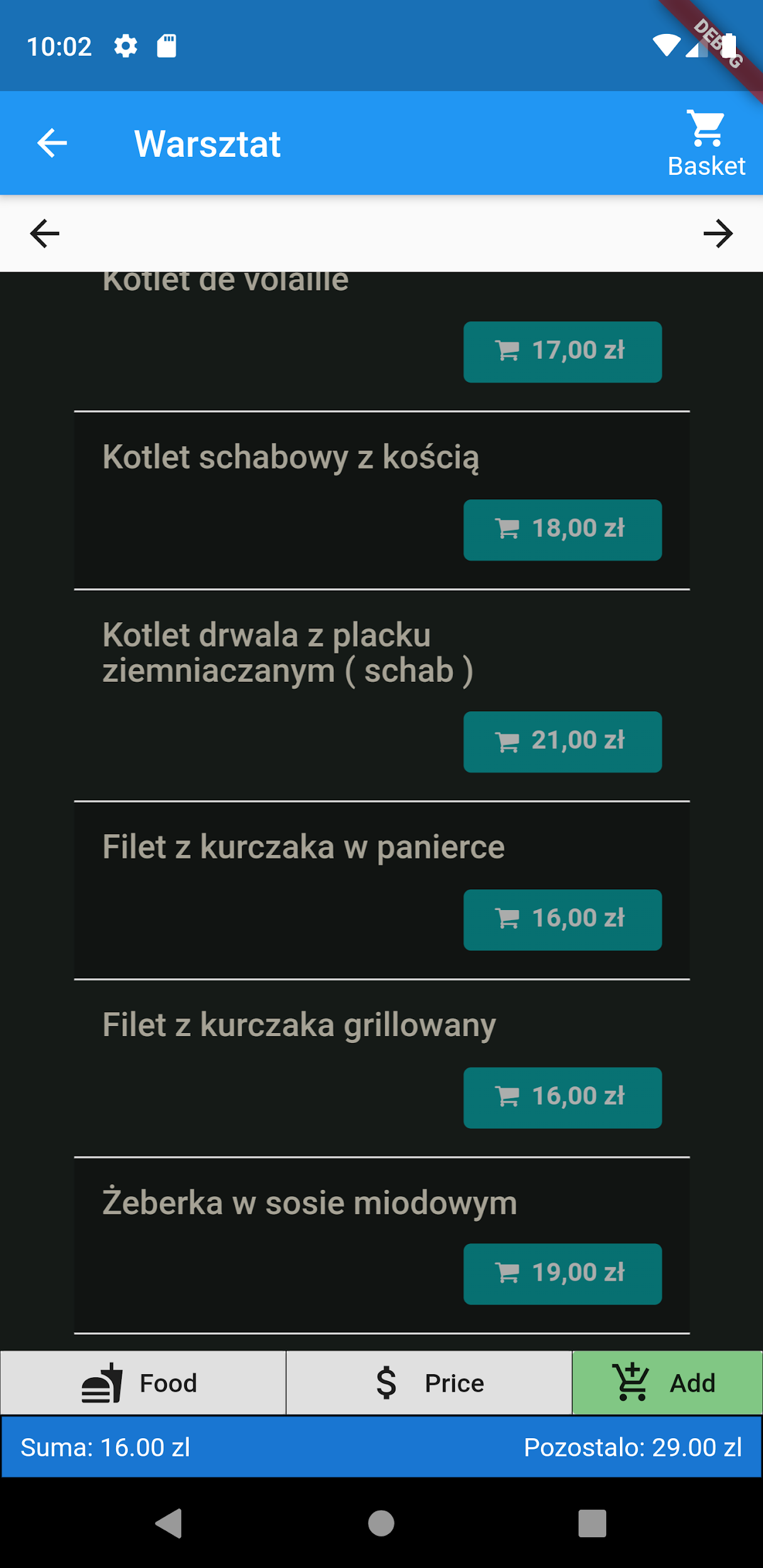
#### Korekta OCR

Po zaznaczeniu nazwy posiłku oraz jego ceny i kliknięciu w przycisk “Add” na dolnym pasku, przenoszeni jesteśmy do okna dodawania posiłku.

Rysunek 8.17 Widok dodawania posiłku

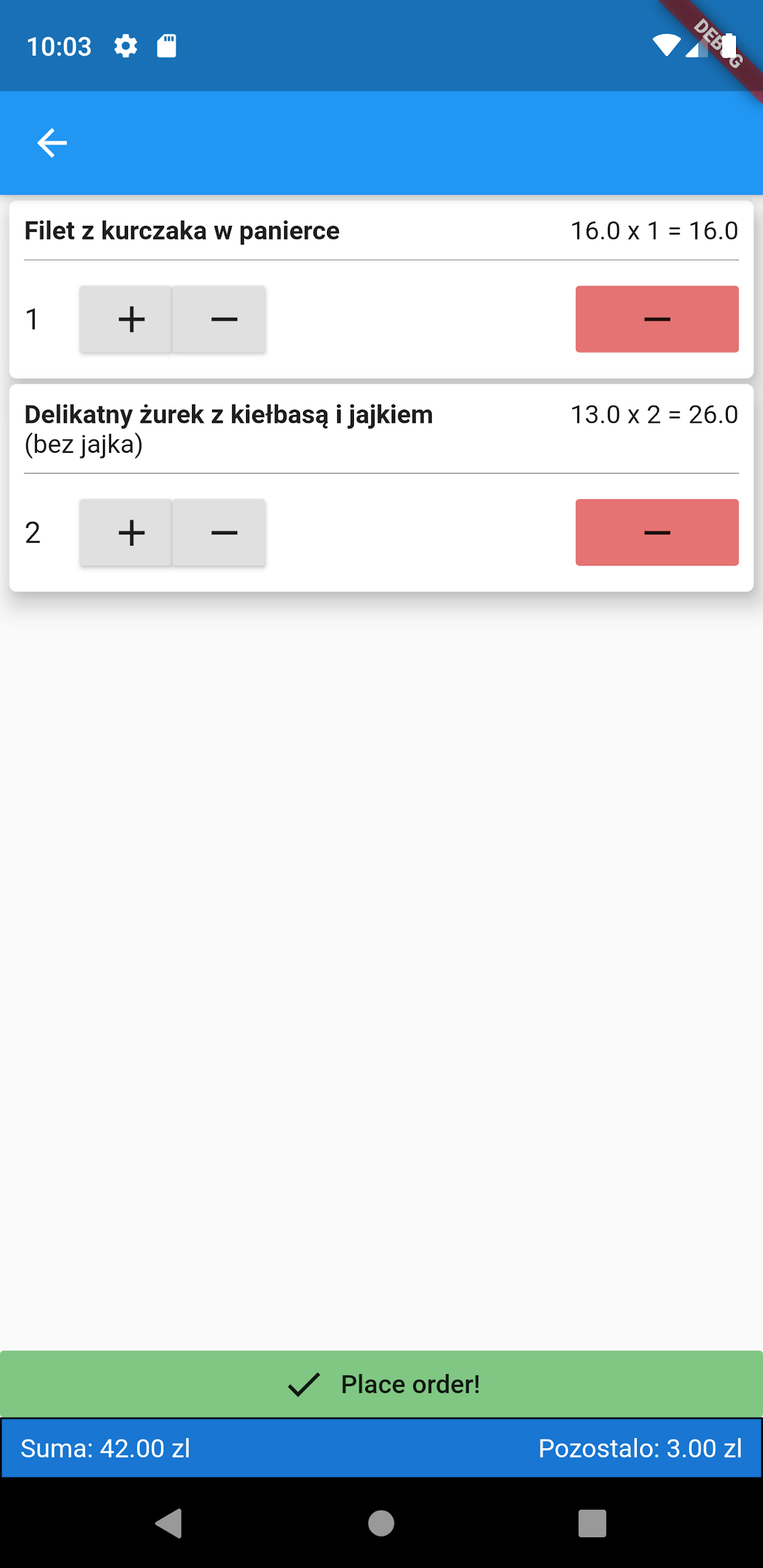
Z poziomu tego widoku mamy możliwość dodania komentarza lub zmiany ilości sztuk. Ponadto możemy zauważyć tutaj wcześniej zaznaczone obszary, a pod każdym z nich, tekst w nich zawarty. Z powodu, że narzędzie OCR czasami ma problemy z rozpoznawaniem tekstu gdy użyto bardziej niestandardowych czcionek, użytkownik ma możliwość ręcznej edycji pól tekstowych i poprawienia tekstu, gdy został on źle odczytany.  
  
Po wprowadzeniu wszystkich potrzebnych modyfikacji i kliknięciu w przycisk *Add to basket*, posiłek dodawany jest do koszyka i następuje przekierowanie do strony z menu, gdzie mamy możliwość dodania kolejnych posiłków.

Na stronie tej możemy zauważyć, że dolna belka zaktualizowała dane o stanie koszyka



Rysunek 8.18 Dolna belka przedstawiająca informacje o stanie aktualnego zamówienia

#### Koszyk

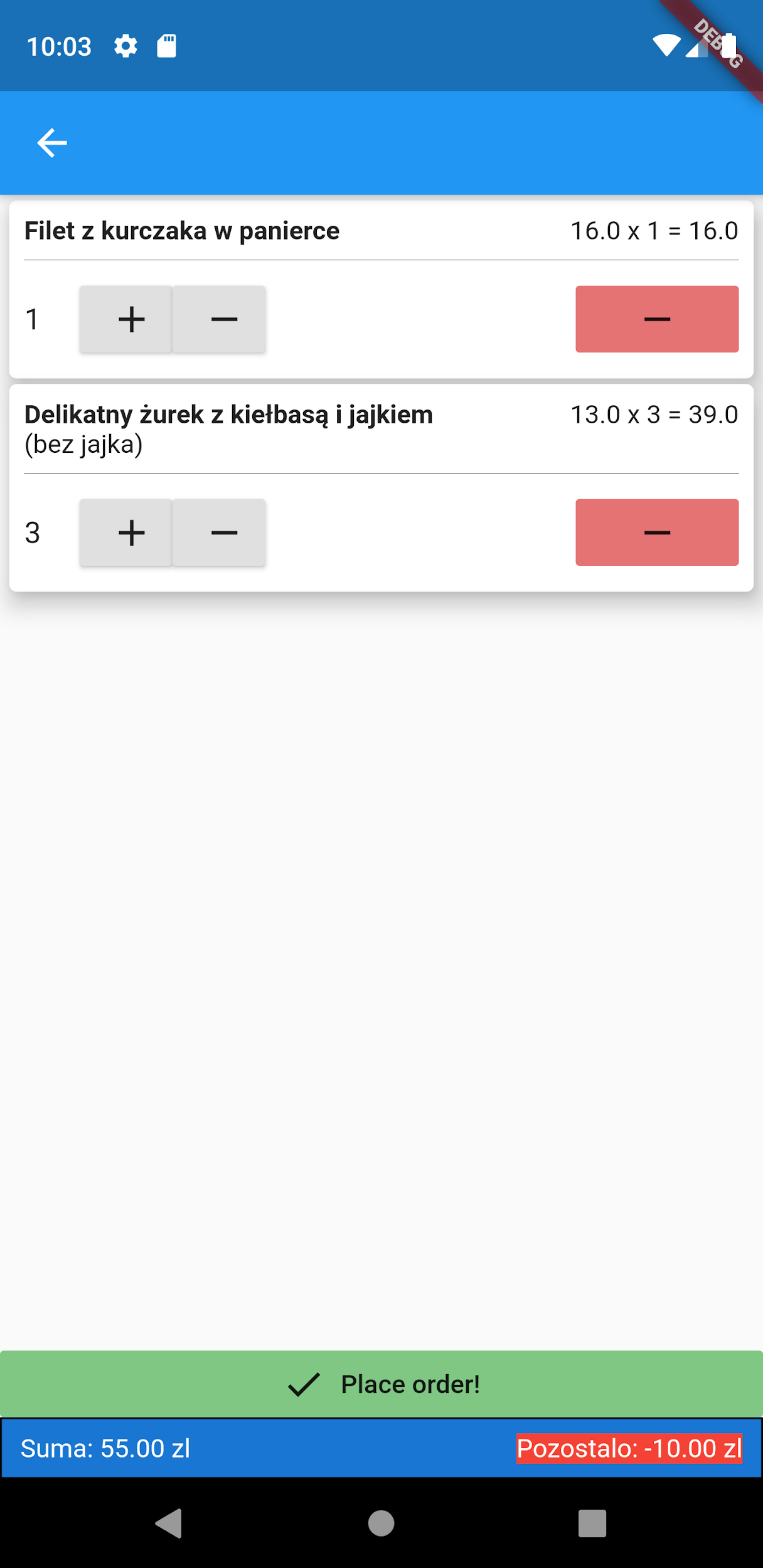


Rysunek 8.19 Koszyk z zamówieniami

Po dodaniu kolejnego posiłku do koszyka prezentuje się tak jak na załączonym zrzucie ekranu.

Z poziomu koszyka mamy możliwość aktualizacji ilości danego posiłku lub całkowitego jego usunięcia.

Dodane komentarze są wyświetlane w nawiasach, co można zauważyć przy drugim posiłku.

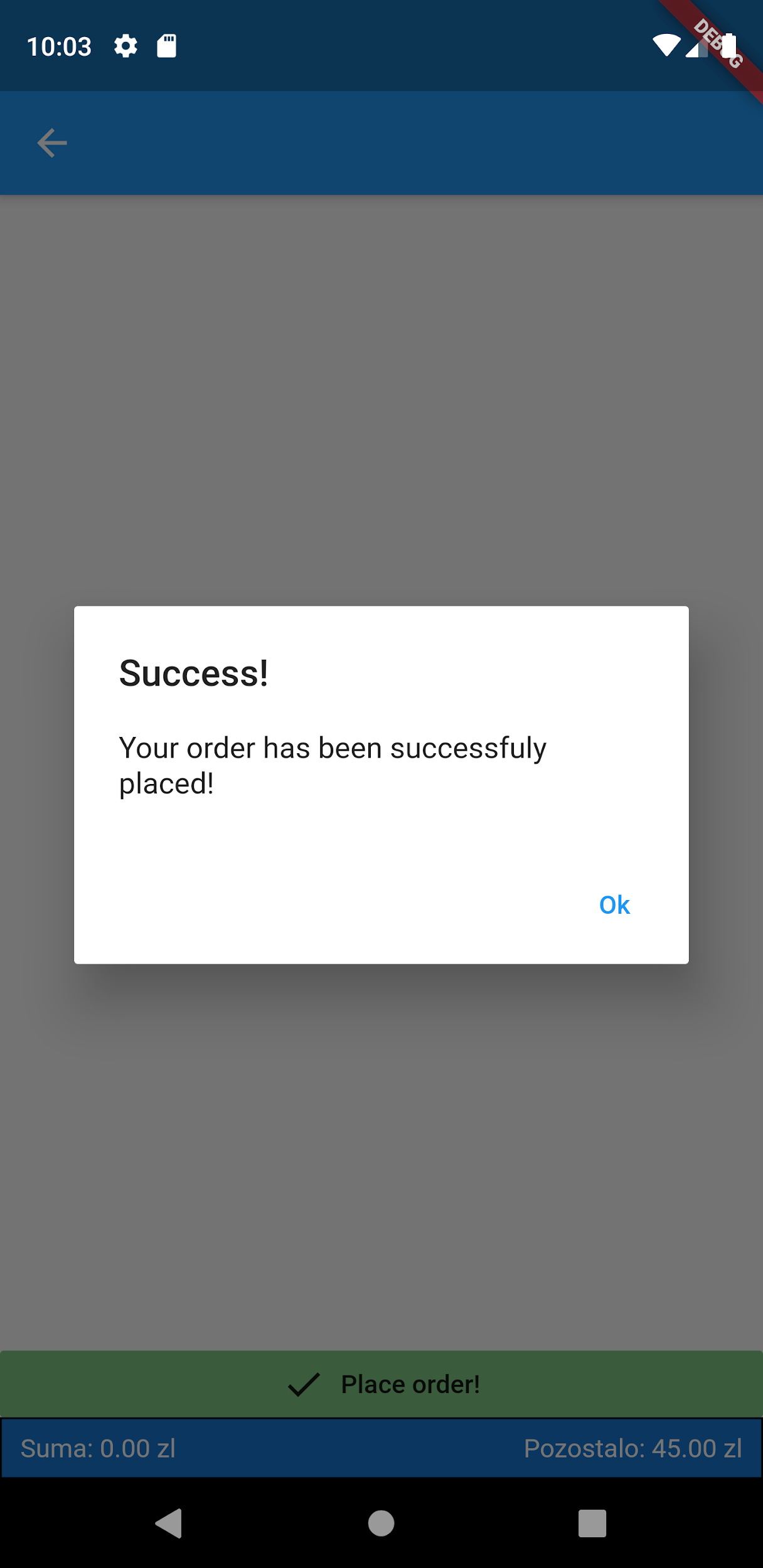


Rysunek 8.20 Widok koszyka z zamówieniami przy przekroczonym limicie cenowym

Wszystkie zmiany wprowadzone z poziomu tego widoku dynamicznie aktualizują wszystkie kalkulacje - np. po zwiększeniu liczby sztuk drugiego posiłku z 2 na 3, możemy zauważyć, że cena obok niego została przeliczona, jak i również cena na dolnej belce.

Ponadto ze względu na przekroczony limit, program zmienia kolor tekstu co nam to sygnalizuje.

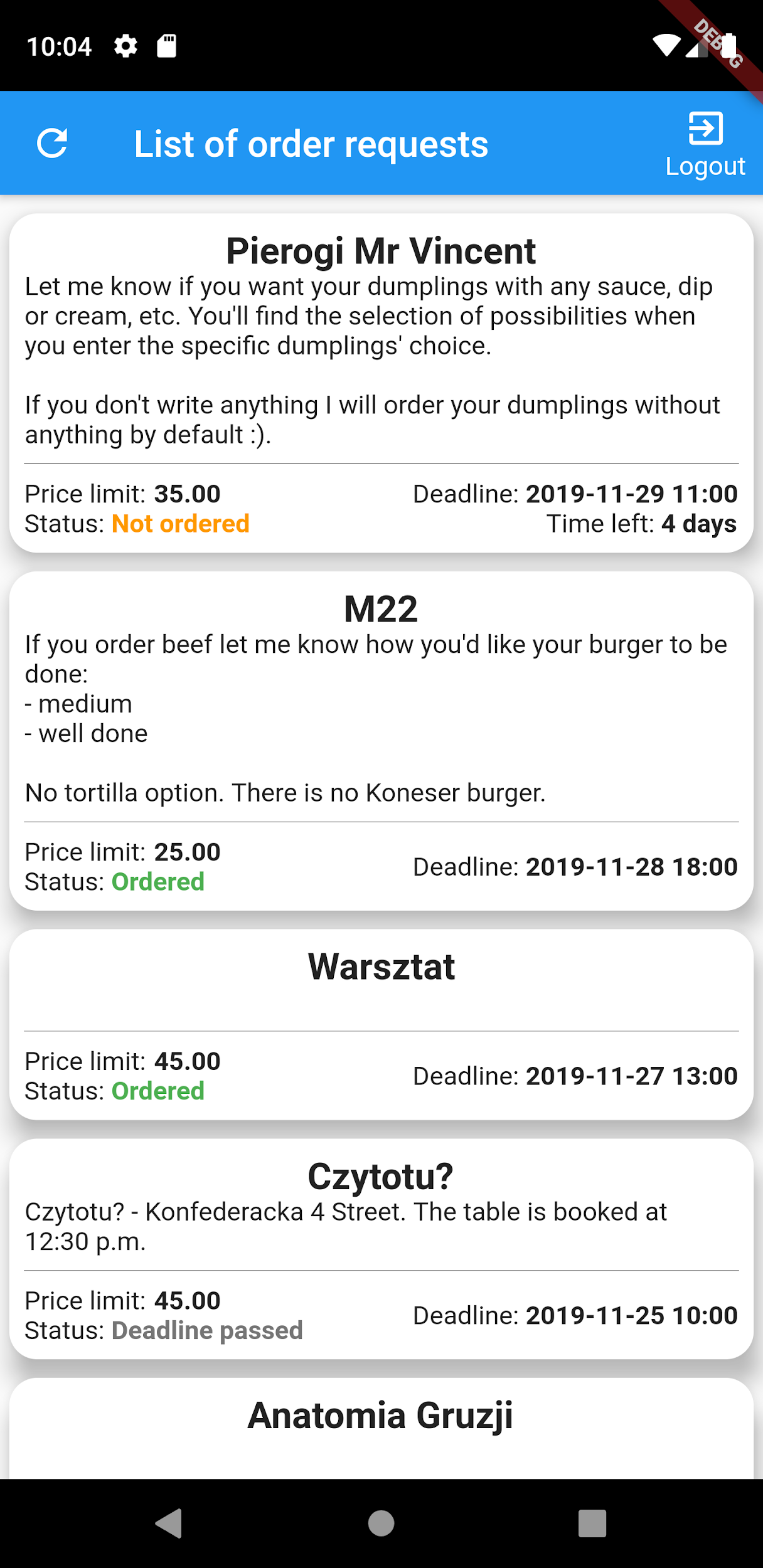
#### Składanie zamówienia



Rysunek 8.21 Komunikat potwierdzający powodzenie żłożenia zamówienia

Jak wspomniano w jednym z poprzednich rozdziałów, przekroczenie limitu ceny nie powoduje braku możliwości złożenia zamówienia. Dokonujemy go klikając w przycisk *Place order!*. Powoduje to wysłanie zamówienia do serwera, który je odpowiednio przetworzy, a na ekranie pojawi się komunikat z informacją o powodzeniu.

Po kliknięciu w przycisk *Ok*, zostajemy przekierowani do strony głównej, która wyświetla wszystkie zlecenia złożenia zamówienia.



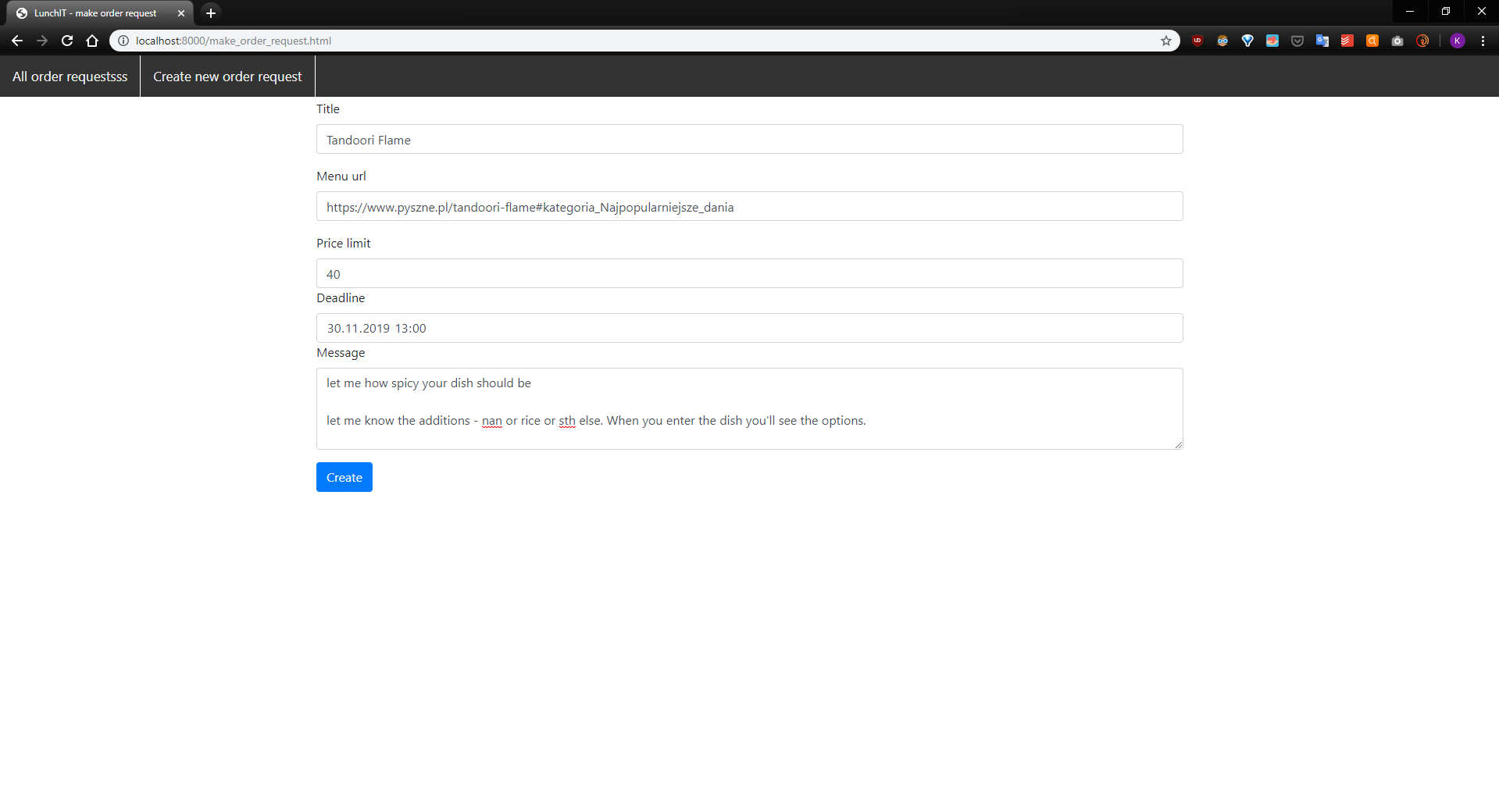
Rysunek 8.22 Zrzut ekranu prezentujący zmianę statusu zamówienia o nazwie “Warsztat”

Jak można zauważyć, status przy złożonym przez nas zamówieniu (trzecia karta od góry – „Warsztat”) zmienił się na „Ordered”

## Aplikacja webowa

### Tworzenie nowego zlecenia

Po przejściu do zakładki *Create new order request*, użytkownik ma możliwość stworzenia nowego zlecenia zamówienia. Pola *Price limit* oraz *Deadline* są polami specjalnymi, które umożliwiają wprowadzenie tylko konkretnych typów wartości. Po uzupełnieniu wszystkich pól oraz kliknięciu w przycisk *Create* następuje wysłanie zapytania do serwera, w celu utworzenia odpowiedniego wpisu w bazie danych, a użytkownik zostaje przeniesiony do strony wyświetlających wszystkie zlecenia, gdzie może zobaczyć również to nowo utworzone.

**

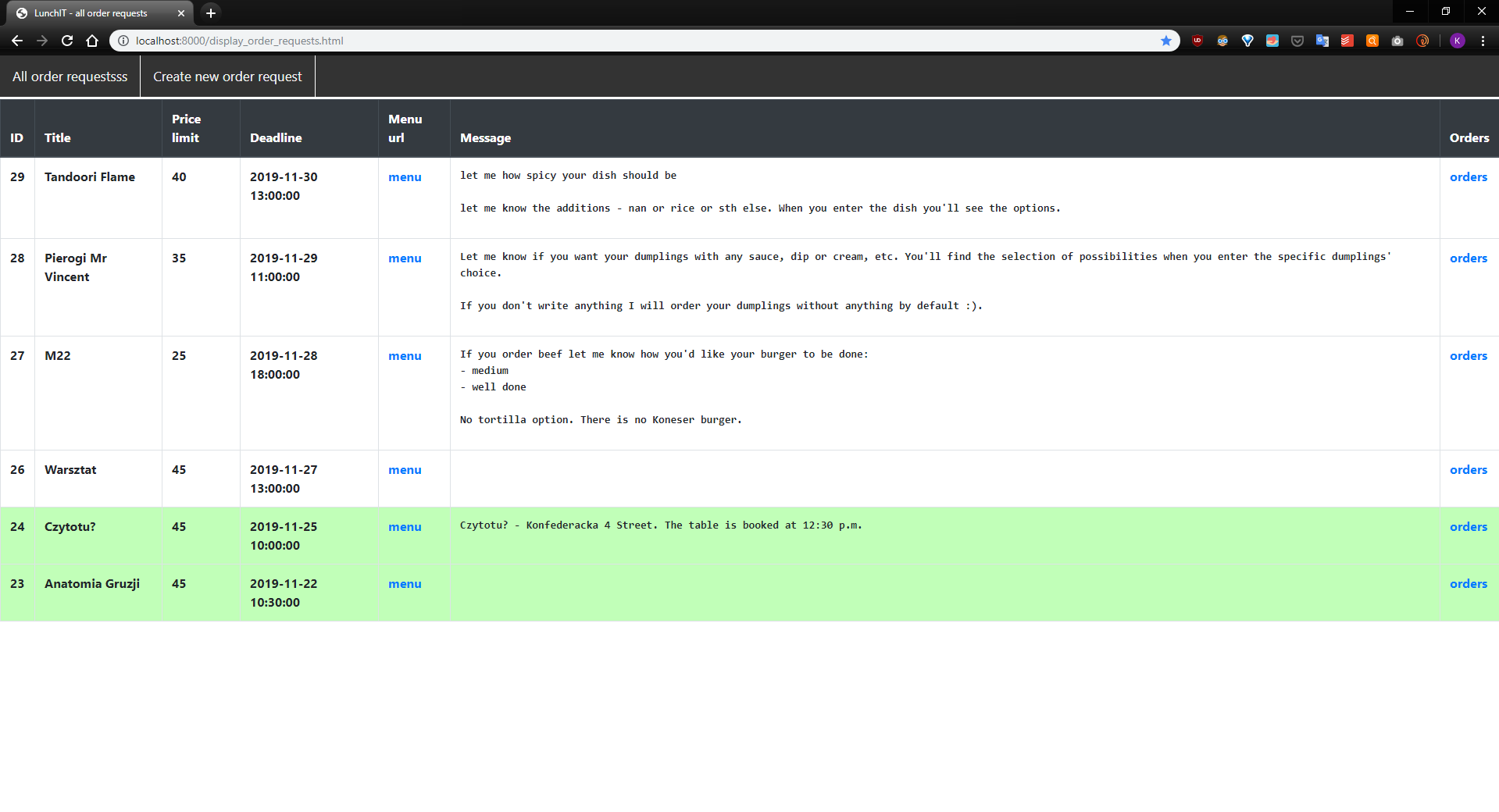
Rysunek 8.23 Widok podstrony z tworzeniem nowych zleceń

### Wyświetlanie listy zleceń

Po przejściu do zakładki *All order requests* wyświetlone zostają wszystkie zlecenia złożenia zamówienia. Dla wygody zostały one posortowane malejąco względem kolumny *Deadline,* a więc zamówienia, których ostateczny termin zamówienia do którego zostało najwięcej czasu zostanie wyświetlony na samej górze listy.

Pozycje na liście, których ostateczny czas na złożenie zamówienia minął kolorowane są na zielono, w celu łatwiejszego odnalezienia pozycji, które są gotowe do zamówienia telefonicznego.

W celu przejścia do strony wyświetlającej zamówienie grupowe należy kliknąć w odnośnik *orders* przy odpowiednim wierszu.



Rysunek 8.24 Widok podstrony wyświetlającej wszystkie zlecenia

Strona ta została zaimplementowana przy pomocy tabeli, która wygląd swój bazuje na zewnętrznej bibliotece CSS o nazwie *Bootstrap*.

W kodzie HTML na samym początku zadeklarowany jest kontener o id=”*navbar*”, który później zostanie wypełniony paskiem nawigacji załadowanym z osobnego pliku. Zastosowano takie rozwiązanie, aby uniknąć powtórzeń w każdej z podstron i łatwiejszym dodawaniu nowych podstron do paska.

Następnie tworzona jest główna struktura tabeli. Deklarowane tam są nazwy nagłówków, oraz nazwy, które nadają tabeli odpowiedni wygląd.

**

Rysunek 8.25 Deklaracja tabeli wyświetlającej zlecenia zamówień

W części skryptowej strony, w momencie gdy cała strona zostanie załadowana, na początku ładowany jest pasek nawigacji z pliku *navbar.html*. Następnie wykonywane jest zapytanie do serwera, które zwraca dane o wszystkich zleceniach zamówień. Ostatecznie dane te są w odpowiedni sposób formatowane i umieszczane we wcześniej zadeklarowanej tabeli. Cała operacja jest wspomagana biblioteką *jQuery*, która ułatwia to zadanie.

**

Rysunek 8.26 Część skryptowa strony display\_order\_requests.html, która pobiera dane z serwera i ładuje je do tabeli

### Wyświetlanie zamówienia grupowego

W celu zaprezentowania wyświetlenia zamówienia grupowego najpierw złożono trzy zamówienia jednostkowe (każde z innego konta). Zamówienia te prezentują się następująco:

**

Rysunek 8.27 Zamówienie jednostkowe - użytkownik nr 1

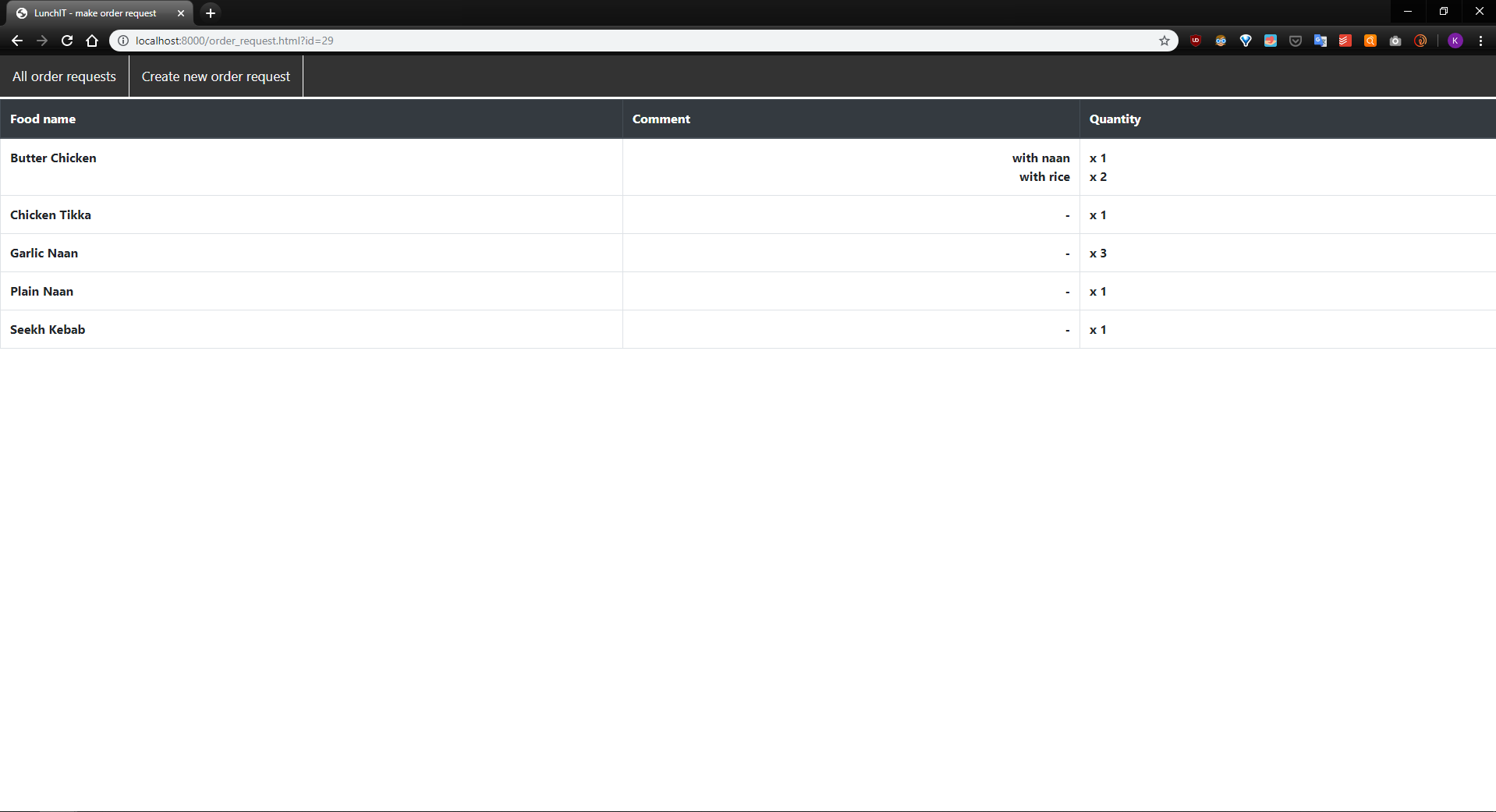
**

Rysunek 8.28 Zamówienie jednostkowe - użytkownik nr 2

**

Rysunek 8.29 Zamówienie jednostkowe - użytkownik nr 3

Zamówienia o takiej samej nazwie i komentarzu są grupowane w jedną pozycję, a ich ilość jest odpowiednio zwiększana. Dlatego ze względu że każdy z użytkowników złożył zamówienie na Garlic Naan i nie dodał żadnego komentarza, na liście grupowej widnieje on tylko raz, a ilość jego sztuk jest równa 3. Zgrupowane również zostało zamówienie na posiłek no nazwie Butter Chicken, gdzie dodany komentarz to “with rice”.



Rysunek 8.30 Zamówienie grupowe wygenerowane z wcześniej przedstawionych zamówień jednostkowych

Implementacja wyświetlania strony wyświetlającej zamówienia grupowe została zaimplementowana w bardzo podobny sposób jak strony wyświetlająca listę wszystkich zleceń zamówień, której implementacja została omówiona w poprzednim podrozdziale.

## Serwer - dokumentacja zasobów

Wszystkie zasoby zwracające odpowiedź w formie obiektu JSON mogą zawierać klucz *error* o typie string. Jest on dostępny wyłącznie gdy wystąpił błąd. Wartość znajdująca się pod tym kluczem zawiera informację opisującą napotkany problem.

Dostęp do zasobów wymagających autoryzacji można uzyskać poprzez dodanie nagłówka o nazwie *AUTHORIZATION,* którego wartość przyjmuje postać *login:hasło*, gdzie hasło to wartość zwrócona z funkcji haszującej *PBKDF2*, do której przekazano hasło użytkownika

### Autentykacja

**opis**: służy do sprawdzenia poprawności danych logowania

**zasób**: /api/authenticate

**metoda**: POST

**wymaga autoryzacji**: nie

**parametry**: -

**typ body zapytania:** JSON

**klucze zapytania:**

* [*string*] - user\_id - id użytkownika
* [*string*] - hashed\_password - hasło zahaszowane funkcją haszującą *PBKDF2*

**przykładowe body**:

{

"user\_id": "username@gmail.com",

"hashed\_password": "$pcks$64,10000,64$530f8afbc74536b9a963b4f1c4cb738bcea7403d4d606b6e074ec5d3baf39d18$5291aa80d1722d0b3c0ab8e3737bba68d1714267dc785191ffcf91e594a162526d4554d4720c1900cfc1366b550b3159addccf2f132b9ce2536d1735b4432835"

}

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*bool*] authenticated - informuje o tym, czy podane dane logowania użytkownika są poprawne

**przykład odpowiedzi**:

{

"authenticated": false

}

### Tworzenie nowego użytkownika

**opis**: umożliwia stworzenie nowego użytkownika

**zasób**: /api/create\_account

**metoda**: POST

**wymaga autoryzacji**: nie

**parametry**: -

**typ body zapytania:** JSON

**klucze zapytania:**

* [*string*] - user\_id - identyfikator użytkownika będący adresem email
* [*string*] - hashed\_password - hasło zahaszowane funkcją haszującą *PBKDF2*

**przykładowe body**:

{

"user\_id": "username@gmail.com",

"hashed\_password": "$pcks$64,10000,64$530f8afbc74536b9a963b4f1c4cb738bcea7403d4d606b6e074ec5d3baf39d18$5291aa80d1722d0b3c0ab8e3737bba68d1714267dc785191ffcf91e594a162526d4554d4720c1900cfc1366b550b3159addccf2f132b9ce2536d1735b4432835"

}

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*string*] status - przyjmuje wartości “created” oraz “not created”

**przykład odpowiedzi**:

{

"status": "created"

}

### Grupowe zamówienie

**opis**: służy do pobrania grupowego zamówienia stworzonego poprzez scalenie wszystkich zamówień jednostkowych należących do danego zlecenia

**zasób**: /api/group\_order

**metoda**: GET

**wymaga autoryzacji**: nie

**parametry**: -

**typ body zapytania:** -

**klucze zapytania:** -

**przykładowe body**: -

**typ odpowiedzi**: JSON

**opis odpowiedzi:** zwraca obiekt JSON posiadający klucze będące nazwami posiłków, każdy z posiłków jest obiektem JSON posiadającym klucze będące komentarzami do posiłków, a wartości przypisane do tych kluczy to liczby całkowite reprezentujące liczbę osób, która złożyła zamówienie o danej nazwie i z danym komentarzem . W przypadku braku komentarza, wartość klucza jest równa “-”.

**przykład odpowiedzi**:

{

"Butter Chicken": {

"with naan": 1,

"with rice": 2

},

"Chicken Tikka": {

"-": 1

},

"Garlic Naan": {

"-": 3

},

"Plain Naan": {

"-": 1

},

}

### Sprawdzanie czy użytkownik złożył zamówienie

**opis**: służy do sprawdzenia czy użytkownik wykonujący zapytanie złożył zamówienie do zlecenia identyfikowanego przez parametry zapytania

**zasób**: /api/has\_ordered

**metoda**: GET

**wymaga autoryzacji**: tak

**parametry**:

* [*int*] order\_request\_id - id identyfikujące zlecenie złożenia zamówienia

**typ body zapytania:** -

**klucze zapytania:** -

**przykładowe body**: -

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*bool*] has\_ordered - informacja czy użytkownik złożył zamówienie

**przykład odpowiedzi**:

{

"has\_ordered": false

}

### Odczytywanie tekstu z obrazka

**opis**: służy do odczytania tekstu znajdującego się na obrazku

**zasób**: /api/image\_to\_text

**metoda**: POST

**wymaga autoryzacji**: nie

**parametry**: -

**typ body zapytania:** form-data

**klucze zapytania:**

* [*pik png, jpg lub jpeg*] file - obraz zawierający tekst, który chcemy odczytać

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*string*] text - łańcuch znaków odczytany z obrazka

**przykład odpowiedzi**:

{

"text": "sample recognized text"

}

### Składanie zamówienia jednostkowego

**opis**: umożliwia złożenie zamówienia jednostkowego

**zasób**: /api/order

**metoda**: POST

**wymaga autoryzacji**: tak

**parametry**: -

**typ body zapytania:** JSON

**klucze zapytania:**

* [*int*] orderRequestId - id zlecenia, dla którego składamy zamówienie jednostkowe
* [*JSON basket*] - basketData - lista obiektów [*JSON meals*]
  + [*JSON meals*] - obiekt reprezentujący posiłek, posiada następujące klucze:
    - [string] foodName - nazwa posiłku,
    - [float] price - cena,
    - [int] quantity - ilość sztuk,
    - [string] comment – komentarz (pole opcjonalne)

**przykładowe body**:

{

"orderRequestId": "31",

"basketData":

[

{

"foodName": "test",

"price": "20",

"quantity": "1",

"comment": ""

}

]

}

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*bool*] status - informacja czy udało się złożyć zamówienie, przyjmuje wartości:
  + "*Failure, order already placed!"* - gdy zamówienie zostało już wcześniej złożone
  + *"Success, order has been placed."* - gdy składanie zamówienia przebiegło pomyślnie

**przykład odpowiedzi**:

{

"status": "Failure, order already placed!"

}

### Sprawdzanie zamówienia jednostkowego

**opis**: umożliwia pobranie posiłków znajdujących się w zamówieniu jednostkowym

**zasób**: /api/order

**metoda**: GET

**wymaga autoryzacji**: tak

**parametry**:

* [*int*] placed\_order\_id - id identyfikujące zamówienie jednostkowe

**typ body zapytania:** -

**klucze zapytania:** -

**przykładowe body**: -

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*JSON object*] order - lista posiłków w postaci obiektu [*JSON meals*],
  + [*JSON meals*] - obiekt reprezentujący posiłek, posiada następujące klucze:
    - [*string*] foodName - nazwa posiłku,
    - [*float*] price - cena,
    - [*int*] quantity - liczba sztuk,
    - [*string/null*] comment - komentarz lub null(gdy brak komentarza)

**przykład odpowiedzi**:

{

"order":

[

{

"comment": null,

"food\_name": "Garlic Naan",

"price": 10,

"quantity": 1

},

{

"comment": "with naan",

"food\_name": "Butter Chicken",

"price": 26,

"quantity": 1

}

]

}

### Utworzenie nowego zlecenia

**opis**: umożliwia utworzenie nowego zlecenia złożenia zamówień jednostkowych

**zasób**: /api/order\_request

**metoda**: POST

**wymaga autoryzacji**: nie

**parametry**: -

**typ body zapytania:** JSON

**klucze zapytania:**

* [*float*] price\_limit - limit pieniężny dla zamówień jednostkowych
* [*string*] title - tytuł zlecenia
* [*date*] deadline - ostateczny termin złożenia zamówienia jednostkowego, gdzie format to: YYYY-MM-DDTHH:MM,
  + YYYY - rok
  + MM - miesiąc
  + DD - dzień
  + T - separator (należy wstawić dokładnie tę literę)
  + HH - godzina
  + MM - miesiąc

**przykładowe body**:

{

"price\_limit": "25",

"title": "Restaurant name",

"deadline": "2020-03-22T22:50",

"message": "test message",

"menu\_url": "http://restaurant.com/menu"

}

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*int*] order\_request\_id - id identyfikujące utworzone zlecenie

**przykład odpowiedzi**:

{

"order\_request\_id": 44

}

### Wszystkie zlecenia

**opis**: umożliwia pobranie wszystkich zleceń wraz z ich parametrami

**zasób**: /api/order\_request

**metoda**: GET

**wymaga autoryzacji**: nie

**parametry**: -

**typ body zapytania:** -

**klucze zapytania:** -

**przykładowe body**: -

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*list*] order\_requests - lista obiektów [*JSON* *request*]
  + [*JSON request*] - obiekt JSON posiadający następujące klucze:
    - [*float*] price\_limit - limit pieniężny dla zamówień jednostkowych
    - [*string*] title - tytuł zlecenia
    - [*date*] deadline - ostateczny termin złożenia zamówienia, gdzie format to: YYYY-MM-DDTHH:MM,
      * YYYY - rok
      * MM - miesiąc
      * DD - dzień
      * T - separator (dokładnie ten znak)
      * HH - godzina
      * MM - miesiąc

**przykład odpowiedzi**:

{

"order\_requests":

[

{

"deadline": "2019-11-30 13:00:00",

"id": 29,

"menu\_url": "https://www.pyszne.pl/tandoori-flame#kategoria\_Najpopularniejsze\_dania",

"message": "let me how spicy your dish should be\n \nlet me know the additions - nan or rice or sth else. When you enter the dish you'll see the options.",

"name": "Tandoori Flame",

"price\_limit": 40

},

{

"deadline": "2019-11-22 10:30:00",

"id": 23,

"menu\_url": "https://www.pyszne.pl/anatomia-gruzji",

"message": "",

"name": "Anatomia Gruzji",

"price\_limit": 45

}

]

}

### Wszystkie zlecenia złożenia zamówień jednostkowych wraz z identyfikatorem złożonego zamówienia

**opis**: umożliwia pobranie wszystkich zleceń wraz z ich parametrami oraz dodatkowym identyfikatorem zamówienia złożonego przez użytkownika wykonującego zapytanie.

**zasób**: /api/user\_order\_requests

**metoda**: GET

**wymaga autoryzacji**: tak

**parametry**: -

**typ body zapytania:** -

**klucze zapytania:** -

**przykładowe body**: -

**typ odpowiedzi**: JSON

**klucze odpowiedzi:**

* [*list*] order\_requests - lista obiektów [*JSON request*]
  + [*JSON request*] - obiekt JSON posiadający następujące klucze:
    - [*float*] price\_limit - limit pieniężny dla zamówień jednostkowych
    - [*string*] title - tytuł zlecenia
    - [*date*] deadline - ostateczny termin złożenia zamówienia, gdzie format to: YYYY-MM-DDTHH:MM,
      * YYYY - rok
      * MM - miesiąc
      * DD - dzień
      * T - separator (dokładnie ten znak)
      * HH - godzina
      * MM - miesiąc
    - [*int/null*] placed\_order\_id - id identyfikujące złożone przez użytkownika zamówienie jednostkowe powiązane z danym zleceniem. Jeżeli użytkownik nie złożył jeszcze zamówienia, pole to przyjmuje wartość *null.*

**przykład odpowiedzi**:

{

"order\_requests": [

{

"deadline": "2019-11-30 13:00:00",

"id": 29,

"menu\_url": "https://www.pyszne.pl/tandoori-lame#kategoria\_Najpopularniejsze\_dania",

"message": "let me how spicy your dish should be.",

"name": "Tandoori Flame",

"price\_limit": 40,

"placed\_order\_id": null

},

{

"deadline": "2019-11-22 10:30:00",

"id": 23,

"menu\_url": "https://www.pyszne.pl/anatomia-gruzji",

"message": "",

"name": "Anatomia Gruzji",

"price\_limit": 45,

"placed\_order\_id": 55

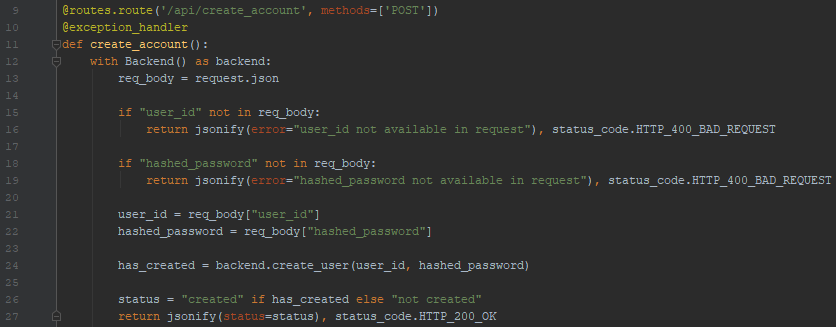
}

]

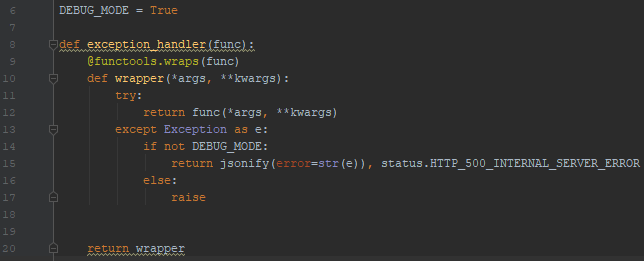
}

## Kody zapytań oraz obsługa wyjątków

Każdy z zasobów zwraca odpowiednie kody odpowiedzi HTTP, dzięki którym można sprawdzić informację o realizacji zapytania. Przykładowo dla zapytania wysłanego w celu złożenia zamówienia otrzymamy kod odpowiedzi *400* w przypadku, gdy ciało zapytania będzie niepoprawne, *200* gdy zapytanie zostanie poprawnie przetworzone lub *500* gdy zostanie rzucony jakiś wyjątek (obsługiwane przez dekorator @*exception\_handler*)



Rysunek 8.31 Implementacja zasobu służącego do stworzenia nowego użytkownika prezentująca zwracane kody odpowiedzi HTTP

**

Rysunek 8.32 Implementacja dekoratora exception\_handler zwracającego kod błędu 500

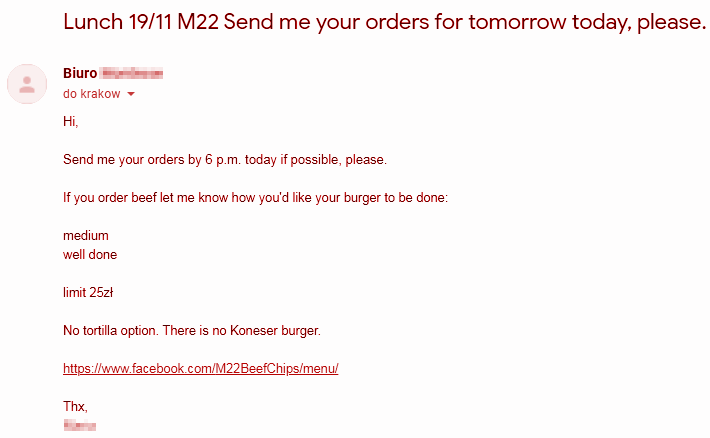
Dekorator *exception\_handler* przechwytuje nieobsłużone wyjątki rzucone w obrębie dekorowanego zasobu oraz zwraca odpowiedź w postaci obiektu JSON posiadającego klucz *error*. Wartość, która znajduje się pod tym kluczem zawiera informację opisującą wyjątek. Odpowiedź ta posiada kod odpowiedzi HTTP równy 500.

# Baza danych

## Analiza przechowywanych danych

W celu utworzenia bazy danych konieczne było przeanalizowanie wszystkich informacji, które będzie musiała przechowywać aplikacja.

Analizę rozpoczęto od analizy jednego z maili otrzymanych od osoby odpowiedzialnej za zamawianie posiłków



Rysunek 9.1 Przykładowy email o konieczności złożenia zamówienia

Z wiadomości tej wywnioskowano, że informację na temat zlecenia złożenia zamówienia jednostkowego, które będą przechowywane są następujące:

* nazwa (aktualnie tytuł maila),
* limit cenowy,
* dodatkowe informacje (np. o konieczności wyboru stopnia wysmażenia mięsa),
* link do menu,
* ostateczny termin złożenia zamówienia

Następnie przeanalizowano kilka złożonych zamówień i wyciągnięto następujące wnioski:

* każde z nich zawiera jedno lub więcej posiłków,
* czasami pojawiają się dodatkowe komentarze np. z prośbą o zmianę składników,
* czasami posiłki zamawiane są w ilości kilku sztuk

Dodatkowo ze względu na to, że użytkownik musi wskazać cenę posiłku zdecydowano, że również ta informacja będzie przechowywana w bazie danych. Umożliwi to w przyszłości \ utworzenia nowych funkcjonalności opartych o te dane - np. można utworzyć funkcję, która będzie sama podpowiadała zestawy obiadowe na podstawie poprzednio złożonych zamówień - potrzebna tutaj będzie znajomość cen posiłków w celu kontroli wydatków, aby nie przekroczyć limitu cenowego.

Ostatecznie postanowiono przechowywać w bazie danych następujące informację o zamówionych posiłkach:

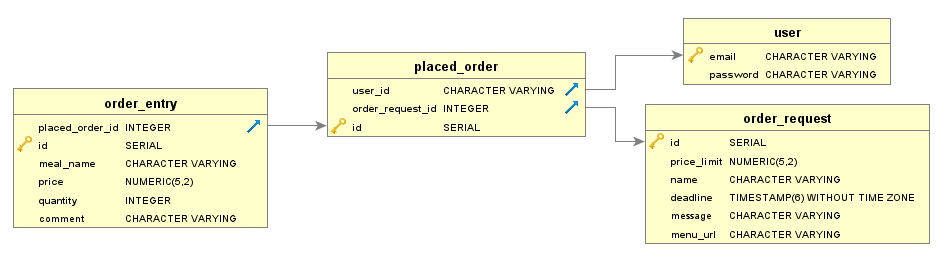
* nazwa posiłku,
* cena,
* ilość,
* dodatkowy komentarz

Ponadto ze względu na to, że dostęp do aplikacji możliwy jest tylko dla osób zalogowanych powstała konieczność przechowywania danych logowania które są następujące:

* login,
* hasło(w formie zahaszowanej ze względów bezpieczeństwa)

## Schemat tabel

Po zebraniu wszystkich informacji z poprzedniego podrozdziału i utworzeniu na ich podstawie tabel przechowywanych w bazie danych, utworzono schemat, który prezentuje się następująco:



Rysunek 9.2 Schemat bazy danych aplikacji

Zawarto w nim cztery tabele:

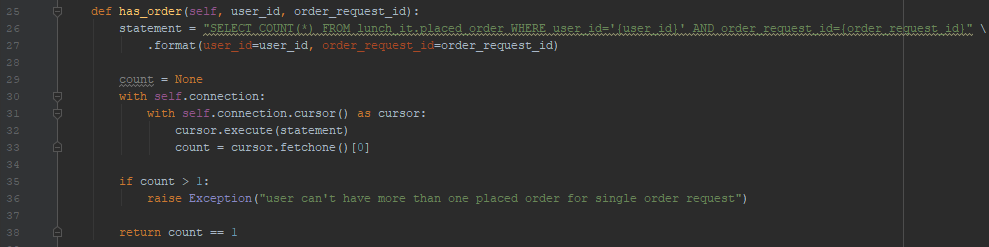
* *order\_entry* - znajdują się w niej posiłki należące do złożonego zamówienia
* *placed\_order* - wiersze reprezentują złożone zamówienia
* *order\_request* - posiada zlecenia złożenia zamówień jednostkowych
* *user* - gromadzi informacje o danych logowania użytkowników aplikacji

# Bezpieczeństwo

## SQL injection

SQL injection to rodzaj ataku na aplikację wykonującą zapytania do bazy danych, w której istnieje możliwość wstrzyknięcia złośliwego kodu. Spowodowane jest to nieodpowiednim sprawdzaniem danych dostarczanych przez użytkownika, które później są przekazywane bezpośrednio jako argumenty zapytania do bazy danych, dzięki czemu możliwe jest wykonanie dowolnego zapytania SQL.

W aplikacji początkowo nieświadomie zaimplementowano zapytania do bazy danych w taki sposób, że były one podatne na taki rodzaj ataku. Dowolna osoba mogła wykonać zapytanie do serwera podając zamiast jednego z argumentów, dowolne zapytanie SQL które jest poprzedzone i zakończone znakiem “;”, który wskazuje na zakończenie zapytania SQL.

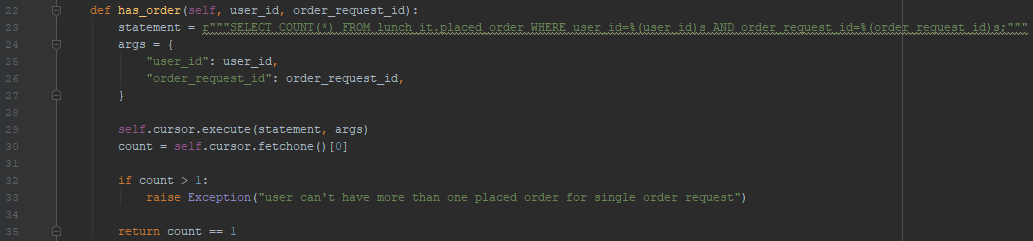
W pierwotnej implementacji użyto metody *format* operującej na łańcuchu znakowym, która wstawia argumenty zapytania bez ich sprawdzania pod kątem ataku SQL injection.   


Rysunek 10.1 Kod podatny na sql injection

Przesyłając do serwera zapytanie sprawdzające czy użytkownik złożył zamówienie, podając jako order\_request\_id zapytanie SQL które chcemy wstrzyknąć(np.“;drop database lunch\_it;”), zapytanie po dodaniu argumentów wyglądałoby następująco:  
 **SELECT COUNT(\*) FROM lunch\_it.placed\_order WHERE user\_id=sample\_user AND order\_request\_id=;drop database;**;

po wykonaniu metody *cursor.execute(statement)* wykonane byłyby 2 komendy. Pierwszą z nich oznaczono powyżej zielonym tłem, a drugą będąca złośliwym kodem wstrzykniętym przez atakującego - pomarańczowym tłem.

W aktualnej implementacji, po dostrzeżeniu podatności na atak SQL injection, zauważono, że biblioteka *psycopg2* oferuje odpowiednie zabezpieczenie realizowane przez sprawdzanie argumentów zapytania. Wymaga ona odpowiedniego oznaczenia miejsc na argumenty w łańcuchu znakowym będącym zapytaniem SQL oraz przekazaniu go wraz z drugim argumentem będącym mapą argumentów do zapytania SQL. Dzięki temu aplikacja jest odporna na ataki SQL injection.

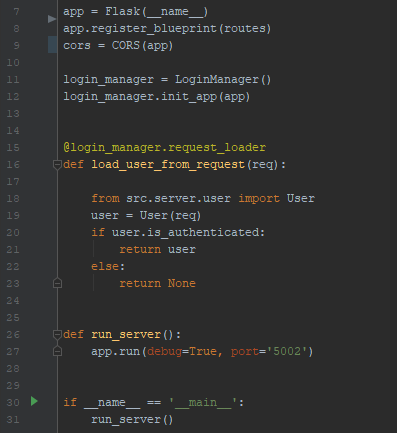
**

Rysunek 10.2 Implementacja bez luki pozwalającej na wstrzyknięcie złośliwego kodu SQL

## Szyfrowanie danych

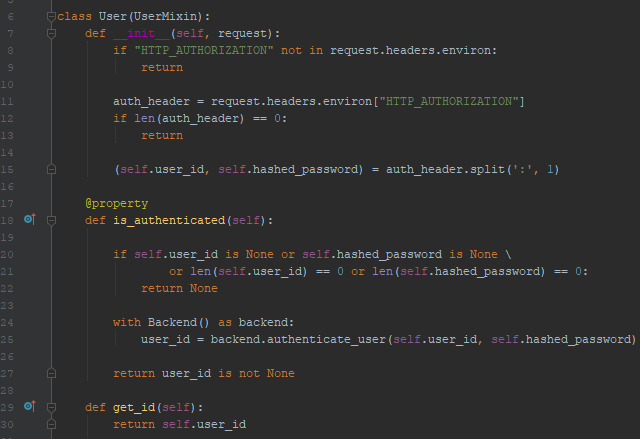
Gdy osoba niepożądana otrzyma dostęp do naszego konta w aplikacji, ze względu na charakter programu i możliwości które nam udostępnia, osoba ta nie jest w stanie wyrządzić nam znaczących szkód. Problemem jednak jest, gdy wykradzione dane logowania zostały użyte przez użytkownika również w innych serwisach (np. w aplikacji bankowej). Z tego względu hasła użytkowników które są przesyłane między klientem a serwerem lub przechowywane w bazie danych są zaszyfrowane. Dzięki temu gdy osoba nieporządana zdoła wykraść hasło, nie będzie miała możliwości dostać się do innej aplikacji/strony w której istnieje konto o tych samych danych dostępowych.

## Autoryzacja zasobów

**Autoryzacja zasobów po stronie serwera realizowana jest przy użyciu biblioteki *flask\_login*. Wymaga ona zadeklarowania obiektu *LoginManager,* który wiązany jest z obiektem o nazwie *app* będącym serwerem. Następnie dostarczamy funkcję(odpowiednio oznaczoną dekoratorem), która jako parametr przyjmuje zapytanie wysłane do serwera. Funkcja ta na podstawie otrzymanego parametru musi zwrócić *None* w przypadku gdy zapytanie nie zostało zautoryzowane lub obiek *User* w przypadku gdy autoryzacja powiodła się.

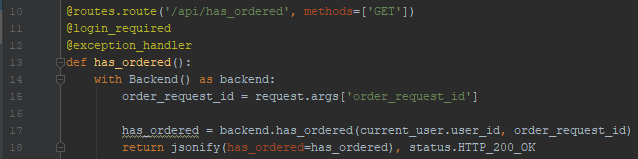
Rysunek 10.3 LoginManager zarządzający użytkownikami i procesem autoryzacji

Autoryzacja odbywa się na podstawie nagłówka *AUTHORIZATION* i wykonywana jest w konstruktorze klasy *User*.

**

Rysunek 10.4 Implementacja klasy User

Następnie konieczne jest dodanie dekoratora @*login\_required* do wszystkich zasobów, które mają wymagać autoryzacji



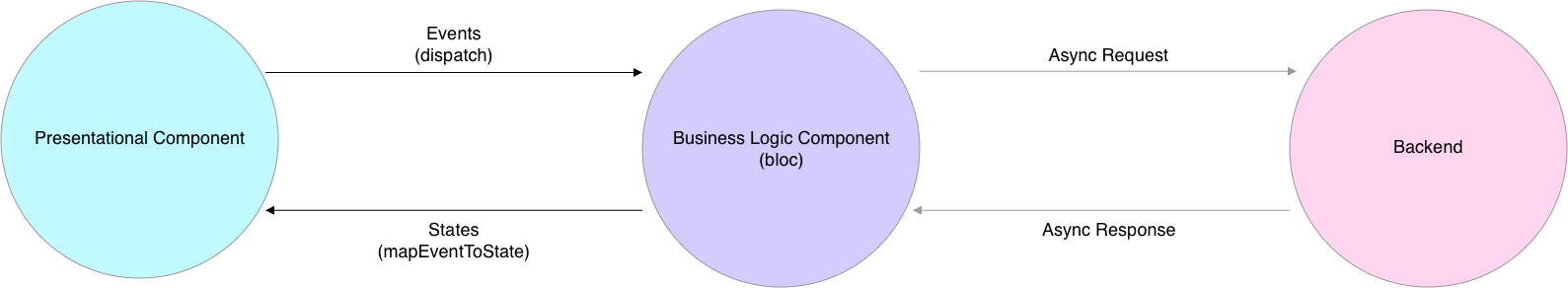
Rysunek 10.5 przykład użycia dekoratora @login\_required, który powoduje, że zasób wymaga autoryzacji

Dzięki przedstawionym krokom, w momencie wykonania zapytania do serwera pod adres zasobu, który wymaga autoryzacji, najpierw sprawdzane jest czy zapytanie powinno zostać dopuszczone do zasobu. W przypadku gdy zapytanie nie posiada danych autoryzacji lub są one niepoprawne, zwracana jest odpowiednia informacja i nie dochodzi do sfinalizowania zapytania. W przeciwnym przypadku zapytanie zostaje przetworzone w normalny sposób.

# Architektura aplikacji

## BLoC

Architekturą zastosowaną w aplikacji mobilnej jest wzorzec *BLoC* (ang. **B**usiness **Lo**gic **C**omponent)



Rysunek 11.1 Schemat wzorca BLoC źródło: https://pub.dev/packages/bloc

Wzorzec ten pozwala na odseparowanie warstwy prezentacji od logiki biznesowej. Użycie wzorca *BLoC* ułatwia testowanie aplikacji oraz ponowne wykorzystywanie komponentów.

Działanie tego wzorca oparte jest na strumieniach, które przesyłają zdarzenia i stany.

Dla przykładu posłużymy się widgetem, który wyświetla dane otrzymane z serwera po naciśnięciu przycisku. Przy użyciu wzorca BLoC realizowane jest to następująco.

Wciśnięcie przycisku dodaje do strumienia obiektu BLoC zdarzenie, które prezentuje wciśnięcie przycisku. Zdarzenie takie może być ukryte wewnątrz metody, którą udostępnia nam obiekt BLoC, dzięki czemu wywołanie go ma bardziej przyjazną formę. Obiekt BLoC nasłuchuje strumienia zdarzeń i wykonuje odpowiednie operacje w zależności od otrzymanego zdarzenia. Na przykład dla zdarzenia wciśnięcia przycisku wykona on zapytanie do serwera, które jest asynchroniczne i zostanie zwrócone w późniejszym czasie. W momencie otrzymania odpowiedzi dodawane jest ono do strumienia, który jest strumieniem stanów. Strumienia tego nasłuchuje warstwa prezentacji. Nasłuchiwanie to realizowane jest najczęściej poprzez widget *StreamBuilder,* który powoduje, że w momencie pojawienia się nowych danych na nasłuchiwanym strumieniu, całe drzewo widgetów osadzonych w *StreamBuilderze* zostaje przebudowane. Dzięki temu pozbywamy się konieczności “ręcznego” odświeżania widoku po stronie warstwy prezentacji.

Dzięki zastosowaniu powyższej architektury warstwa prezentacji wysyła jedynie zdarzenia i oczekuje stanów, nie wie ona nic o sposobie w jaki pozyskiwany i jak przetwarzany jest stan, który otrzymuje do wyświetlenia.

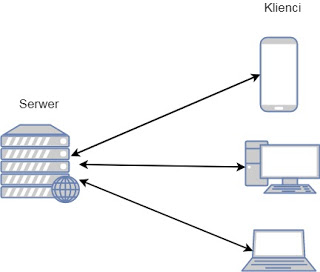
Podobnie jest z warstwą logiki biznesowej. Otrzymuje ona tylko zdarzenia i w zależności od otrzymanego zdarzenia wykonuje odpowiednie dla niego czynności czego skutkiem jest jakiś stan (np. informacje o wszystkich dostępnych użytkownikach). Stan ten jest odsyłany do warstwy prezentacji poprzez strumień stanów. Dzięki temu warstwa logiki biznesowej nie wpływa w żaden sposób na sposób prezentacji danych.

## Klient - serwer

Cała aplikacja jest oparta o architekturę typu klient-serwer. W systemie takim wyróżnia się 2 rodzaje urządzeń – serwer, który oferuje usługi oraz klienci korzystający z tych usług. Serwer przechowuje wszystkie zasoby i udostępnia je przez zdefiniowane interfejsy. Dzięki temu wszystkie dane będące na takim urządzeniu są dużo bezpieczniejsze. Komunikacja pomiędzy dwoma rodzajami urządzeń odbywa się przez określone protokoły komunikacyjne z czego najpopularniejszym jest TCP/IP.

W architekturze tej serwer posiada wszystkie informacje o strukturze sieci. Klienci nie wiedzą o istnieniu innych klientów i nie mają możliwości bezpośredniej komunikacji między sobą bez udziału serwera.

W naszej aplikacji wyróżniamy 2 rodzaje klientów - klient mobilny będący aplikacją zainstalowaną na smartfonie oraz klient webowy będący klientem użytkowanym poprzez przeglądarkę.



Rysunek 11.2 Schemat architektury klient-server źródło: http://architektura-oprogramowania.blogspot.com/p/architektura-klient-serwer.html

# Podsumowanie

## Perspektywa dalszego rozwoju aplikacji

### Propozycje zamawiania posiłków

Ze względu na dane, które są przechowywane w bazie danych istnieje możliwość dodania do aplikacji w łatwy sposób kolejnej funkcjonalności, którą byłoby proponowanie zestawów obiadowych przez aplikację. Dane te zawierają nazwy posiłków wraz z ich cenami. Dzięki temu możliwe jest stworzenie zestawów opartych na poprzednich zamówieniach użytkowników, które będą wpasowywać się w limit cenowy. Rozwiązanie takie mogłoby jeszcze bardziej usprawnić proces zamawiania posiłków, ze względu na minimalizację czasu poświęconego na wybór posiłku oraz likwidację procesu wyboru posiłków opartego o zaznaczanie pozycji z menu.

### Usprawnienie OCR

Aktualny algorytm rozpoznawania tekstu na obrazach nie jest w perfekcyjny. Z tego powodu dodano możliwość ręcznej korekty tego, co zostało rozpoznane jako tekst. Ze względu na to, że użytkownik dostarcza korektę rozpoznanego tekstu, możliwe jest zapisywanie tych danych, które będą powiązane z obrazkiem, z którego rozpoznawany jest tekst, a następnie stworzenie algorytmu sztucznej inteligencji, która uczyła by się na tych danych korekcyjnych. Zabieg taki umożliwiłby poprawienie skuteczności odczytów tekstu z zaznaczanych obszarów.

### Powiadomienia

Przydatną funkcjonalnością byłoby również otrzymywanie powiadomień związanych z koniecznością złożenia zamówienia. Dzięki temu zminimalizowano by możliwość zapomnienia o złożeniu zamówienia. Dodatkowo powiadomienia takie mogłyby służyć do informowania użytkowników o przewidywanym czasie dostawy.

## Wnioski

Zaproponowane rozwiązanie problemu w postaci aplikacji LunchIT spełnia postawione założenia i pozwala na usprawnienia procesu grupowego zamawiania posiłków. Celem który sobie postawiłem podczas pisania tej pracy było nauczenie się nowych technologii oraz napisanie aplikacji, która będzie użyteczna i ułatwi komuś życie. Projektowanie aplikacji, zaczynając od analizy problemu poprzez dobór technologii po implementację było bardzo ciekawym doświadczeniem, które pozwoliło mi na zapoznanie się z problemami, z którymi nigdy nie miałem styczności w moim życiu zawodowym. Postawiony cel uważam za zrealizowany.

# Wykaz rysunków

Rysunek 6.1 Diagram przepływu danych, poziom 0 12

Rysunek 6.2 Diagram przepływu danych, poziom 1 12

Rysunek 7.1 Prognoza udziałów rynkowych systemów na smartfony źródło: https://www.idc.com/promo/smartphone-market-share/os 13

Rysunek 8.1 Widok logowania 16

Rysunek 8.2 sprawdzanie poprawności danych logowania 17

Rysunek 8.3 Widok logowania po podaniu nieprawidłowych danych logowania 17

Rysunek 8.4 Widok rejestracji nowego użytkownika 17

Rysunek 8.5 Implementacja modułu odpowiedzialnego za sprawdzanie zapamiętanych danych 18

Rysunek 8.6 Przykład użycia metody pomocniczej służącej do komunikacji z serwerem 19

Rysunek 8.7 Miejsce użycia argumentu sendWithAuthHeader 19

Rysunek 8.8 Przygotowanie nagłówka autoryzacji 19

Rysunek 8.9 Główny widok prezentujący listę zleceń zamówienia. 21

Rysunek 8.10 Widok prezentujący złożone zamówienie 22

Rysunek 8.11 Widok zamawiania posiłku 23

Rysunek 8.12 Przykład prezentujący moment zaznaczania posiłku 24

Rysunek 8.13 metoda build klasy MarkingManager 25

Rysunek 8.14 Metoda initState klasy MarkingManager 26

Rysunek 8.15 Metoda getMarked klasy ContentMarker 27

Rysunek 8.16 Metoda pomocnicza używana w metodzie getMarked() 27

Rysunek 8.17 Widok dodawania posiłku 28

Rysunek 8.18 Dolna belka przedstawiająca informacje o stanie aktualnego zamówienia 28

Rysunek 8.19 Koszyk z zamówieniami 29

Rysunek 8.20 Widok koszyka z zamówieniami przy przekroczonym limicie cenowym 30

Rysunek 8.21 Komunikat potwierdzający powodzenie żłożenia zamówienia 31

Rysunek 8.22 Zrzut ekranu prezentujący zmianę statusu zamówienia o nazwie “Warsztat” 32

Rysunek 8.23 Widok podstrony z tworzeniem nowych zleceń 33

Rysunek 8.24 Widok podstrony wyświetlającej wszystkie zlecenia 34

Rysunek 8.25 Deklaracja tabeli wyświetlającej zlecenia zamówień 35

Rysunek 8.26 Część skryptowa strony display\_order\_requests.html, która pobiera dane z serwera i ładuje je do tabeli 36

Rysunek 8.27 Zamówienie jednostkowe - użytkownik nr 1 36

Rysunek 8.28 Zamówienie jednostkowe - użytkownik nr 2 36

Rysunek 8.29 Zamówienie jednostkowe - użytkownik nr 3 37

Rysunek 8.30 Zamówienie grupowe wygenerowane z wcześniej przedstawionych zamówień jednostkowych 37

Rysunek 8.31 Implementacja zasobu służącego do stworzenia nowego użytkownika prezentująca zwracane kody odpowiedzi HTTP 48

Rysunek 8.32 Implementacja dekoratora exception\_handler zwracającego kod błędu 500 48

Rysunek 9.1 Przykładowy email o konieczności złożenia zamówienia 49

Rysunek 9.2 Schemat bazy danych aplikacji 50

Rysunek 10.1 Kod podatny na sql injection 51

Rysunek 10.2 Implementacja bez luki pozwalającej na wstrzyknięcie złośliwego kodu SQL 52

Rysunek 10.3 LoginManager zarządzający użytkownikami i procesem autoryzacji 52

Rysunek 10.4 Implementacja klasy User 53

Rysunek 10.5 przykład użycia dekoratora @login\_required, który powoduje, że zasób wymaga autoryzacji 53

Rysunek 11.1 Schemat wzorca BLoC źródło: https://pub.dev/packages/bloc 54

Rysunek 11.2 Schemat architektury klient-server źródło: http://architektura-oprogramowania.blogspot.com/p/architektura-klient-serwer.html 55

# Bibliografia

1. Bereza-Jarociński B., Szomański B., *Inżynieria oprogramowania. Jak zapewnić jakość tworzonym aplikacjom,* Helion, Gliwice, Polska, 2009
2. Duckett J., *HTML and CSS : Design and Build Websites,* John Wiley & Sons Inc, New York, United States, 2011
3. Duckett J., *JavaScript and JQuery : Interactive Front-End Web Development*, John Wiley & Sons Inc, New York, United States, 2014
4. Dybikowski Z., *PostgreSQL. Wydanie II*, Helion, Gliwice, Polska, 2012
5. Mainkar P., *Giordano S., Google Flutter Mobile Development Quick Start Guide : Get up and running with iOS and Android mobile app development*, Packt Publishing Limited, Birmingham, United Kingdom, 2019
6. Martin R. *Clean Code: A Handbook of Agile Software Craftsmanship 1st Edition,* Pearson Education (US), Upper Saddle River, United States, 2009
7. Matthes E.*, Python Crash Course (2nd Edition) : A Hands-On, Project-Based Introduction to Programming*, No Starch Press, San Francisco, California, 2019
8. Hjelle G., *Primer on Python Decorators* (online) https://realpython.com/primer-on-python-decorators/ (dostęp: 2019-11-16)
9. *Smartphone Market Share* (online), https://www.idc.com/promo/smartphone-market-share/os. (dostęp: 2019-11-15)
10. *What can we learn from Google’s offices about workplace design?* (online)
11. https://www.workspacedesign.co.uk/what-can-we-learn-from-googles-offices-about-workplace-design/ (dostęp: 2019-11-13)
12. *Flutter Documentation* (online) https://flutter.dev/docs (dostęp: 2019-11-10)
13. *PostgreSQL Tutorial* (online), http://www.postgresqltutorial.com/ (dostęp: 2019-11-13)