

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 Zając, prof. PK**

Kraków 2020

Spis treści

[1. Słownik 4](#_Toc29927038)

[2. Wstęp 5](#_Toc29927039)

[3. Cel, zakres pracy i efekt końcowy 6](#_Toc29927040)

[3.1. Cel 6](#_Toc29927041)

[3.2. Zakres 6](#_Toc29927042)

[3.3. Efekt końcowy 6](#_Toc29927043)

[4. Podejście do problemu 7](#_Toc29927044)

[4.1. Analiza aktualnego podejścia 7](#_Toc29927045)

[4.1.1. Opis realizacji procesu 7](#_Toc29927046)

[4.1.2. Problematyczność aktualnego procesu 7](#_Toc29927047)

[4.2. Koncepcja rozwiązania 8](#_Toc29927048)

[5. Wymagania aplikacji 10](#_Toc29927049)

[5.1. Wymagania funkcjonalne 10](#_Toc29927050)

[5.2. Wymagania niefunkcjonalne 11](#_Toc29927051)

[6. Diagram przepływu danych 13](#_Toc29927052)

[7. Wybór technologii 14](#_Toc29927053)

[7.1. Klient mobilny 14](#_Toc29927054)

[7.1.1. Analiza 14](#_Toc29927055)

[7.1.2. Opis wybranej technologii 15](#_Toc29927056)

[7.2. Serwer i silnik logiczny 15](#_Toc29927057)

[7.3. Baza danych 16](#_Toc29927058)

[7.4. Klient webowy 16](#_Toc29927059)

[8. Implementacja 17](#_Toc29927060)

[8.1. Aplikacja mobilna 17](#_Toc29927061)

[8.1.1. Dostęp 17](#_Toc29927062)

[8.1.2. Przeglądanie zleceń złożenia zamówienia jednostkowego 21](#_Toc29927063)

[8.1.3. Wyświetlanie złożonego zamówienia 23](#_Toc29927064)

[8.1.4. Składanie zamówień 24](#_Toc29927065)

[8.2. Aplikacja webowa 34](#_Toc29927066)

[8.2.1. Tworzenie nowego zlecenia 34](#_Toc29927067)

[8.2.2. Wyświetlanie listy zleceń 35](#_Toc29927068)

[8.2.3. Wyświetlanie zamówienia grupowego 37](#_Toc29927069)

[8.3. Serwer – dokumentacja zasobów 39](#_Toc29927070)

[8.3.1. Autentykacja 40](#_Toc29927071)

[8.3.2. Tworzenie nowego użytkownika 41](#_Toc29927072)

[8.3.3. Grupowe zamówienie 42](#_Toc29927073)

[8.3.4. Sprawdzanie czy użytkownik złożył zamówienie 43](#_Toc29927074)

[8.3.5. Odczytywanie tekstu z obrazka 44](#_Toc29927075)

[8.3.6. Składanie zamówienia jednostkowego 45](#_Toc29927076)

[8.3.7. Sprawdzanie zamówienia jednostkowego 46](#_Toc29927077)

[8.3.8. Utworzenie nowego zlecenia 47](#_Toc29927078)

[8.3.9. Wszystkie zlecenia 48](#_Toc29927079)

[8.3.10. Wszystkie zlecenia złożenia zamówień jednostkowych wraz z identyfikatorem złożonego zamówienia 49](#_Toc29927080)

[8.4. Kody zapytań oraz obsługa wyjątków 50](#_Toc29927081)

[9. Baza danych 51](#_Toc29927082)

[9.1. Analiza przechowywanych danych 51](#_Toc29927083)

[9.2. Schemat tabel 52](#_Toc29927084)

[10. Bezpieczeństwo 53](#_Toc29927085)

[10.1. SQL injection 53](#_Toc29927086)

[10.2. Szyfrowanie danych 54](#_Toc29927087)

[10.3. Autoryzacja zasobów 55](#_Toc29927088)

[11. Architektura aplikacji 57](#_Toc29927089)

[11.1. BLoC 57](#_Toc29927090)

[11.2. Klient-serwer 58](#_Toc29927091)

[12. Podsumowanie 59](#_Toc29927092)

[12.1. Perspektywa dalszego rozwoju aplikacji 59](#_Toc29927093)

[12.1.1. Propozycje zamawiania posiłków 59](#_Toc29927094)

[12.1.2. Usprawnienie OCR 59](#_Toc29927095)

[12.1.3. Powiadomienia 59](#_Toc29927096)

[12.2. Wnioski 59](#_Toc29927097)

[13. Wykaz rysunków 60](#_Toc29927098)

[14. Bibliografia 62](#_Toc29927099)

# 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, zlecenie złożenia zamówienia** – informacja o konieczności złożenia zamówień jednostkowych.

**zamawiający** – osoba składająca zamówienie jednostkowe.

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

# Wstęp

Czasy współczesne cechuje duży pośpiech, wobec czego poszukuje się rozwiązań pozwalających efektywnie zarządzać czasem. Jednym ze środków optymalizujących gospodarowanie tym zasobem jest zamawianie jedzenia na dowóz. Z takiej możliwości korzystają między innymi firmy oferujące swoim pracownikom posiłki zamawiane z restauracji.

W ostatnich latach w Polsce dużą popularność zyskały serwisy, które tworzą wielkie sieci restauracji oraz dostarczają dogodny interfejs dla końcowego użytkownika, umożliwiający zamówienie jedzenia oraz jego opłacenie. Ze względu na to, że uczestnictwo w takiej sieci nie jest darmowe dla właścicieli lokali gastronomicznych, nie wszyscy decydują się na dołączenie do programu, ze względów finansowych. Ma to wpływ na klientów, którzy pozbawieni są możliwości złożenia zamówienia przez Internet. Jedyną, a zarazem mało dogodną opcją, z jakiej mogą skorzystać, jest złożenie zamówienia przez telefon i dokonanie płatności przy odbiorze.

Niektóre z firm postanowiły w ramach benefitów fundować pracownikom jedzenie, zwiększając tym samym atrakcyjność swoich ofert pracy. Zaletą takiego rozwiązania jest to, że po pracy zatrudniony nie musi zajmować się przygotowywaniem posiłków. Dodatkowym atutem tego rozwiązania, niosącym korzyść nie tylko dla pracownika, lecz również dla pracodawcy, jest zlikwidowanie czynnika powodującego dekoncentrację – uczucia głodu. Przekłada się to pozytywnie na wydajność. Co więcej, dzięki wspólnemu spożywaniu posiłków, zatrudnione osoby wchodzą częściej w interakcję. Wpływa to pozytywnie na relacje międzyludzkie i może poprawiać komunikację między pracownikami.

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. Jedyną opcją jest kontakt telefoniczny i podyktowanie listy wybranych dań.

Firma cały czas się rozrasta, a proces zamawiania posiłków staje się coraz bardziej problematyczny ze względu na to, że zajmuje on coraz więcej czasu, w szczególności dotyczy to osoby odpowiedzialnej za zamawianie obiadów. Ponadto przy dużej ilości zamówień szansa na pomyłkę podczas sumowania zamówień jednostkowych znacznie wzrasta. Przypadkowo pominięte zamówienia są zamawiane w kolejnej turze. Często wiążę się to z dodatkowymi kosztami za dowóz, ponieważ jeden posiłek nie przekracza minimalnej kwoty zamówienia wymaganej w przypadku darmowej dostawy.

# Cel, zakres pracy i efekt końcowy

## Cel

Celem pracy jest optymalizacja czasochłonnego procesu grupowego zamawiania posiłków dla pracowników firm IT.

Dedykowane rozwiązanie oparto na doświadczeniach osób uczestniczących w procesie zamawiania jedzenia w jednej z krakowskich firm IT. Dodatkowym celem jest osiągnięcie rozwiązania o niskich kosztach utrzymania i wdrożenia.

## Zakres

Zakres pracy obejmuje stworzenie systemu składania zamówień, w którym zarówno osoby składające zamówienia jednostkowe (pracownicy firmy), jak i osoba administrująca całym procesem (łącznik między osobami składającymi zamówienia jednostkowe a restauracją) zaoszczędzą czas potrzebny do wykonania czynności związanych z zamawianiem.

Zostanie również utworzony prototyp aplikacji dostępnej dla wszystkich pracowników bez względu na system zainstalowany na posiadanym urządzeniu mobilnym. Całe rozwiązanie oparte będzie wyłącznie na darmowych narzędziach.

Całość jako system oraz aplikacja udostępniona dla użytkownika powinny usprawnić proces składania zamówień jednostkowych (przez pracowników firmy), jak i grupowych (przez administratora).

## Efekt końcowy

W efekcie końcowym powstanie aplikacja, która zaoszczędzi czas wszystkim pracownikom, a głównie osobie odpowiedzialnej za składanie zamówień grupowych, wpływając na usprawnienie działania przedsiębiorstwa.

# Podejście do problemu

## Analiza aktualnego podejścia

### Opis realizacji procesu

Aktualny proces zamawiania posiłków w firmie, dla której tworzona jest aplikacja, przebiega następująco. W godzinach przedpołudniowych osoba odpowiedzialna za zamawianie obiadów wysyła do reszty pracowników informację, z jakiej restauracji zamawiane jest jedzenie, do której godziny należy odesłać listę posiłków, jaki jest limit cenowy oraz menu, z którego należy skorzystać. Po otrzymaniu wszystkich zamówień jednostkowych, osoba składająca zamówienie w restauracji musi przeanalizować wszystkie maile i uporządkować je w formie dogodnej do podyktowania przez telefon tj. zsumować powtarzające się zamówienia. Po złożeniu zamówienia grupowego następuje oczekiwanie na dostawę, po której posiłki przydzielane są do odpowiednich osób. Cały proces zostaje zakończony, gdy każdy z pracowników otrzyma swoje zamówienie.

### Problematyczność aktualnego procesu

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

Podczas zamawiania pojawiają się następujące problemy:

* niewygodne sprawdzanie, czy przygotowano już zlecenie zamówienia – konieczność przeszukania skrzynki e-mailowej by odnaleźć odpowiednią wiadomość dotyczącą zlecenia,
* ręczna kontrola sumarycznej wartości wybranych posiłków względem limitu cenowego,
* uciążliwość przepisywania nazw posiłków podczas zamawiania z menu w formie obrazu,
* konieczność „przeskakiwania” pomiędzy menu a wiadomością e-mail,
* ręczne zbieranie zamówień jednostkowych poprzez przeszukiwanie wiadomości

e-mail i scalanie ich w zamówienie grupowe,

* możliwość pomyłki przy scalaniu zamówienia grupowego – zwiększenie kosztów,
* zapominanie przez pracowników, co zamówili – ponownie wymaga to przeszukiwania skrzynki e-mail.

## 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 konieczności dostępu do komputera.

System wspierany będzie aplikacją webową, która posłuży do tworzenia zleceń zamówienia. W tym przypadku zdecydowano się na taką formę, ponieważ aplikacja mobilna byłaby niewygodna w momencie, gdy konieczne jest wypełnianie formularza zawierającego dużo danych, w tym często takich, które są kopiowane z innych okien (np. link do menu). Mobilność w tym przypadku nie jest konieczna, ponieważ zlecenia są tworzone przez pracownika odpowiedzialnego za zamawianie posiłków w restauracjach 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 takich zleceń. Pozwala to osobie odpowiedzialnej za zamawianie posiłków w restauracji na utworzenie zleceń na cały nadchodzący tydzień. Spowoduje to, że w kolejnych dniach jej obowiązki w zakresie zamawiania jedzenia ograniczone zostaną do wykonania połączenia telefonicznego w celu przekazania wygenerowanego zamówienia grupowego.

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

W przypadku, gdy zostało ono złożone, po kliknięciu na wcześniej wspomniany obiekt, następuje przeniesienie do widoku, w którym istnieje możliwość sprawdzenia listy wybranych posiłków.

Jeśli zamówienie jednostkowe nie zostało złożone, użytkownik przenoszony jest do widoku wyświetlającego menu.

Widok ten pozwala na wybór posiłków bez konieczności przepisywania ich nazw. Odbywa się to poprzez zaznaczanie fragmentów menu, które następnie przetwarzane są na tekst. Widoczna jest również informacja o tym, ile aktualnie wynosi sumaryczny koszt zamówienia.

Ze względu na możliwość przekroczenia limitu o niewielką kwotę (brak określonej wartości) oraz na to, że pracownicy czasami umawiają się na wspólne zamówienie zestawu, którym się podzielą (zamówienie składa jedna z osób), przekroczenie limitu nie powoduje braku możliwości złożenia zamówienia, lecz wyświetla łatwo dostrzegalną informację o zbyt wysokiej cenie wybranych posiłków. Pracownicy darzeni są zaufaniem, więc nie ma konieczności kontrolowania ich wydatków. Przyjmuje się, że będą oni przestrzegać limitu cenowego, więc informacja o sumarycznym koszcie zamówienia ma jedynie charakter informacyjny.

Do każdego zamawianego posiłku pracownik ma możliwość dodania komentarza, w którym może poprosić o zmianę składników lub o inny sposób przygotowania posiłku.

Dodatkowo istnieje możliwość podglądu wszystkich pozycji w aktualnym zamówieniu poprzez przejście do koszyka, gdzie widoczna jest informacja o poszczególnych daniach. Z poziomu koszyka możliwe jest zwiększenie/zmniejszenie liczby sztuk lub całkowitego usunięcia pozycji z zamówienia. Skompletowane zamówienie finalizowane jest za pomocą przycisku, po którego użyciu interakcja pracownika z aplikacją chwilowo się kończy i następuje oczekiwanie na dostawę jedzenia.

Kolejną aktywność podejmuje osoba składająca zamówienie w restauracji. Gdy minie wcześniej ustalony ostateczny termin złożenia zamówienia jednostkowego, odnajdowane jest odpowiednie zlecenie na stronie wyświetlającej je w sposób posortowany. Używana jest opcja wygenerowania zamówienia grupowego, które jest gotowe do podyktowania przez telefon podczas połączenia telefonicznego z restauracją.

Zamówienie grupowe składa się z wielu zamówień jednostkowych, które są sumowane w przypadku, gdy wystąpią powtórzenia. Dzięki temu mając dwa zamówienia jednostkowe zawierające ten sam posiłek, na zamówieniu grupowym pozycja taka pojawi się tylko raz, a liczba wskazująca jej ilość zostanie odpowiednio zwiększona.

Z poziomu aplikacji mobilnej dostępna jest informacja o złożonym wcześniej zamówieniu. Zapobiega to konieczności powrotu do stanowiska pracy w celu sprawdzenia listy wybranych posiłków w przypadku, gdy zapomniano, co zostało zamówione.

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

# Wymagania aplikacji

Aby ułatwić proces planowania implementacji aplikacji oraz określenia tego, co powinna realizować, przeanalizowano problem grupowego zamawiania posiłków z restauracji, a następnie wyszczególniono w kolejnych podrozdziałach wymagania funkcjonalne i niefunkcjonalne, które program ma spełniać.

## Wymagania funkcjonalne

Wymagania funkcjonalne opisują, jakie usługi dostarcza program, co powinien on realizować oraz jak zachowuje się on w określonych sytuacjach [1]. Są one następujące:

**Aplikacja mobilna:**

* wybieranie posiłków bez potrzeby przepisywania ich nazw,
* automatyczne sumowanie cen wszystkich wybranych posiłków w celu kontrolowania kosztów zamówienia (limit cenowy),
* możliwość podglądu złożonego zamówienia,
* dodawanie komentarzy do zamawianych posiłków (np. w celu zmiany składników),
* dostęp tylko dla zalogowanych użytkowników,
* możliwość utworzenia konta,
* opcja 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 wymagających tego zasobów,
* możliwość pobrania wszystkich utworzonych zleceń zamówień,
* opcja pobrania konkretnego zlecenia zamówień na podstawie jego ID,
* opcja utworzenia nowego konta użytkownika na podstawie podanego adresu e-mail oraz hasła,
* przetwarzanie obrazu na tekst (OCR),
* sprawdzanie czy użytkownik złożył zamówienie jednostkowe dla danego zlecenia zamówienia,
* możliwość złożenia zamówienie jednostkowego,
* opcja 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,
* opcja utworzenia nowego zlecenia zamówień,
* ustalanie limitu cenowego dla zlecenia zamówienia,
* ustalanie limitu czasowego na złożenie zamówienia jednostkowego,
* możliwość dołączenia dodatkowych informacji do zlecenia zamówienia, które będą widoczne dla osoby zamawiającej (np. o możliwości dodatkowego wyboru, który nie jest wspomniany w menu),
* opcja wyświetlenia listy zamówień sformatowanej w sposób wygodny do podyktowania przez telefon przy składaniu zamówienia

## Wymagania niefunkcjonalne

Wymagania niefunkcjonalne opisują, jak program powinien realizować swoje zadania. Definiują one cechy oprogramowania oraz ograniczenia nałożone na aplikację [1]. Dla niniejszego programu 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.

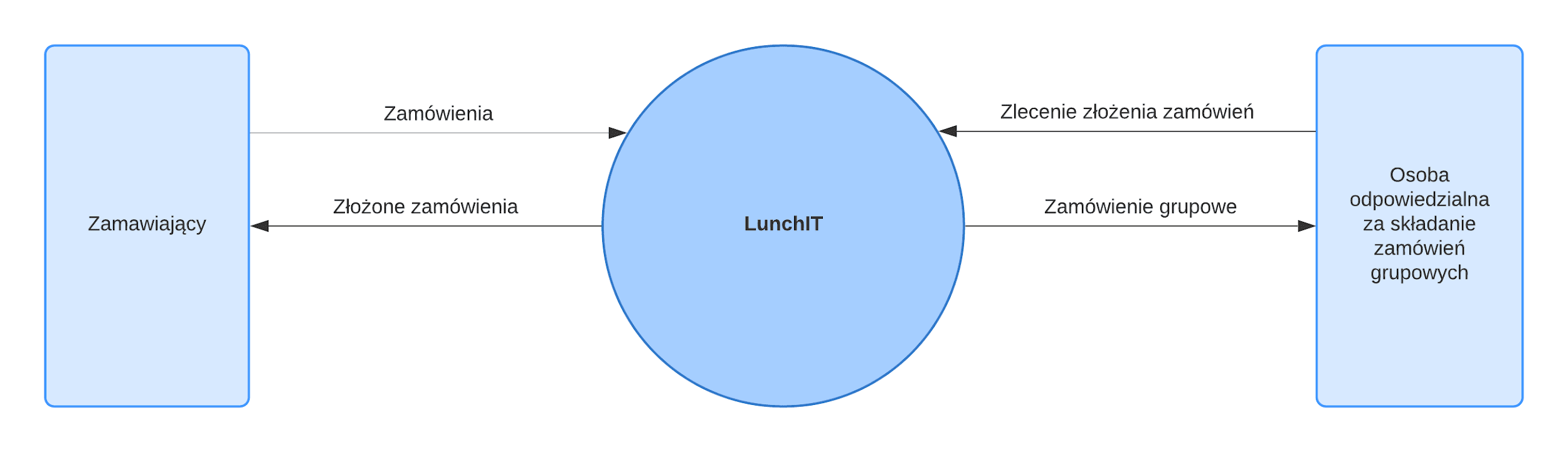
**Aplikacja mobilna:**

* wydajność – płynność działania i brak przerw w responsywności programu,
* kompatybilność – możliwość uruchomienia aplikacji na systemie Android Jelly Bean, v16, 4.1.x lub nowszym oraz na systemie iOS 8 lub nowszym,
* poprawność danych
  + weryfikacja nieprawidłowych danych wprowadzanych przez użytkownika,
  + sprawdzanie, czy wymagane pola zostały wypełnione.
* użyteczność – łatwy i intuicyjny interfejs,
* rozmiar – zajmuje mniej niż 100 MB pamięci.

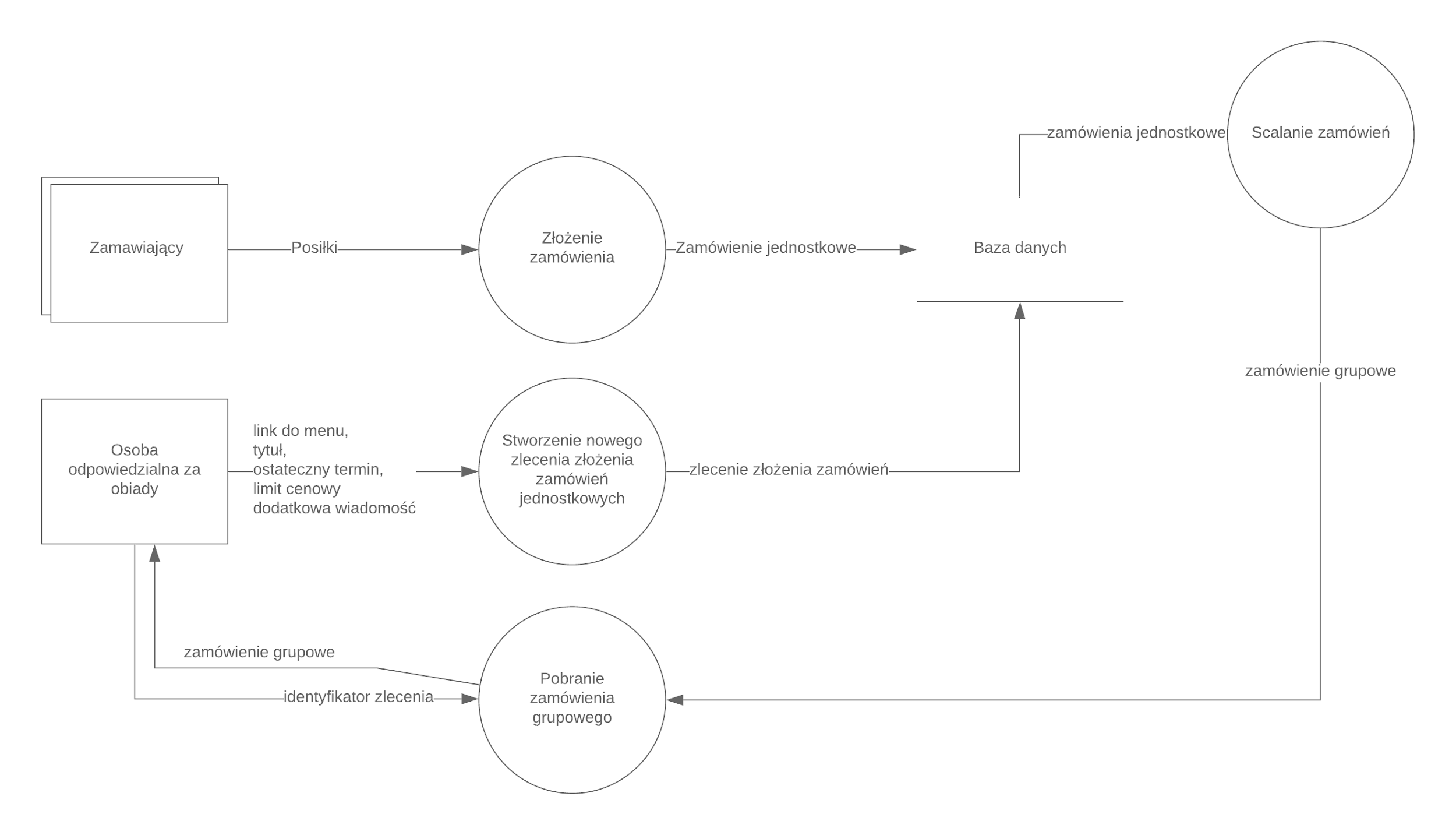
**System jako całość:**

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

# Diagram przepływu danych



Rysunek 6.1 Diagram przepływu danych, poziom 0 (źródło: praca własna)



Rysunek 6.2 Diagram przepływu danych, poziom 1 (źródło: praca własna)

# Wybór technologii

Podstawowymi kryteriami podczas doboru technologii użytych w aplikacji był brak kosztów związanych z użyciem danej technologii oraz szybkość wytwarzania oprogramowania [1].

## Klient mobilny

### Analiza

Badając strukturę rynku smartfonów 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 [9].



Rysunek 7.1 Prognoza udziałów rynkowych systemów na smartfony (źródło: https://www.idc.com/promo/smartphone-market-share/os - dostęp 2019-11-13)

Można zauważyć, że tendencja ta utrzymuje się na podobnym poziomie od 2017 roku. Ze względu na brak innych znaczących udziałowców rynkowych, większość mobilnych aplikacji powstających w bieżących latach obsługiwana jest tylko przez te dwie platformy.

Tworzenie aplikacji na oba środowiska wymaga napisania oddzielnych kodów źródłowych, znacząco zwiększając koszty wykonania oraz utrzymania programu. Dla obniżenia nakładów pieniężnych potrzebnych do stworzenia klienta 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. Stworzony został przez Google w 2017 roku. Framework ten jest użytkowany przy pomocy języka Dart [5].

Do zalet Fluttera należy hot-reload, który pozwala na wyświetlanie zmian zastosowanych w kodzie bez potrzeby przebudowywania programu, skutkując szybszym naprawianiem błędów w aplikacji oraz wygodniejszym projektowaniem jej interfejsu.

Wybrany framework udostępnia wiele predefiniowanych widżetów, których można używać do budowy aplikacji. W przypadku, gdy dostępna pula nie oferuje wystarczająco dużo, istnieje możliwość utworzenia własnych widżetów wedle potrzeb, a następnie wielokrotnego stosowania ich w kodzie, unikając powtórzeń.

Ponadto Flutter udostępnia API, umożliwiające komunikację z kodem napisanym w natywnych językach. Daje to możliwość optymalizacji pewnych fragmentów programu w zależności od systemu operacyjnego urządzenia, na którym uruchamiana jest aplikacja.

Flutter jest nową technologią, więc niektóre elementy mogą nie działać poprawnie. Za pomocą wcześniej wspomnianego API, który dostarcza framework, możliwe jest zaimplementowanie części kodu w natywnym języku. Zapewnia to, że rozwój aplikacji nie zostanie zablokowany przez błędy występujące w technologii.

## Serwer i silnik logiczny

Jako że jedną z kluczowych funkcji zaproponowanego rozwiązania jest przetwarzanie obrazów na tekst, a wymogiem jest szybkość pisania aplikacji, do implementacji serwera wybrano wieloplatformowy język skryptowy Python. Jego głównym założeniem jest możliwość pisania aplikacji na wysokim poziomie abstrakcji. Do zalet Pythona należy wysoce rozbudowany pakiet bibliotek standardowych, czyli takich, które dostarczane są razem z interpreterem języka. Dostępna jest również duża ilość bibliotek stworzonych przez społeczność, możliwych do pobrania i szybkiego skonfigurowania do działania za pomocą menadżera pakietów o nazwie pip [7].

Najważniejsze z użytych bibliotek to:

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

## Baza danych

Podczas wyboru bazy głównymi czynnikami decyzyjnymi były licencja pozwalająca na darmowe używanie systemu zarządzającego danymi w projektach komercyjnych oraz jego popularność. Umożliwia to łatwiejsze znalezienie materiałów edukacyjnych oraz pomocy społeczności.

Wybrany został PostgreSQL, który jest systemem zarządzania relacyjnymi bazami danych. Jest on darmowy i otwarto źródłowy oraz implementuje większość standardu SQL:2011. PostgresSQL można uruchomić na wielu systemach operacyjnych takich jak FreeBSD, OS X, Linux, Windows czy UNIX [4].

## Klient webowy

Do stworzenia klienta webowego wykorzystano jedno z podstawowych, a zarazem najpopularniejszych narzędzi:

**CSS** – język służący do opisywania wyglądu stron internetowych. Definiuje listę reguł, według których przeglądarka powinna wyświetlać zawartość wcześniej zadeklarowanego w języku HTML elementu. CSS umożliwia określanie lokalizacji elementów, koloru, marginesów, a nawet animacji [2].

**HTML** – jest to język znaczników wykorzystywany do tworzenia dokumentów HTML. Pozwala na określenie struktury strony internetowej [2].

**JavaScript** – język programowania wykorzystywany przy tworzeniu stron internetowych. Umożliwia między innymi dodanie logiki do strony WWW oraz dynamiczną zmianę treści lub jej wyglądu w zależności od sytuacji [3].

**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 odbieranie odpowiedzi [3].

**Bootstrap** – biblioteka języka CSS udostępniająca narzędzia do projektowania interfejsu graficznego stron internetowych. Niewielkim nakładem pracy, uzyskuje się stronę o atrakcyjnym wyglądzie.

# Implementacja

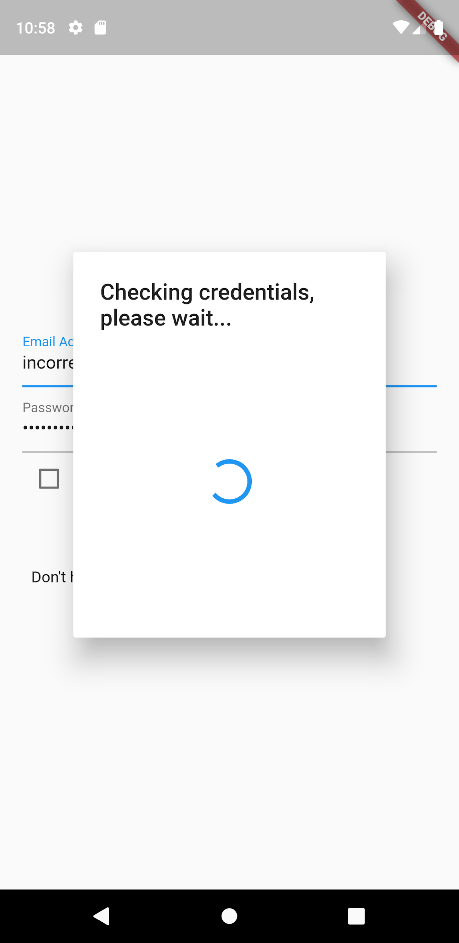
## Aplikacja mobilna

### Dostęp

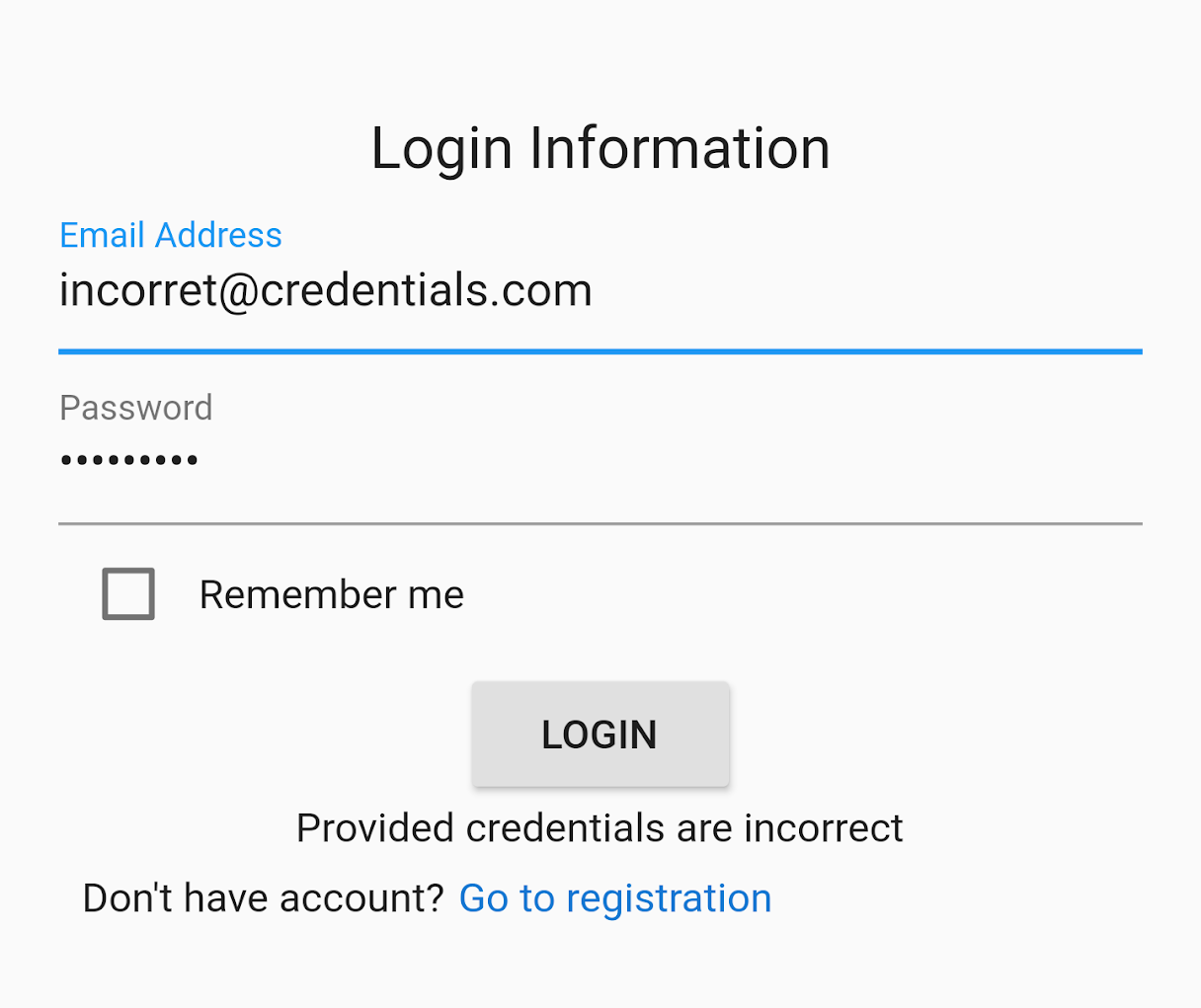
Rysunek 8.1 Widok logowania (źródło: praca własna)

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

Użytkownik ma możliwość zapamiętania hasła, dzięki czemu przy kolejnych uruchomieniach aplikacji, nie ma konieczności ponownego logowania się.

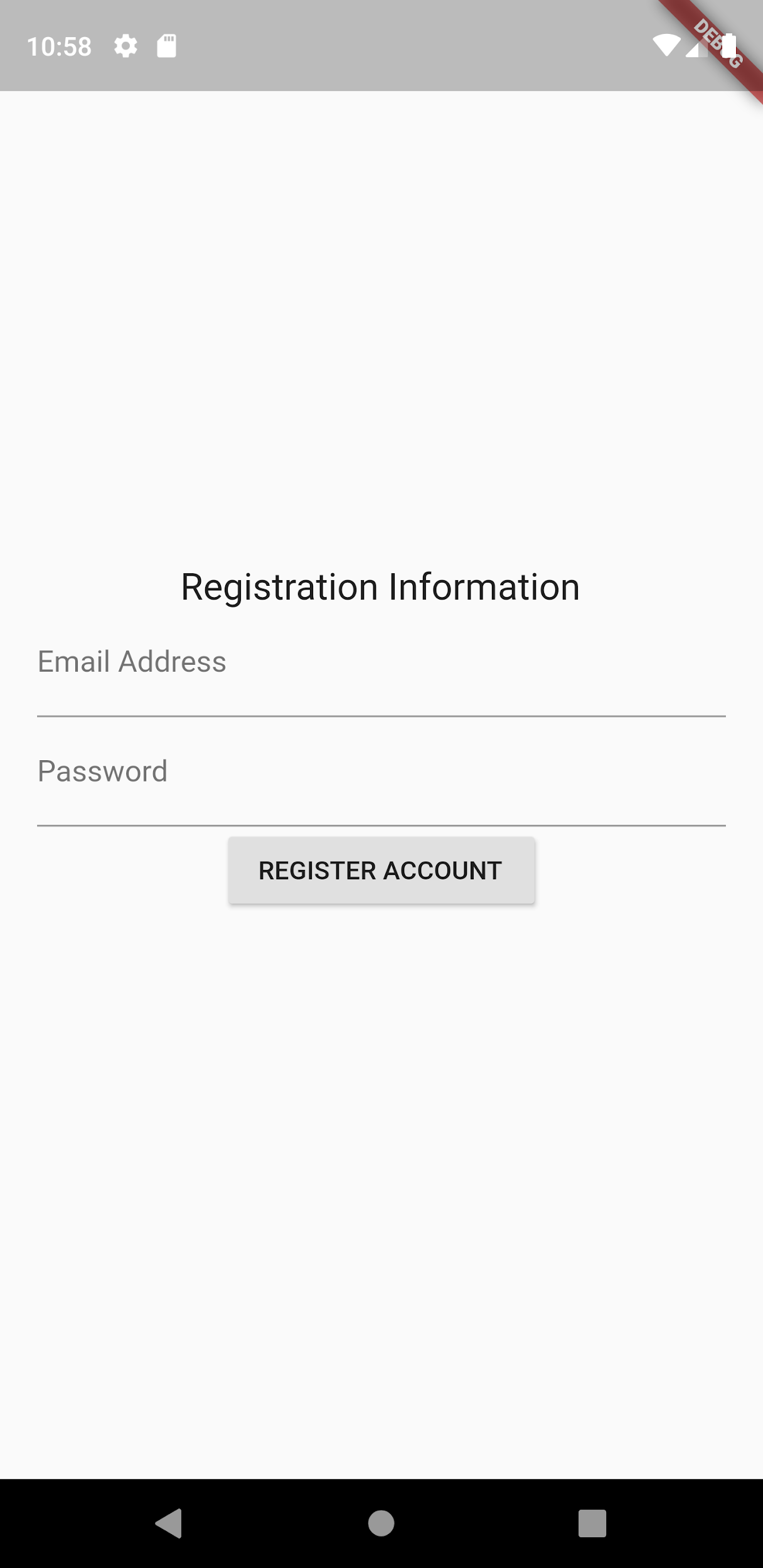
Po kliknięciu na *Zaloguj* użytkownik informowany jest o trwającym procesie sprawdzania poprawności danych logowania, realizowanym poprzez zapytanie do serwera, który szuka użytkownika w bazie danych.

Rysunek 8.2 Sprawdzanie poprawności danych logowania (źródło: praca własna)



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

Rysunek 8.3 Widok logowania po podaniu nieprawidłowych danych logowania (źródło: praca własna)



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

Rysunek 8.4 Widok rejestracji nowego użytkownika (źródło: praca własna)

Dostęp dla tylko zalogowanych użytkowników zaimplementowano w oparciu *Navigator* będący wbudowanym modułem. Odpowiedzialny jest za nawigowanie pomiędzy wyświetlanymi widokami, czyli za prezentacje odpowiednich treści na ekranie i wykonywanie odpowiedniej logiki z nią powiązanej.

Pierwszym widżetem, budowanym po uruchomieniu aplikacji jest *SavedCredentialsChecker*, którego zadaniem jest sprawdzenie, czy użytkownik zapamiętał swoje hasło podczas poprzedniej sesji. Następuje to poprzez sprawdzenie istnienia pliku z zahaszowanymi danymi logowania. Gdy ów plik istnieje, odczytywana jest jego zawartość i wykonywane zostaje zapytanie do serwera, który zwraca odpowiedź z informacją, czy dane logowania są prawidłowe. W przypadku gdy odnaleziono użytkownika w bazie danych i podane hasło jest poprawne, 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 (źródło: praca własna)

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

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

Do komunikacji z serwerem przygotowano specjalną metodę pomocniczą, która powoduje, że wysyłanie zapytań do serwera jest dużo wygodniejsze [6]. Poniżej zaprezentowano przykład użycia tej metody w celu komunikacji z zasobem, który wymaga autoryzacji.



Rysunek 8.6 Przykład użycia metody pomocniczej służącej do komunikacji z serwerem (źródło: praca własna)

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

**

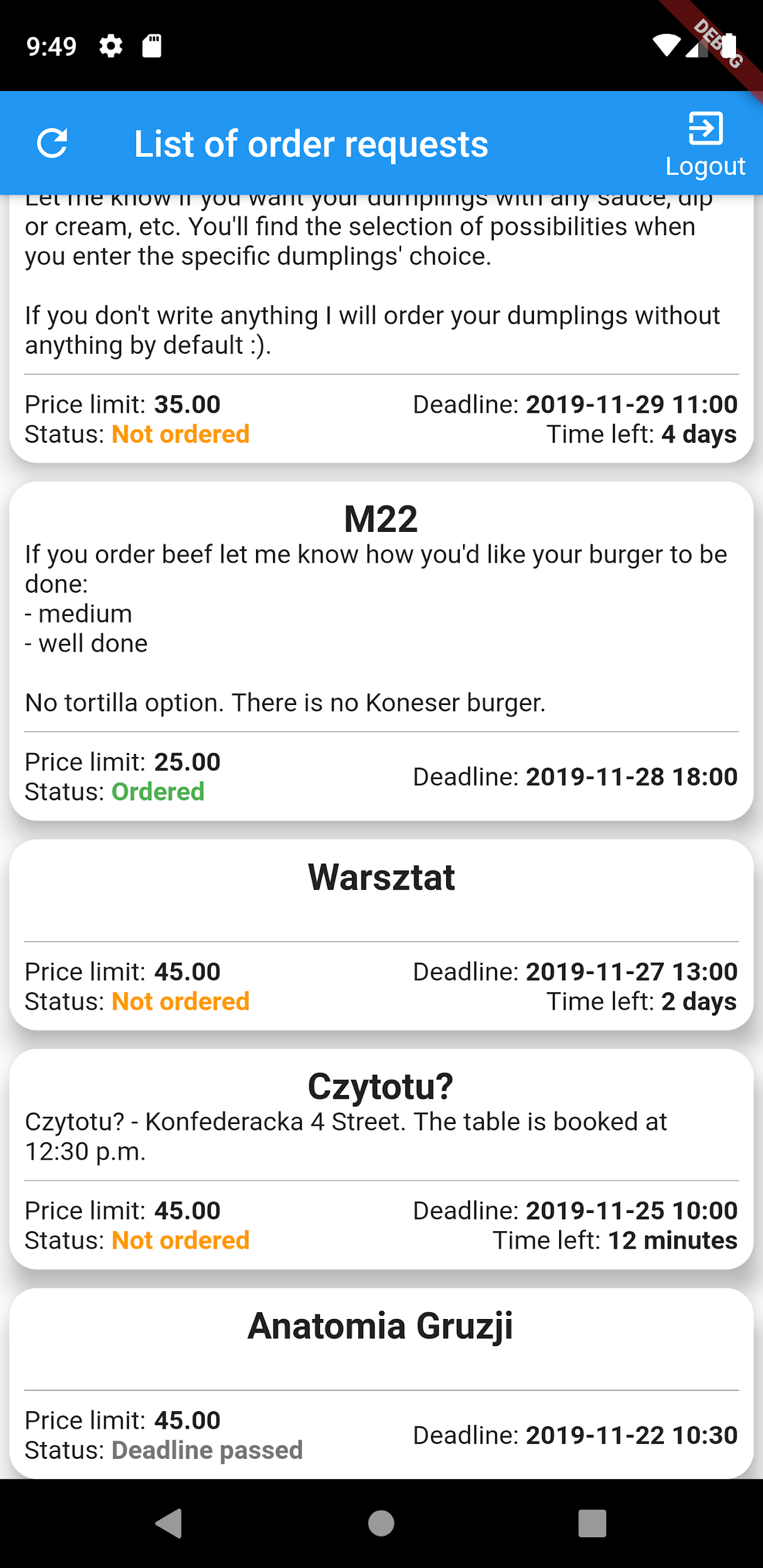
Rysunek 8.7 Miejsce użycia argumentu sendWithAuthHeader (źródło: praca własna)

**

Rysunek 8.8 Przygotowanie nagłówka autoryzacji (źródło: praca własna)

### Przeglądanie zleceń złożenia zamówienia jednostkowego

Po pomyślnym zalogowaniu następuje przeniesienie do głównego widoku wyświetlającego listę wszystkich zleceń złożenia zamówień jednostkowych. Lista zaimplementowana jest przy pomocy *ListView*, którego elementy dostarczane są poprzez strumień o nazwie *orderRequest*. *OrderRequestBloc* zarządza wcześniej wspomnianym strumieniem, wysyłając do widoku wszystkie potrzebne dane. Realizowane jest to przez wysłanie zapytanie do serwera z prośbą o dostarczenie wszystkich zleceń złożenia zamówień jednostkowych dla aktualnego użytkownika. Zwracana wartość ma typ *Future*, którego zastosowanie, umożliwia asynchroniczne wykonywanie aplikacji, dzięki czemu interfejs jest nadal responsywny podczas oczekiwania na odpowiedź. W przypadku, gdyby nie zastosowano tego rozwiązania, program nie byłby responsywny aż do momentu otrzymania odpowiedzi od serwera. Odpowiedź zostanie dostarczona w późniejszym czasie, a *OrderRequestBloc*, w momencie pojawienia się danych, przekaże je do StreamSink, który prześle je dalej do strumienia nasłuchiwanego w UI. Interfejs użytkownika wyświetlający listę zleceń złożenia zamówień nasłuchuje wspomniany strumień. Robi to przy pomocy widżetu *StreamBuilder.* Wykorzystanie wzorca BLoC, strumieni oraz widżetu *StreamBuilder* powoduje, żeUI aplikacji reaguje natychmiastowo na zmiany wyświetlanych informacji. Zastosowana implementacja pozwala na uzyskanie separacji kodu odpowiedzialnego za wygląd, od tego odpowiedzialnego za logikę programu (w tym zapytań do serwera).



Kliknięcie na kartę reprezentującą zlecenie zamówienia, dla którego złożono zamówienie jednostkowe powoduje przeniesienie do widoku wyświetlającego listę wybranych posiłków. Jeżeli zamówienie nie zostało jeszcze złożone, kliknięcie to przenosi do widoku zamawiania.

W przypadku gdy czas na złożenie zamówienia minął, kliknięcie na ową kartę nie powoduje żadnej reakcji.

Każda z kart wyświetla takie informacje jak:

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

Rysunek 8.9 Główny widok prezentujący listę zleceń zamówienia. (źródło: praca własna)

To jaki jest status danego zlecenia możemy stwierdzić po informacji wyświetlanej w lewym dolnym rogu karty. Dla łatwiejszej nawigacji komunikat oznaczono kolorem.

### Wyświetlanie złożonego zamówienia



Rysunek 8.10 Widok prezentujący złożone zamówienie (źródło: praca własna)

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

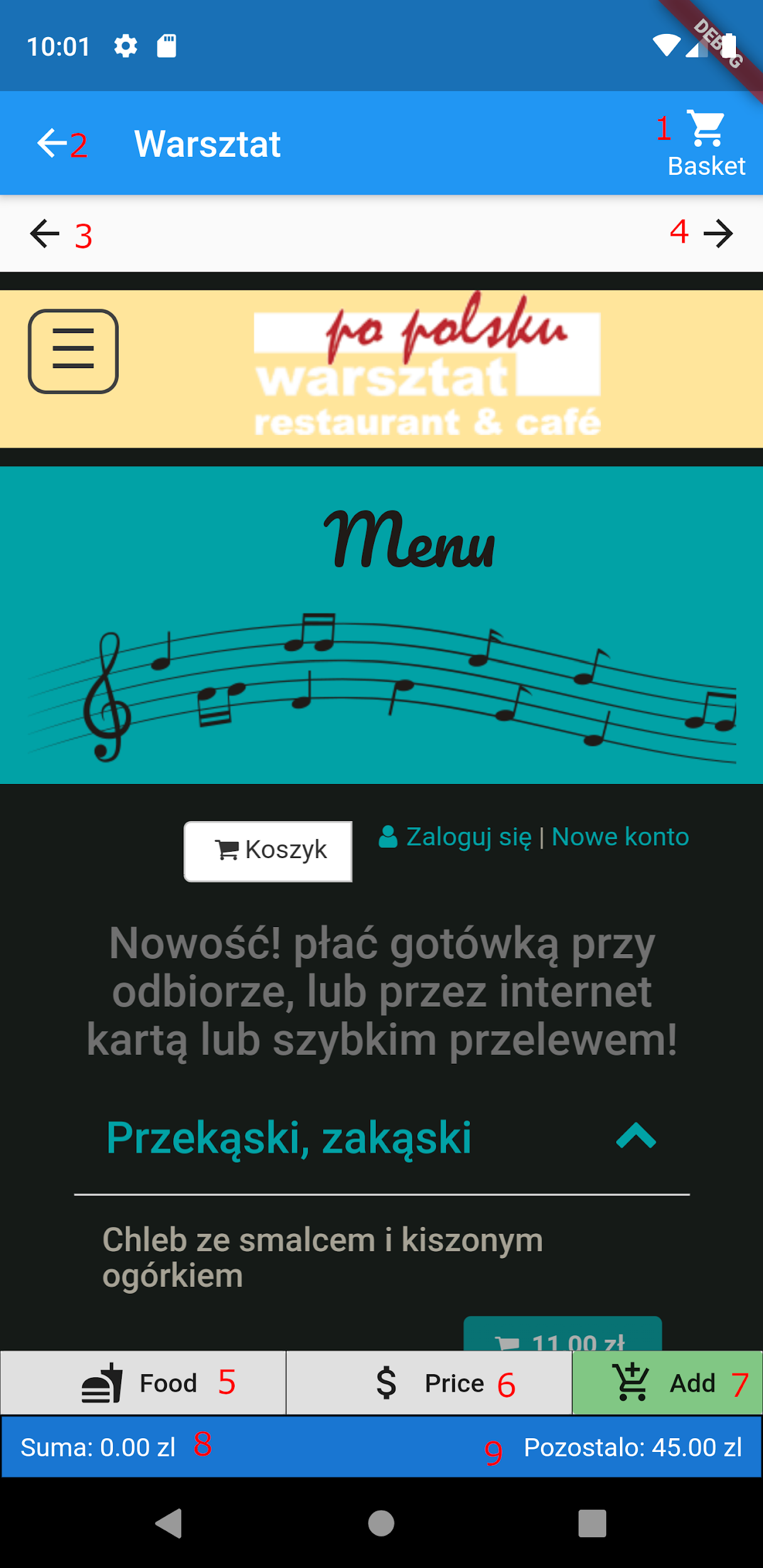
Komentarze do posiłków (o ile je dodano) wyświetlane są w nawiasach pod poszczególnymi pozycjami.

Na dole widoku dodano belkę z informacją o sumarycznym koszcie zamówienia.

### Składanie zamówień

#### Wyświetlanie menu

Widok zamawiania, wyświetlany po kliknięciu na kartę reprezentującą niezłożone zamówienie, składa się z następujących elementów:



Rysunek 8.11 Widok zamawiania posiłku (źródło: praca własna)

**1** – przycisk umożliwiający przejście do koszyka, gdzie widoczne są nazwy wszystkich wybranych posiłków,

**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,

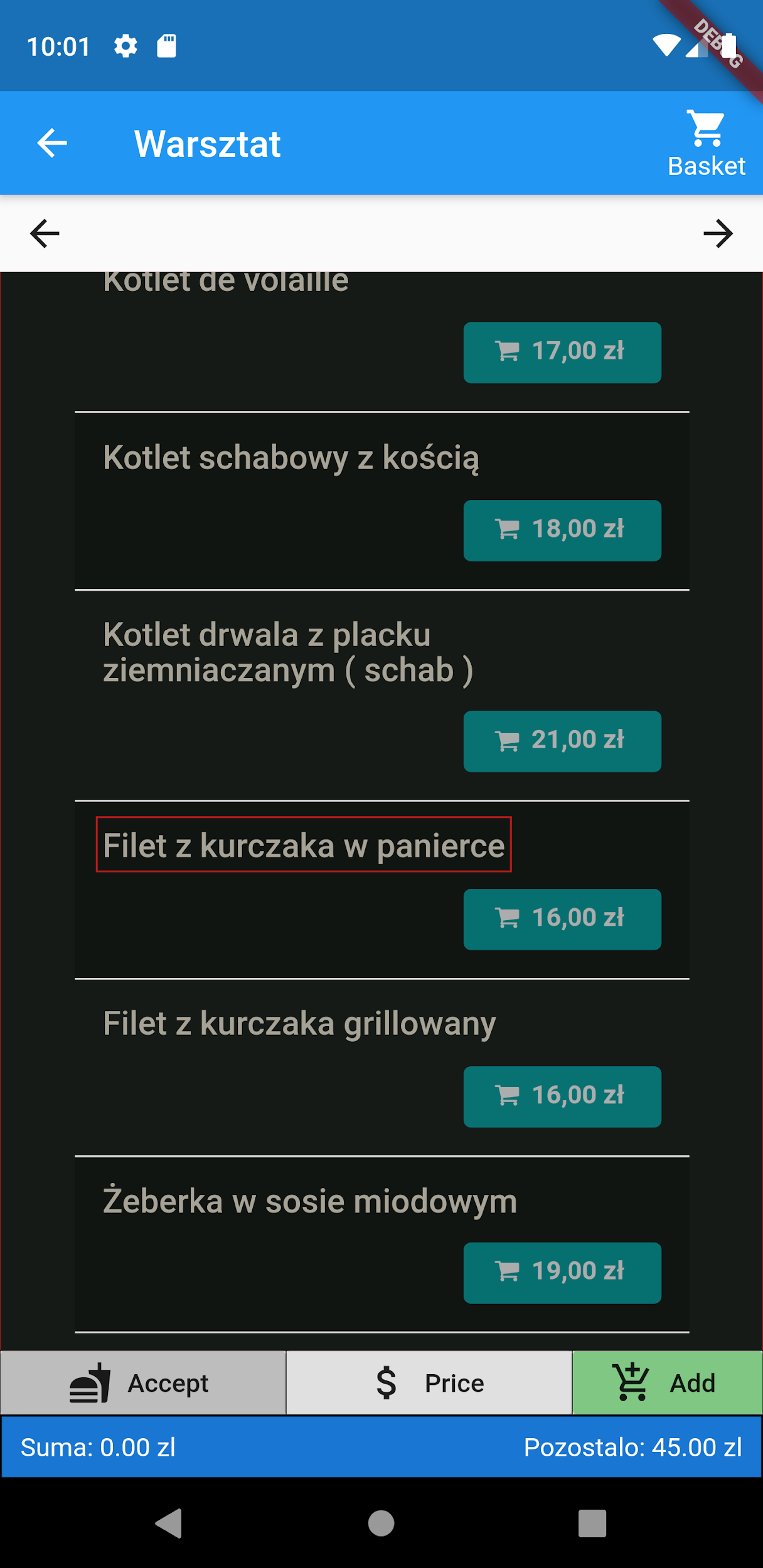
7 – dodawanie do koszyka zaznaczonego posiłku. Po kliknięciu mamy możliwość podania liczby sztuk lub dodania komentarza,

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

**9** – kwota, która pozostała do wydania, obliczona na podstawie aktualnej wartości pozycji w koszyku i limitu dla zlecenia,

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

#### Wybieranie posiłków



Wybieranie posiłków realizowane jest przy pomocy narzędzi uruchamianych z dolnego paska narzędzi.

Po kliknięciu na przycisk *Food* uruchamia się tryb zaznaczania nazwy dania. Posiłek wybierany jest poprzez przeciągnięcie palcem po ekranie tak, aby nazwa znalazła się wewnątrz prostokąta. Zatwierdzenie zaznaczonego obszaru wykonywane jest poprzez kliknięcie na *Accept*. Proces wskazywania ceny przebiega analogicznie, używając przycisku *Price*. Ostatecznie wszystkie zaznaczone dane zatwierdzane są kliknięciem na *Add*.

Rysunek 8.12 Przykład prezentujący moment zaznaczania posiłku (źródło: praca własna)

Narzędzie zaznaczania nazwy posiłków oraz ich cen zaimplementowano w oparciu o klasę *MarkingManager.*



Rysunek 8.13 Metoda build klasy MarkingManager (źródło: praca własna)

Przy implementacji klasy *MarkingManager* wspomagano się wbudowanym widżetem o nazwie *Stack*, umożliwiającym wyświetlanie widżetów jeden nad drugim. Posiada on zawsze co najmniej jeden element, którym jest widżet wyświetlający zawartość strony internetowej prezentującej menu restauracji [10].

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



Rysunek 8.14 Metoda initState klasy MarkingManager (źródło: praca własna)

Widżet wyświetlający menu ustawiany jest jako pierwszy element stosu, poprzez dodanie go do listy *\_stackContent*, która następnie przekazywana jest do widżetu *Stack*. Powoduje to, że menu osadzone jest najgłębiej.

W momencie, gdy tworzy się stan dla widżetu *MarkingManager*, robiony jest zrzut ekranu przedstawiający stronę internetową, który później widnieje w miejscu oryginalnego widoku. Pozwoliło to na uzyskanie efektu „zamrożenia” tego, co wyświetla przeglądarka. Zastosowanie opisanego rozwiązania podyktowane jest istnieniem stron internetowych dynamicznie zmieniających swój wygląd, dla których zaznaczanie posiłków byłoby utrudnione, gdyż zawartość strony przesuwałaby się.

*ContentMarker* to widżet, który otrzymuje wcześniej zrobiony zrzut ekranu i na nim wyświetla prostokąt symbolizujący zaznaczony obszar. Posiada on *GestureDetector*, który wykrywa dotknięcia wyświetlacza, dzięki czemu możliwe jest aktualizowanie wielkości zaznaczonego ekranu.

W momencie, gdy użytkownik naciśnie przycisk *Accept* potwierdzający zakończenie zaznaczania, wywoływana jest metoda *getMarked* widżetu *ContentMarker*, która pobiera informacje o zaznaczonym obszarze oraz wykonany wcześniej zrzut ekranu przedstawiający menu. Następnie wycina zaznaczony obszar w celu późniejszego przetworzenia. Całe zadanie jest dosyć złożone i powodowało chwilowe braki responsywności interfejsu w trakcie jego wykonywania, więc realizowane jest na osobnym *Isolate*, będącym Flutterowym odpowiednikiem wszechobecnie znanego wszystkim programistom wątku. Służy do tego metoda *compute*, która przyjmuje funkcję do wykonania oraz dokładnie jeden argument, który jest przekazywany do tej funkcji [10]. Ze względu na ograniczenie liczby argumentów dostarczane one są jako mapa, a następnie przesyłane do funkcji pomocniczej *createImageOutOfMarkedRectAndBackground*, która będzie wykonana na osobnym *Isolate*. Funkcja ta przyjmuje tylko jeden argument.



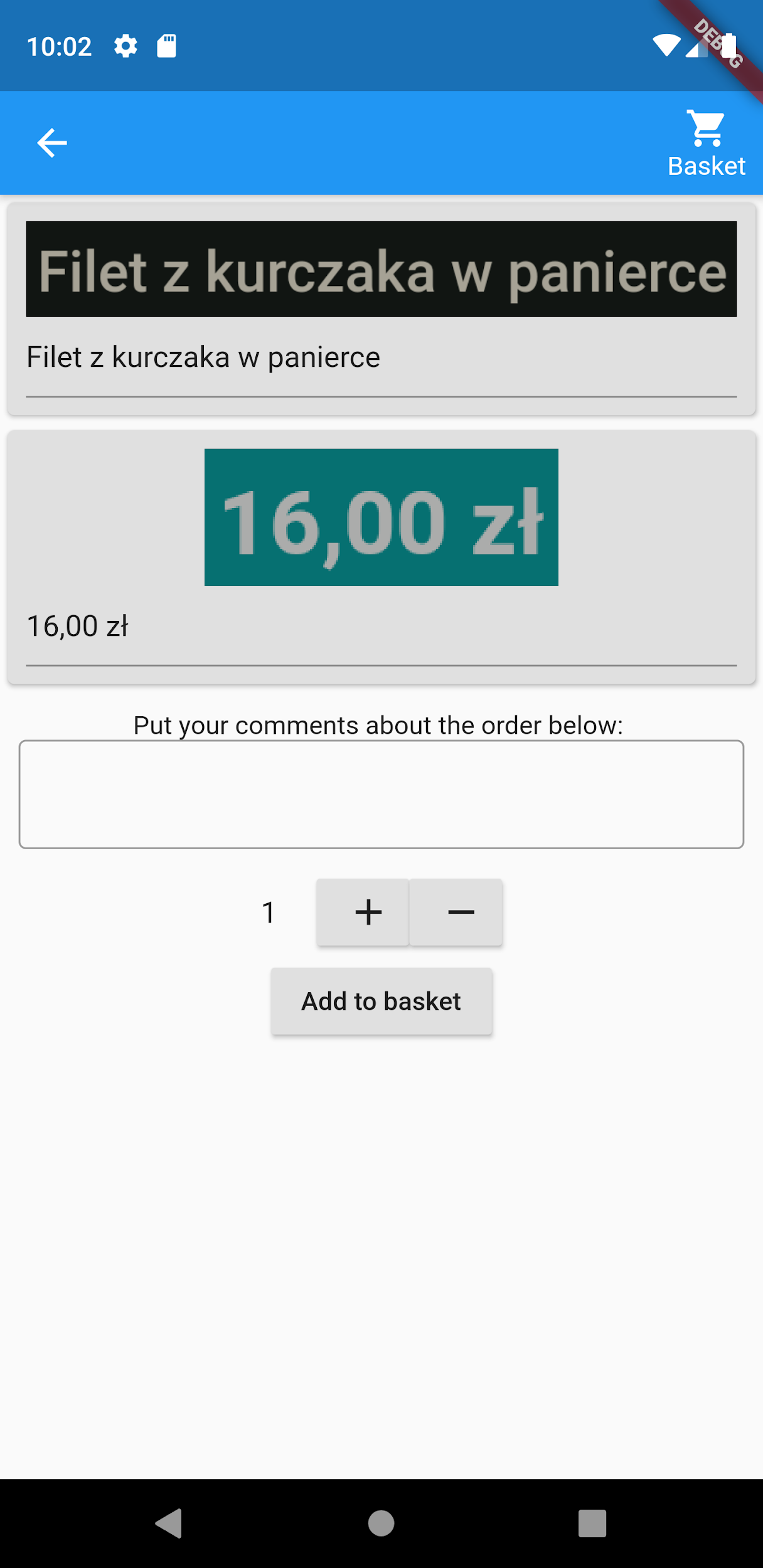
Rysunek 8.15 Metoda getMarked klasy ContentMarker (źródło: praca własna)

**

Rysunek 8.16 Metoda pomocnicza używana w metodzie getMarked() (źródło: praca własna)

Po otrzymaniu wyciętych części zrzutów ekranu przedstawiających tylko zaznaczone obszary, klient mobilny przesyła je do serwera, gdzie przy pomocy biblioteki OCR o nazwie *PyTesseract* są przetwarzane, a następnie zwracane zostają rozpoznane teksty na każdym z nich.

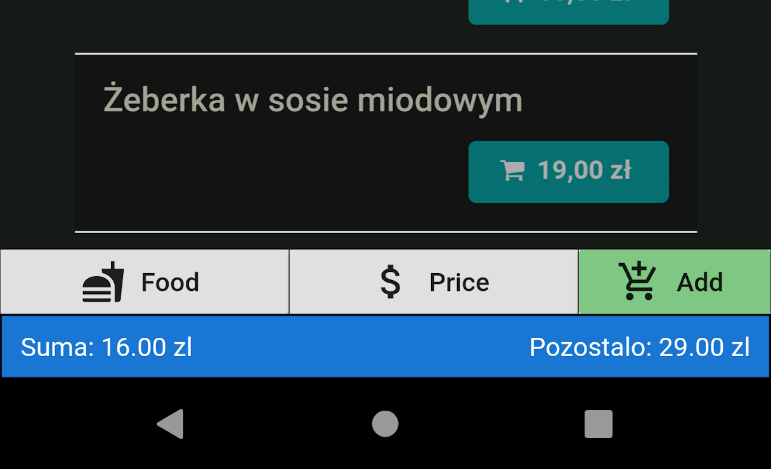
#### Korekta OCR

Po zaznaczeniu nazwy posiłku oraz jego ceny i kliknięciu na przycisk *Add* znajdującym się na dolnym pasku, następuje przeniesienie do okna dodawania posiłku.

Rysunek 8.17 Widok dodawania posiłku (źródło: praca własna)

Z poziomu tego widoku możliwe jest dodanie komentarza oraz zmiana ilości sztuk. Można zauważyć tutaj wcześniej zaznaczone obszary, a pod nimi tekst w nich zawarty. Przy użyciu niestandardowych czcionek, OCR czasami nieprawidłowo rozpoznaje tekst. Z tego powodu użytkownik ma możliwość ręcznej edycji pól tekstowych i poprawienia tekstu, gdy został on źle odczytany.

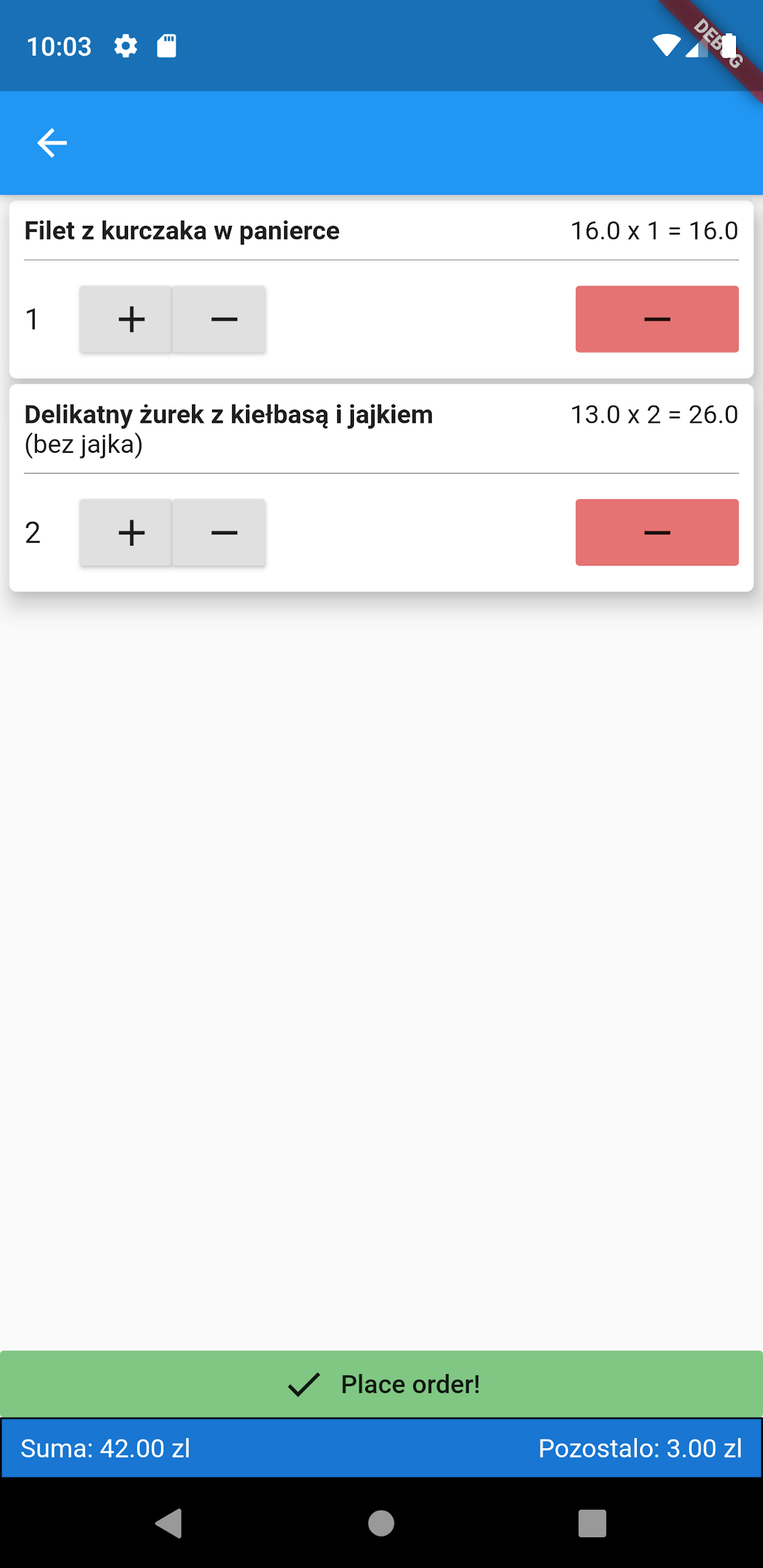
Po wprowadzeniu wszystkich potrzebnych modyfikacji i kliknięciu na *Add to basket*, nowa pozycja dołączana jest do koszyka i następuje przekierowanie do strony z menu, gdzie istnieje możliwość dodania kolejnych posiłków.



Na stronie tej widać, że dolna belka zaktualizowała dane o stanie koszyka.

Rysunek 8.18 Dolna belka przedstawiająca informacje o stanie aktualnego zamówienia (źródło: praca własna)

#### Koszyk

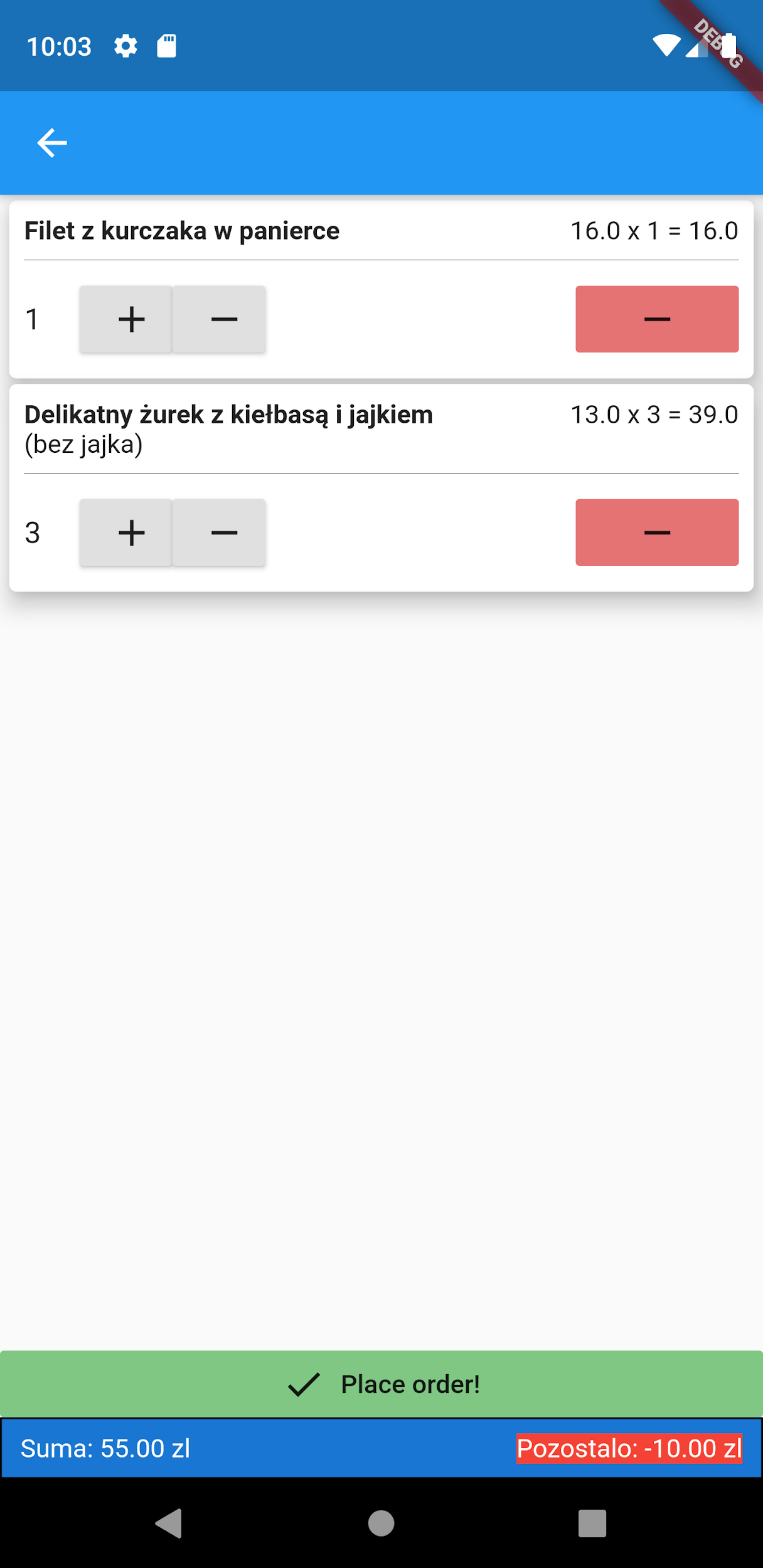


Rysunek 8.19 Koszyk z zamówieniami (źródło: praca własna)

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

Z poziomu tego widoku możliwa jest aktualizacja ilości danego posiłku lub całkowite jego usunięcie.

Dodane komentarze są wyświetlane w nawiasach. Widoczne jest to przy drugim posiłku.

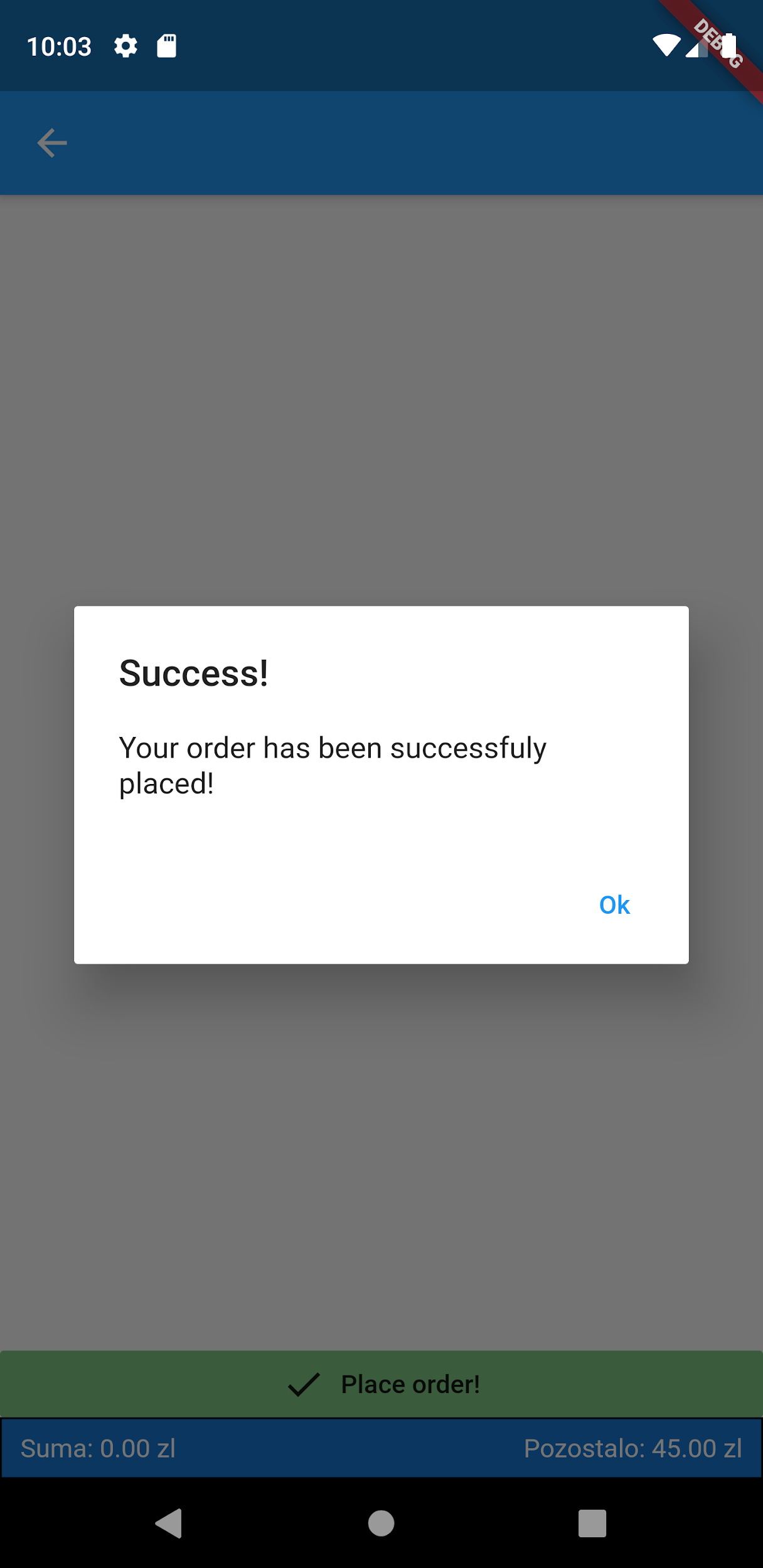


Zmiany wprowadzone z poziomu koszyka dynamicznie aktualizują wszystkie kalkulacje. Dla przykładu zwiększono liczbę sztuk drugiego posiłku z dwóch na trzy. Zauważalne jest, że cena sumaryczna dla danej pozycji (znajdująca się po prawej stronie) została przeliczona, jak i również całkowity koszt widoczny na dolnej belce.

Ponadto ze względu na przekroczony limit, program zmienił kolor tła tekstu wyświetlającego pozostałe środki.

Rysunek 8.20 Widok koszyka z zamówieniami przy przekroczonym limicie cenowym (źródło: praca własna)

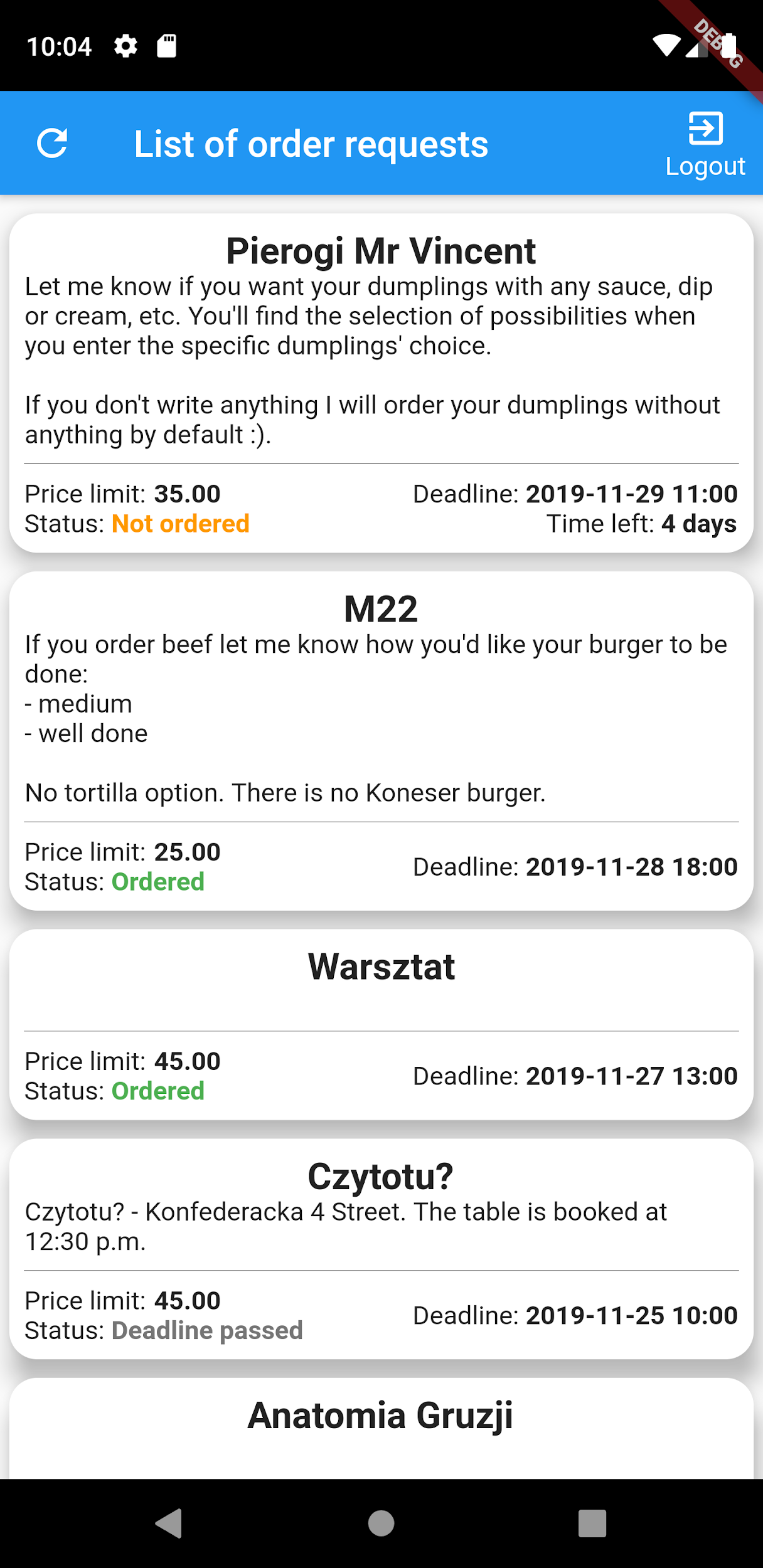
#### Składanie zamówienia



Rysunek 8.21 Komunikat potwierdzający powodzenie złożenia zamówienia (źródło: praca własna)

Jak wspomniano w rozdziale opisującym koncepcję rozwiązania, przekroczenie limitu ceny nie powoduje braku możliwości złożenia zamówienia. Możliwa jest finalizacja poprzez kliknięcie na przycisk *Place order!*. Powoduje to wysłanie listy wybranych posiłków do serwera, które zostaną odpowiednio przetworzone, a na ekranie pojawi się komunikat potwierdzający przyjęcie zamówienia.

Po kliknięciu na przycisk *Ok*, zostajemy przekierowani do strony głównej.



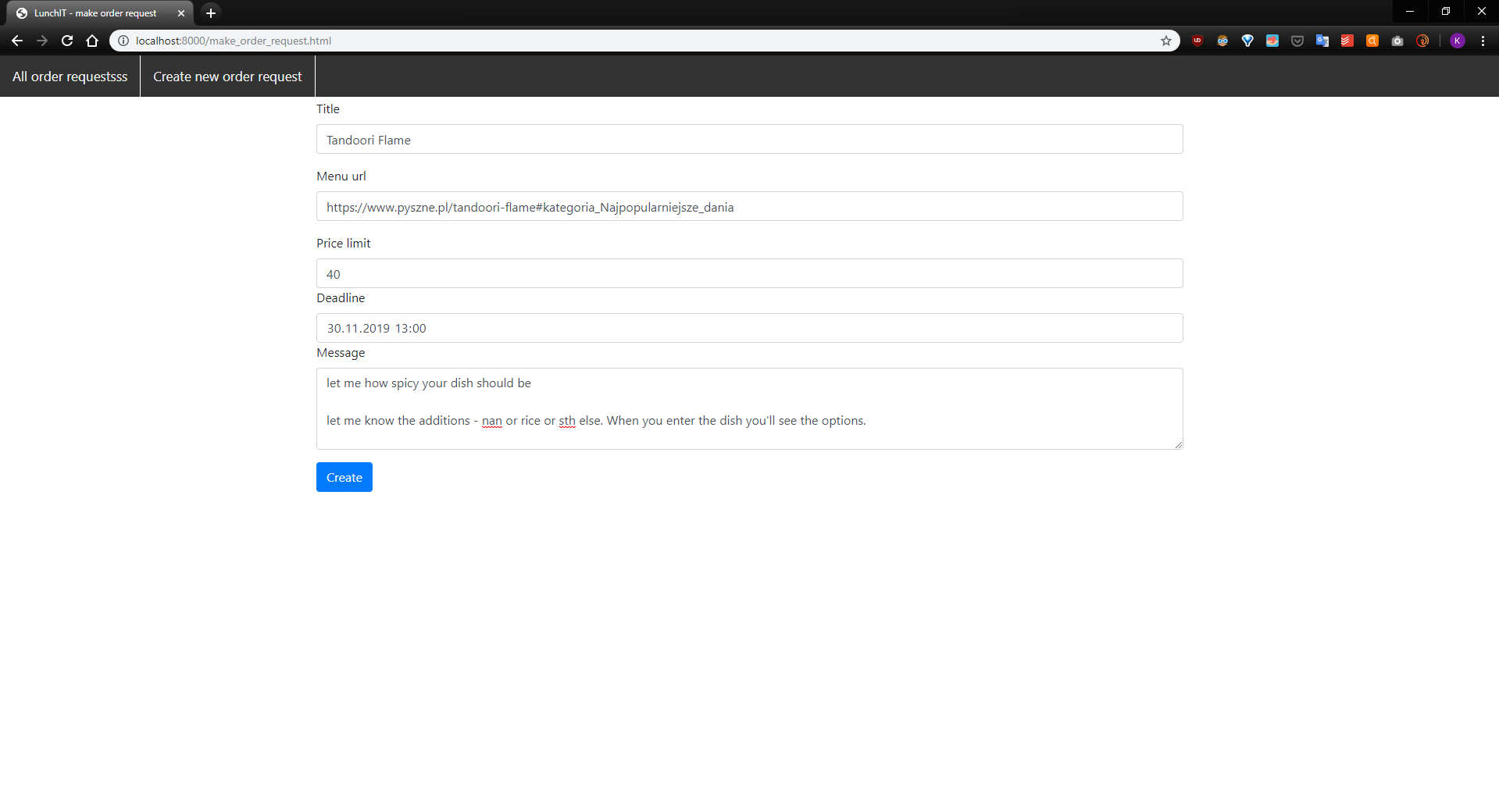
Rysunek 8.22 Zrzut ekranu prezentujący zmianę statusu zamówienia o nazwie „Warsztat” (źródło: praca własna)

Dołączony zrzut ekranu przedstawia zmianę statusu przy złożonym zamówieniu (trzecia karta od góry – „Warsztat”), który został zaktualizowany na wartość „Ordered”.

## Aplikacja webowa

### Tworzenie nowego zlecenia

Po przejściu do zakładki *Create new order request*, użytkownik ma możliwość stworzenia nowego zlecenia zamówień. Pola *Price limit* oraz *Deadline* umożliwiają wprowadzenie tylko konkretnych typów wartości. Po uzupełnieniu wszystkich informacji oraz kliknięciu na przycisk Create, wysyłane jest do serwera zapytanie, którego przetworzenie powoduje dodanie odpowiedniego wpisu w bazie danych. Użytkownik zostaje wtedy przeniesiony do strony wyświetlającej wszystkie zlecenia, gdzie może zobaczyć również to nowo utworzone.

**

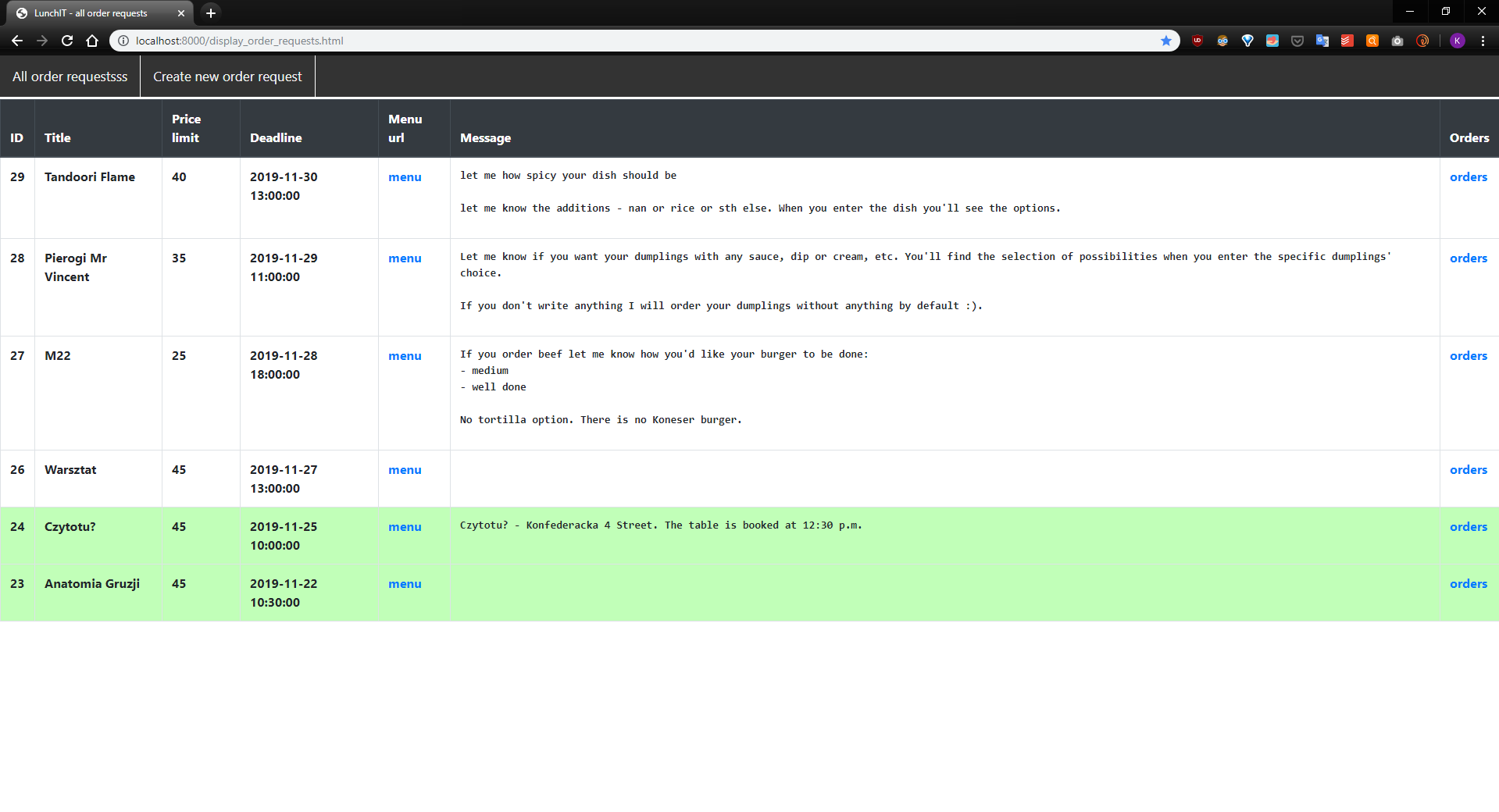
Rysunek 8.23 Widok podstrony z tworzeniem nowych zleceń (źródło: praca własna)

### Wyświetlanie listy zleceń

Po przejściu do zakładki *All order requests* wyświetlone zostają wszystkie zlecenia złożenia zamówień. Dla wygody zostały posortowane malejąco względem kolumny *Deadline,* a więc zlecenia złożenia zamówień, dla których czas dostępny na wybór posiłku jest najdłuższy, zostaną wyświetlone na samej górze listy.

Wiersze listy przedstawiające zlecenia, dla których ostateczny czas na złożenie zamówienia minął, mają kolor zielony, w celu łatwiejszego odnalezienia pozycji, które należy zrealizować w restauracji.

Odnośnik *orders* znajdujący się w ostatniej kolumnie umożliwia przejście do strony wyświetlającej zamówienie grupowe dla danego zlecenia.



Rysunek 8.24 Widok podstrony wyświetlającej wszystkie zlecenia (źródło: praca własna)

Strona wyświetlająca listę zleceń została zaimplementowana przy pomocy tabeli, której wygląd zdefiniowano używając zewnętrznej biblioteki CSS o nazwie *Bootstrap*.

Na początku kodu HTML zadeklarowany jest kontener o id=”*navbar*”, który później zostanie wypełniony paskiem nawigacji załadowanym z osobnego pliku. Takie rozwiązanie zostało zastosowane, aby uniknąć powtórzeń deklaracji paska nawigacji w każdej z podstron. Dodatkowo pozwala to na łatwiejsze dołączanie do niego nowych podstron (brak konieczności edycji kilku plików z definicją).

Następnie tworzona jest główna struktura tabeli. Deklarowane tam są nazwy nagłówków oraz nazwy, które nadają tabeli odpowiedni wygląd [2].

**

Rysunek 8.25 Deklaracja tabeli wyświetlającej zlecenia zamówień (źródło: praca własna)

W momencie, gdy cała strona zostanie załadowana, na początku części skryptowej strony wczytuje się pasek nawigacji z pliku *navbar.html*. Później 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* ułatwiającą to zadanie [3].

**

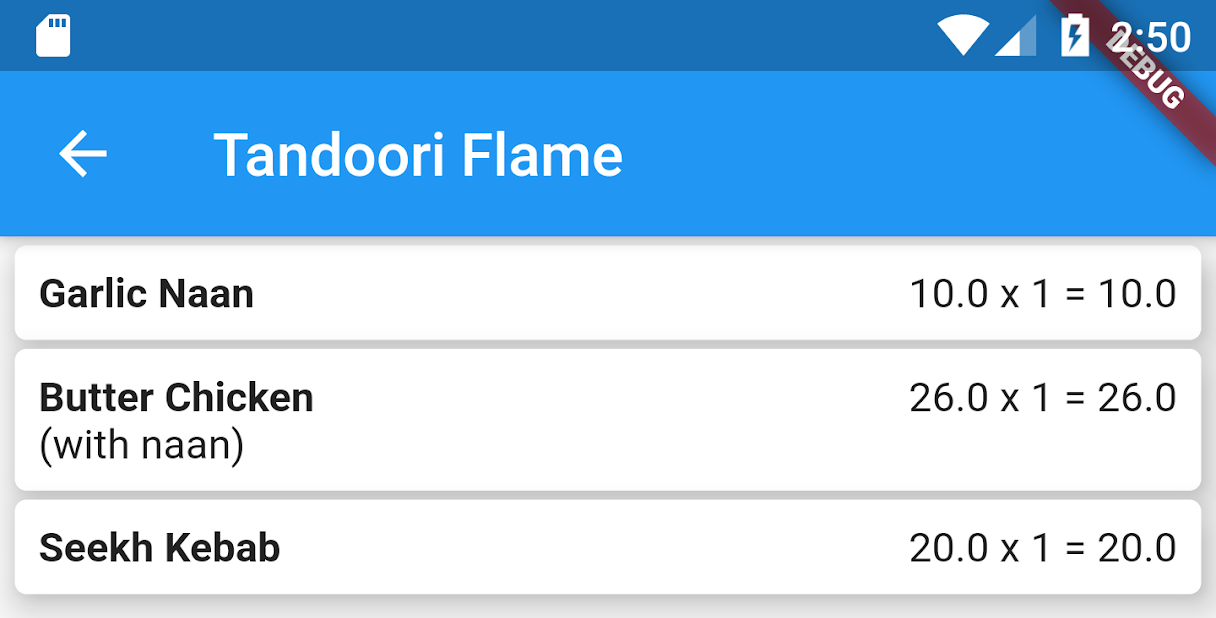
Rysunek 8.26 Część skryptowa strony display\_order\_requests.html, która pobiera dane z serwera i ładuje je do tabeli (źródło: praca własna)

### Wyświetlanie zamówienia grupowego

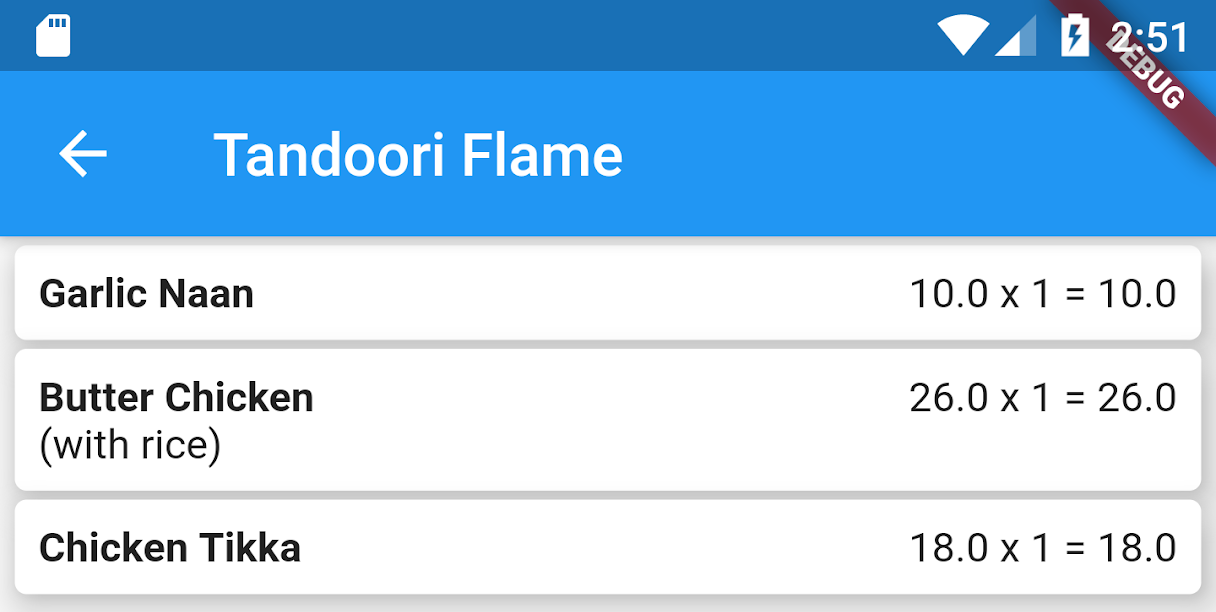
Dla zaprezentowania wyświetlenia zamówienia grupowego najpierw złożono trzy zamówienia jednostkowe (każde z innego konta). Prezentują się one następująco:

**

Rysunek 8.27 Zamówienie jednostkowe – użytkownik nr 1 (źródło: praca własna)

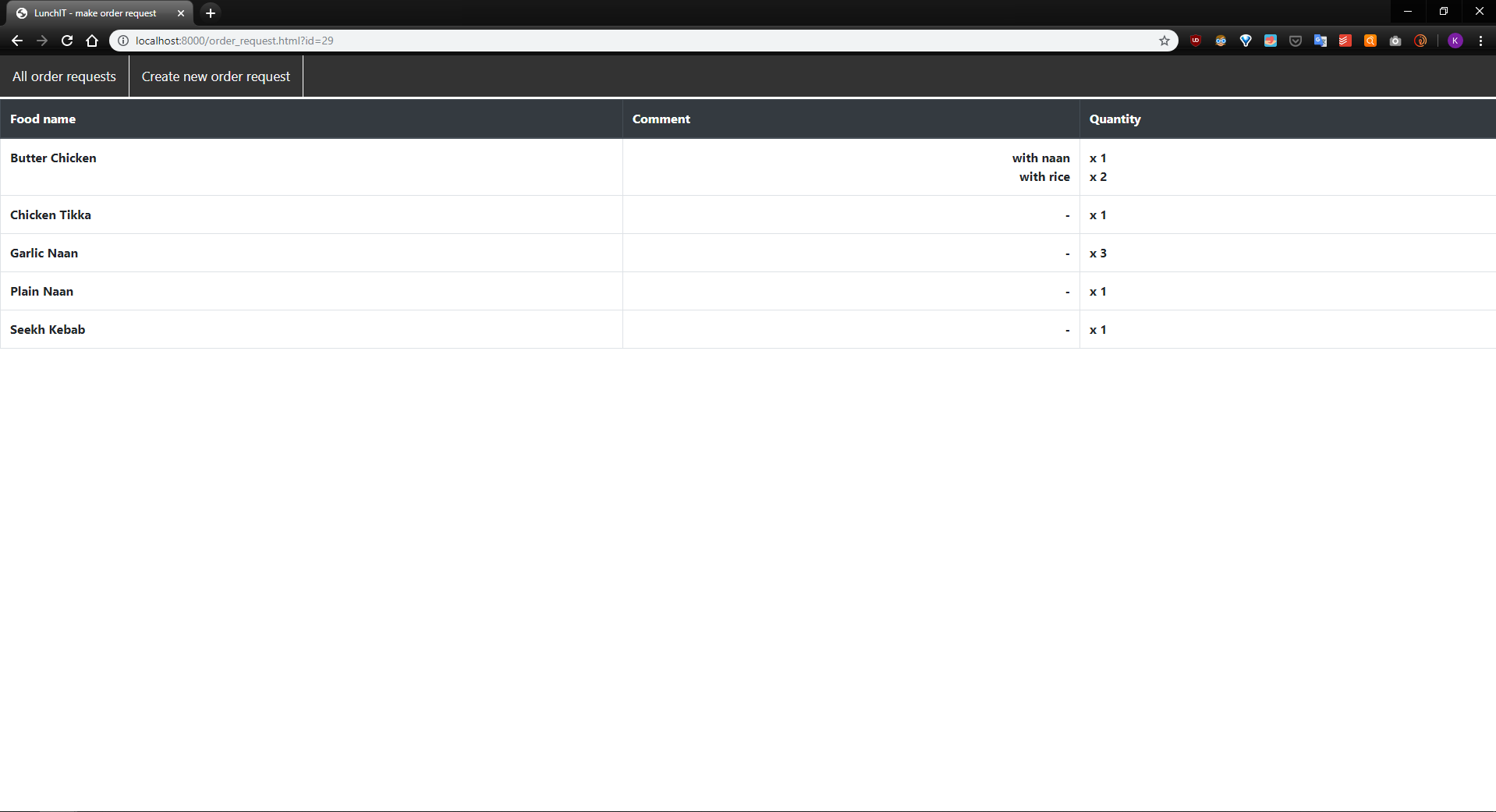
**

Rysunek 8.28 Zamówienie jednostkowe – użytkownik nr 2 (źródło: praca własna)

**

Rysunek 8.29 Zamówienie jednostkowe – użytkownik nr 3 (źródło: praca własna)

Zamówienia o takiej samej nazwie i zawierające ten sam komentarz są grupowane w jedną pozycję, a ich ilość jest odpowiednio zwiększana. Ze względu na to, że każdy z użytkowników złożył zamówienie na posiłek o nazwie Garlic Naan i nie dodał żadnego komentarza, na liście grupowej widnieje on tylko raz, a jego wartość w kolumnie reprezentującej ilość sztuk wynosi 3. Zgrupowane również zostało zamówienie danie Butter Chicken, gdzie dodany komentarz to „with rice”.



Rysunek 8.30 Zamówienie grupowe wygenerowane z wcześniej przedstawionych zamówień jednostkowych (źródło: praca własna)

Implementacja strony prezentującej zamówienia grupowe została wykonana w bardzo podobny sposób, jak omówiona w poprzednim rozdziale strona wyświetlająca listę wszystkich zleceń zamówień.

## Serwer – dokumentacja zasobów

Wszystkie zasoby zwracające odpowiedź w formie obiektu JSON mogą zawierać klucz *error* powiązany z wartością o typie *string*. Klucz dostępny jest wówczas, gdy wystąpił błąd. Wartość ta zawiera informację opisującą napotkany problem.

Dostęp do zasobów wymagających autoryzacji można uzyskać poprzez dodanie nagłówka o nazwie *AUTHORIZATION,* którego wartość przyjmuje postać *L:H*, gdzie *L* to login, a *H* 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 e-mail,
* [*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, które są nazwami posiłków. Z kolei ich wartości to obiekty JSON o kluczach będących komentarzami do zamówionych pozycji. Wartości przypisane do tych kluczy to liczby całkowite reprezentujące ilość osób, które złożyły 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 dla zlecenia identyfikowanego przez parametry zapytania

**zasób**: /api/has\_ordered

**metoda**: GET

**wymaga autoryzacji**: tak

**parametry**:

* [*int*] order\_request\_id – identyfikator zlecenia 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 do odczytania

**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 składane jest 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 – identyfikator zamówienia jednostkowego

**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**: służy do tworzenia 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 – identyfikator utworzonego zlecenia

**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 pieniężnydla 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",

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

"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 z 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 – identyfikator złożonego przez użytkownika zamówienia jednostkowego powiązanego 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",

"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",

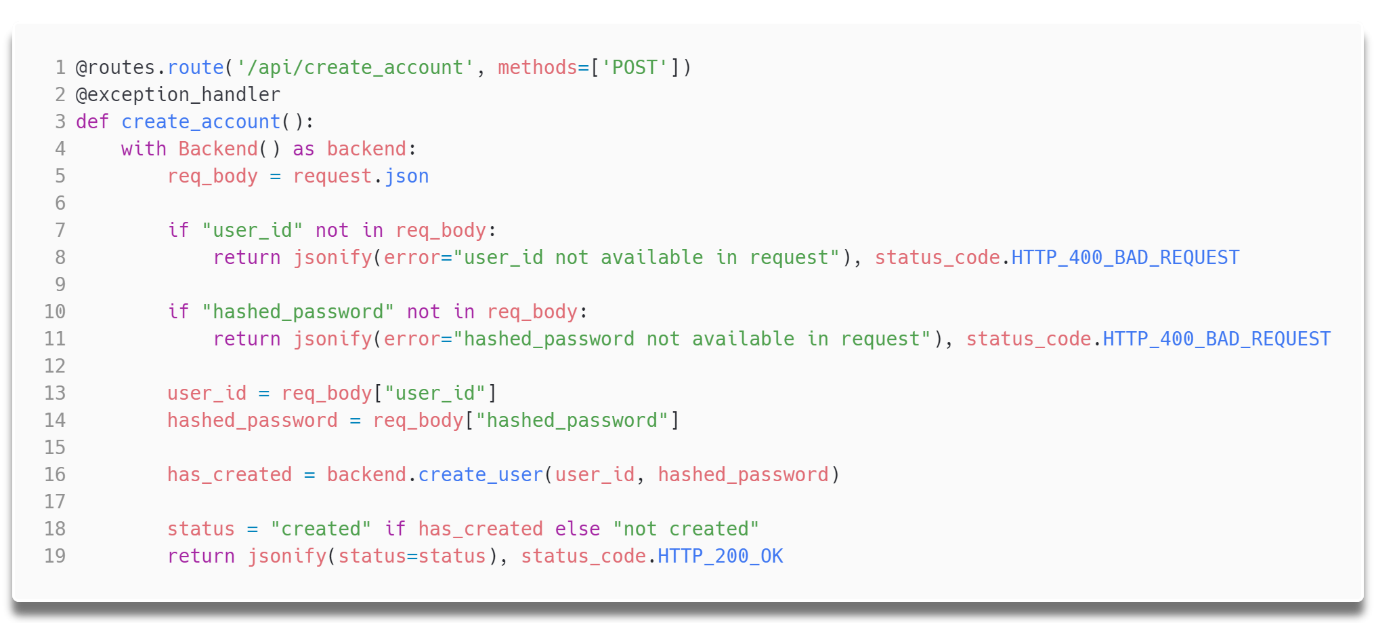
"price\_limit": 45,

"placed\_order\_id": 55

}]}

## Kody zapytań oraz obsługa wyjątków

Każdy z zasobów zwraca odpowiednie kody odpowiedzi HTTP, pozwalające na sprawdzenie informacji o realizacji zapytania [6]. Przykładowo dla zapytania wysłanego w celu złożenia zamówienia otrzymamy kod odpowiedzi *400* w przypadku, gdy ciało zapytania będzie niepoprawne, *200* dla poprawnie przetworzonego zapytania lub *500* w momencie wystąpienia nieprzechwyconego wyjątku (obsługiwane przez dekorator @*exception\_handler*).



Rysunek 8.31 Implementacja zasobu służącego do stworzenia nowego użytkownika prezentująca zwracane kody odpowiedzi HTTP (źródło: praca własna)

**

Rysunek 8.32 Implementacja dekoratora exception\_handler zwracającego kod błędu 500 (źródło: praca własna)

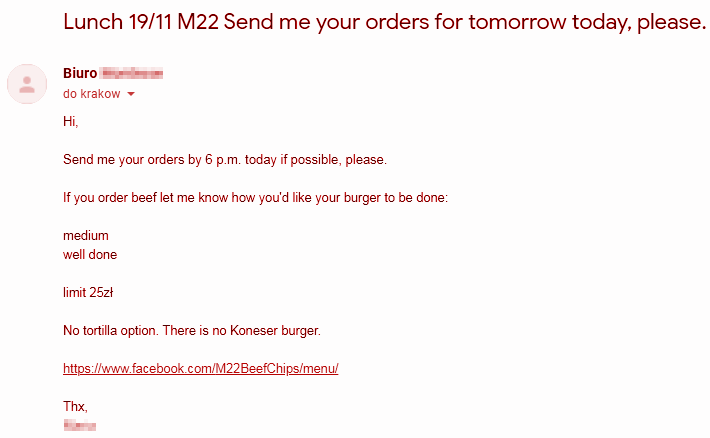
Dekorator *exception\_handler* przechwytuje nieobsłużone wyjątki rzucone w obrębie dekorowanego zasobu [8]. Zwraca odpowiedź w postaci obiektu JSON posiadającego klucz *error*. Wartość, która znajduje się pod tym kluczem zawiera informację opisującą błąd serwera. Kod odpowiedzi HTTP równy jest wtedy wartości 500.

# Baza danych

## Analiza przechowywanych danych

Aby utworzyć bazę danych konieczne było przeanalizowanie wszystkich informacji, które będzie musiała przechowywać aplikacja.

Zaczęto od przeglądu treści jednego z maili otrzymanych od osoby odpowiedzialnej za zamawianie posiłków.



Rysunek 9.1 Przykładowy e-mail o konieczności złożenia zamówienia (źródło: praca własna)

Z wiadomości tej wywnioskowano, że informacje na temat zlecenia zamówienia, 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 poniższe wnioski:

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

Użytkownik zawsze musi wskazać cenę posiłku. Ze względu na to zdecydowano, że również ta informacja będzie przechowywana w bazie danych. Pozwoli to w przyszłości na utworzenie nowych funkcjonalności – np. możliwe będzie utworzenie funkcji podpowiadającej zestawy obiadowe na podstawie poprzednio złożonych zamówień – aby nie przekroczyć limitu cenowego konieczna będzie znajomość cen posiłków.

Ostatecznie postanowiono przechowywać w bazie danych poniższe informacje o zamówionych posiłkach:

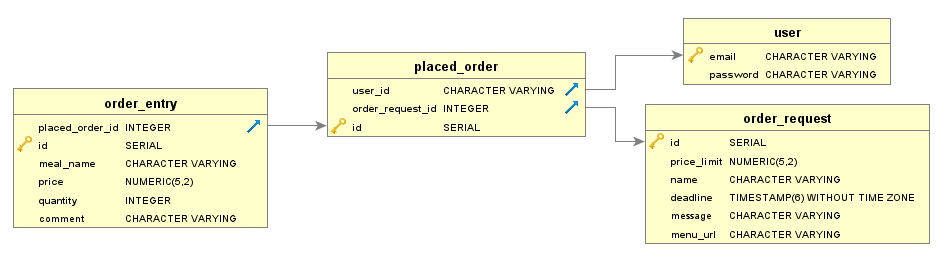
* nazwa,
* cena,
* ilość,
* dodatkowy komentarz.

Przez 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 wyników analiz omówionych w poprzednim rozdziale i utworzeniu na ich podstawie tabel przechowywanych w bazie danych, utworzono schemat, który prezentuje się następująco [11]:



Rysunek 9.2 Schemat bazy danych aplikacji (źródło: praca własna)

Zawarto w nim cztery tabele:

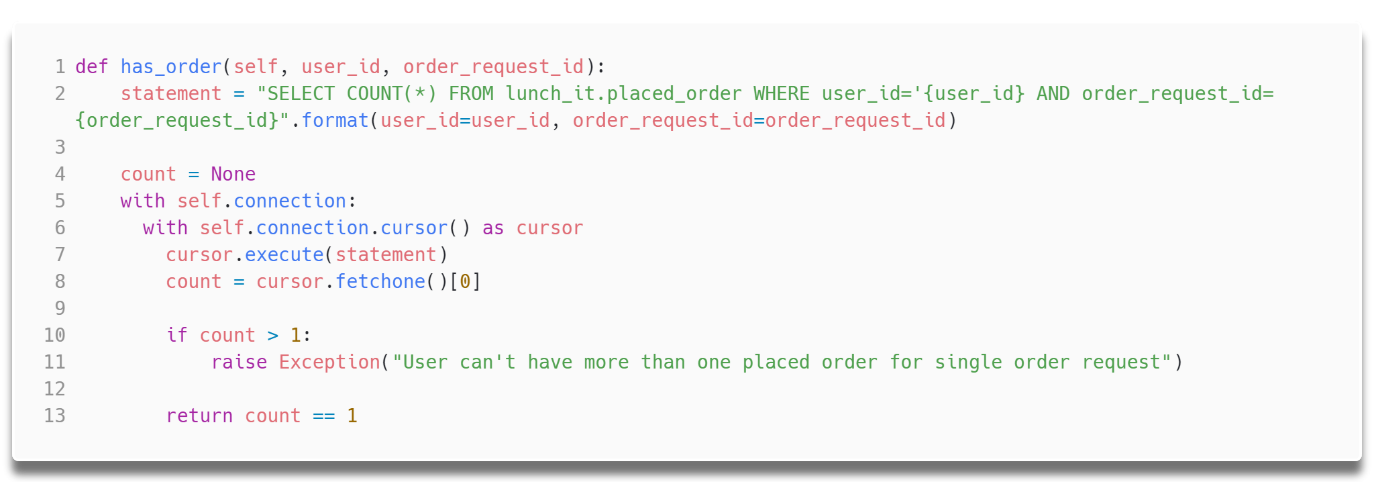
* *order\_entry* – znajdują się w niej posiłki należące do złożonego zamówienia,
* *placed\_order* – wiersze reprezentują złożone zamówienia,
* *order\_request* – posiada zlecenia złożenia zamówień jednostkowych,
* *user* – gromadzi informacje o danych logowania użytkowników aplikacji.

# 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 temu możliwe jest wykonanie dowolnego zapytania SQL.

W aplikacji początkowo nieświadomie zaimplementowano zapytania do bazy danych w taki sposób, że były podatne na taki rodzaj ataku. Atakujący miał możliwość wykonania zapytania do serwera, podając zamiast jednego z argumentów, dowolne odpowiednio sformatowane zapytanie SQL. Należało dodać na jego końcu i początku znak “;” oznaczający zakończenie wyrażenia.

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 (źródło: praca własna)

Przesyłając do serwera zapytanie sprawdzające, czy użytkownik złożył zamówienie, podając złośliwy kod SQL jako order\_request\_id (np. “;drop database lunch\_it;”), zapytanie do bazy danych po dodaniu argumentów wyglądałoby następująco:

**SELECT COUNT(\*) FROM lunch\_it.placed\_order WHERE user\_id=sample\_user AND order\_request\_id=;drop database;**;

Po wywołaniu metody *cursor.execute(statement)* wykonane byłyby 2 komendy. Pierwszą z nich oznaczono powyżej zielonym tłem, natomiast drugą, będąca złośliwym kodem „wstrzykniętym” przez atakującego, pomarańczowym.

Po dostrzeżeniu podatności na atak SQL injection w zaprezentowanej implementacji, poprawiono logikę odpowiedzialną za zapytania do bazy danych i usunięto możliwość dokonania opisanego ataku. Wykonano to przy pomocy biblioteki *psycopg2,* która, jak później dostrzeżono, oferuje odpowiednie zabezpieczenie realizowane przez sprawdzanie argumentów zapytania. Konieczne jest wówczas odpowiednie oznaczenie miejsc na argumenty w łańcuchu znakowym, będącym zapytaniem SQL oraz przekazaniu go wraz z drugim parametrem, czyli mapą argumentów do ów zapytania SQL.

**

Rysunek 10.2 Implementacja bez luki pozwalającej na „wstrzyknięcie” złośliwego kodu SQL (źródło: praca własna)

## Szyfrowanie danych

Ze względu na charakter programu i możliwości, jakie udostępnia, niemożliwe jest wyrządzenie szkód materialnych przez niepożądane osoby, które uzyskały dostęp do czyjegoś konta w aplikacji. Problem pojawia się, gdy wykradzione dane logowania zostały użyte przez użytkownika również w innych serwisach (np. w aplikacji bankowej). Z tego względu hasła, które są przesyłane między klientem a serwerem lub przechowywane w bazie danych są zaszyfrowane. Dzięki temu, gdy osoba niepożądana zdoła wykraść hasło, nie będzie miała możliwości otrzymania dostępu do innej aplikacji/strony, w której istnieje konto o tych samych danych dostępowych.

## Autoryzacja zasobów

**

Rysunek 10.3 LoginManager zarządzający użytkownikami i procesem autoryzacji (źródło: praca własna)

Autoryzacja zasobów po stronie serwera zaimplementowano w oparciu o bibliotekę *flask\_login*. Wymaga ona zadeklarowania obiektu *LoginManager,* który zostaje powiązany z obiektem *Flask*. Następnie dostarczana jest funkcja (odpowiednio oznaczona dekoratorem) przyjmująca jako parametr zapytanie wysłane do serwera. W przypadku gdy autoryzacja się powiodła, owa funkcja powinna zwracać obiekt *User.* Gdy zapytanie nie zostało zautoryzowane – *None*.

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

**

Rysunek 10.4 Implementacja klasy User (źródło: praca własna)

Ostatnią czynnością jest dodanie dekoratora @*login\_required* do wszystkich zasobów, które wymagają autoryzacji.



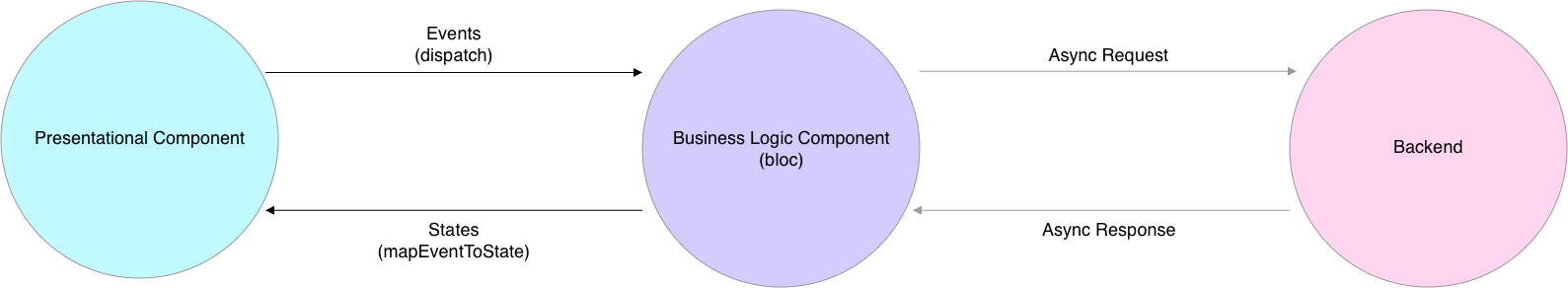
Rysunek 10.5 Przykład użycia dekoratora @login\_required, który powoduje, że zasób wymaga autoryzacji. (źródło: praca własna)

Wyżej wymienione działania spowodują, że w momencie wykonania zapytania do serwera pod adres, gdzie wymaga się autoryzacji, najpierw sprawdzane jest, czy zapytanie powinno być dopuszczone do zasobu. Gdy przesłane zostały dane autoryzacji i są one poprawne, zapytanie zostanie przetwarzane w normalny sposób. W przeciwnym przypadku zwracana jest odpowiednia informacja i nie dochodzi do sfinalizowania zapytania.

# 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 - dostęp 2019-11-15)

Wspomniany wyżej wzorzec pozwala na odseparowanie warstwy prezentacji od logiki biznesowej. Dodatkowoułatwia testowanie aplikacji oraz ponowne wykorzystywanie komponentów. Zapewniają to strumienie, które przesyłają zdarzenia i stany.

Dla łatwiejszego zobrazowania tego, jak funkcjonuje zastosowana architektura, dla przykładu użyty zostanie widżet, który po naciśnięciu przycisku wyświetla dane otrzymane z serwera. Przy zastosowaniu wzorca BLoC cały proces wygląda następująco.

Wciśnięcie przycisku dodaje do jednego ze strumieni obiektu BLoC zdarzenie, które o tym sygnalizuje. Zadanie to można ukryć wewnątrz metody mającej bardziej przyjazną formę. Obiekt BLoC nasłuchuje strumienia zdarzeń i realizuje odpowiednie operacje w zależności od typu otrzymanego obiektu. Na przykład dla zdarzenia wciśnięcia przycisku wykona asynchroniczne zapytanie do serwera. Odpowiedź zostanie zwrócona w późniejszym czasie, a następnie dodana do strumienia stanów, nasłuchiwanego przez warstwę prezentacji. Nasłuchiwanie to realizowane jest najczęściej poprzez widżet *StreamBuilder,* który powoduje, że w momencie pojawienia się nowych danych, całe drzewo widżetów w nim osadzonych zostaje przebudowane.

Zastosowanie powyższej architektury niweluje konieczność „ręcznego” odświeżania widoku po stronie warstwy prezentacji, która jedynie wysyła zdarzenia i oczekuje stanów, nie wiedząc nic o sposobie, w jaki pozyskiwany i jak przetwarzany jest stan, który otrzymuje do wyświetlenia.

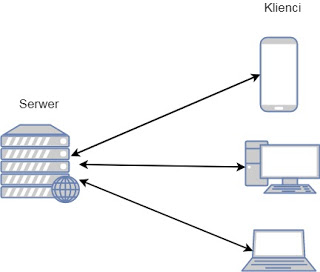
Podobnie jest z warstwą logiki biznesowej, która otrzymuje zdarzenia i w zależności od ich typów wykonuje odpowiednie czynności. W rezultacie powstaje stan (np. informacje o wszystkich dostępnych użytkownikach) odsyłany do warstwy prezentacji poprzez strumień stanów. Dzięki temu warstwa logiki biznesowej nie wpływa na sposób prezentacji danych.

## Klient-serwer

Cała aplikacja oparta jest o architekturę typu klient-serwer. W systemie takim wyróżnia się dwa rodzaje urządzeń – serwer, który oferuje usługi oraz klientów korzystających z tych usług. Serwer przechowuje wszystkie zasoby i udostępnia je przez zdefiniowane interfejsy, zwiększając bezpieczeństwo. Transfer danych pomiędzy dwoma rodzajami urządzeń odbywa się poprzez określone protokoły komunikacyjne, spośród których najpopularniejszym jest TCP/IP.

Serwer w tej architekturze posiada wszystkie informacje o strukturze sieci. Klienci nie wiedzą o istnieniu innych urządzeń tego typu i nie mają możliwości bezpośredniej komunikacji między sobą bez udziału serwera.

W aplikacji do zamawiania posiłków wyróżnia się dwa rodzaje klientów – mobilny będący aplikacją zainstalowaną na smartfonie oraz webowy, który jest użytkowany poprzez przeglądarkę.



Rysunek 11.2 Schemat architektury klient-serwer (źródło: http://architektura-oprogramowania.blogspot.com/p/architektura-klient-serwer.html - dostęp 2019-11-15)

# Podsumowanie

## Perspektywa dalszego rozwoju aplikacji

### Propozycje zamawiania posiłków

Ze względu na dane, które są przechowywane w bazie, istnieje możliwość łatwego dodania do aplikacji funkcji, którą byłoby proponowanie zestawów obiadowych. Dane te zawierają nazwy posiłków wraz z cenami. Dzięki temu możliwe jest stworzenie zestawów opartych na poprzednich zamówieniach użytkowników, które będą wpasowywać się w limit cenowy. Opisane rozwiązanie mogłoby jeszcze bardziej usprawnić proces zamawiania posiłków, ze względu na minimalizację czasu poświęconego na wybór dania oraz likwidację konieczności manualnego zaznaczania tekstowych pozycji z menu.

### Usprawnienie OCR

Skuteczność zastosowanego algorytmu rozpoznawania tekstu na obrazach nie jest stuprocentowa. Z tego powodu dodano możliwość ręcznej korekty tego, co zostało rozpoznane jako tekst. Ze względu na to, że użytkownik dostarcza korekty rozpoznanego tekstu, możliwe jest zapisywanie tych danych. Byłyby one powiązane z obrazkiem, będącym źródłem odczytanych znaków. Na podstawie tych informacji możliwe będzie stworzenie algorytmu sztucznej inteligencji, która uczyłaby się na tych danych korekcyjnych. Zabieg taki umożliwiłby poprawienie skuteczności odczytów tekstu z zaznaczanych obszarów.

### Powiadomienia

Przydatną funkcją byłoby również otrzymywanie powiadomień związanych z koniecznością złożenia zamówienia. Pozwalałoby to na uniknięcie sytuacji, w których użytkownik zapomniałby o dokonaniu wyboru dań obiadowych. Dodatkowo powiadomienia mogłyby służyć do informowania użytkowników o przewidywanym czasie dostawy.

## Wnioski

Praca inżynierska przedstawia sposób automatyzacji grupowego zamawiania jedzenia w jednej z Krakowskich firm IT. Zawarty został opis podejścia oraz utrudnienia, z którymi na co dzień zmagali się pracownicy. Zaobserwowane problemy poddano analizie, na podstawie której zaproponowano aplikację wspomagającą użytkowników w wyborze posiłków. Następnie opisano proces jej powstawania, który uwzględniał nie tylko implementacje programu, lecz także podejmowane decyzje. Zaproponowane rozwiązanie problemu w postaci aplikacji LunchIT spełnia postawione założenia i pozwala na usprawnienia procesu grupowego zamawiania posiłków. Użyte technologie umożliwiają uruchomienie całego systemu na wielu platformach oraz są darmowe.

# Wykaz rysunków

Rysunek 6.1 Diagram przepływu danych, poziom 0 (źródło: praca własna) 13

Rysunek 6.2 Diagram przepływu danych, poziom 1 (źródło: praca własna) 13

Rysunek 7.1 Prognoza udziałów rynkowych systemów na smartfony (źródło: https://www.idc.com/promo/smartphone-market-share/os - dostęp 2019-11-13) 14

Rysunek 8.1 Widok logowania (źródło: praca własna) 17

Rysunek 8.2 Sprawdzanie poprawności danych logowania (źródło: praca własna) 18

Rysunek 8.3 Widok logowania po podaniu nieprawidłowych danych logowania (źródło: praca własna) 18

Rysunek 8.4 Widok rejestracji nowego użytkownika (źródło: praca własna) 18

Rysunek 8.5 Implementacja modułu odpowiedzialnego za sprawdzanie zapamiętanych danych (źródło: praca własna) 19

Rysunek 8.6 Przykład użycia metody pomocniczej służącej do komunikacji z serwerem (źródło: praca własna) 20

Rysunek 8.7 Miejsce użycia argumentu sendWithAuthHeader (źródło: praca własna) 20

Rysunek 8.8 Przygotowanie nagłówka autoryzacji (źródło: praca własna) 21

Rysunek 8.9 Główny widok prezentujący listę zleceń zamówienia. (źródło: praca własna) 22

Rysunek 8.10 Widok prezentujący złożone zamówienie (źródło: praca własna) 23

Rysunek 8.11 Widok zamawiania posiłku (źródło: praca własna) 24

Rysunek 8.12 Przykład prezentujący moment zaznaczania posiłku (źródło: praca własna) 25

Rysunek 8.13 Metoda build klasy MarkingManager (źródło: praca własna) 26

Rysunek 8.14 Metoda initState klasy MarkingManager (źródło: praca własna) 27

Rysunek 8.15 Metoda getMarked klasy ContentMarker (źródło: praca własna) 28

Rysunek 8.16 Metoda pomocnicza używana w metodzie getMarked() (źródło: praca własna) 28

Rysunek 8.17 Widok dodawania posiłku (źródło: praca własna) 29

Rysunek 8.18 Dolna belka przedstawiająca informacje o stanie aktualnego zamówienia (źródło: praca własna) 29

Rysunek 8.19 Koszyk z zamówieniami (źródło: praca własna) 30

Rysunek 8.20 Widok koszyka z zamówieniami przy przekroczonym limicie cenowym (źródło: praca własna) 31

Rysunek 8.21 Komunikat potwierdzający powodzenie złożenia zamówienia (źródło: praca własna) 32

Rysunek 8.22 Zrzut ekranu prezentujący zmianę statusu zamówienia o nazwie „Warsztat” (źródło: praca własna) 33

Rysunek 8.23 Widok podstrony z tworzeniem nowych zleceń (źródło: praca własna) 34

Rysunek 8.24 Widok podstrony wyświetlającej wszystkie zlecenia (źródło: praca własna) 35

Rysunek 8.25 Deklaracja tabeli wyświetlającej zlecenia zamówień (źródło: praca własna) 36

Rysunek 8.26 Część skryptowa strony display\_order\_requests.html, która pobiera dane z serwera i ładuje je do tabeli (źródło: praca własna) 37

Rysunek 8.27 Zamówienie jednostkowe – użytkownik nr 1 (źródło: praca własna) 37

Rysunek 8.28 Zamówienie jednostkowe – użytkownik nr 2 (źródło: praca własna) 38

Rysunek 8.29 Zamówienie jednostkowe – użytkownik nr 3 (źródło: praca własna) 38

Rysunek 8.30 Zamówienie grupowe wygenerowane z wcześniej przedstawionych zamówień jednostkowych (źródło: praca własna) 39

Rysunek 8.31 Implementacja zasobu służącego do stworzenia nowego użytkownika prezentująca zwracane kody odpowiedzi HTTP (źródło: praca własna) 50

Rysunek 8.32 Implementacja dekoratora exception\_handler zwracającego kod błędu 500 (źródło: praca własna) 50

Rysunek 9.1 Przykładowy e-mail o konieczności złożenia zamówienia (źródło: praca własna) 51

Rysunek 9.2 Schemat bazy danych aplikacji (źródło: praca własna) 52

Rysunek 10.1 Kod podatny na sql injection (źródło: praca własna) 53

Rysunek 10.2 Implementacja bez luki pozwalającej na „wstrzyknięcie” złośliwego kodu SQL (źródło: praca własna) 54

Rysunek 10.3 LoginManager zarządzający użytkownikami i procesem autoryzacji (źródło: praca własna) 55

Rysunek 10.4 Implementacja klasy User (źródło: praca własna) 56

Rysunek 10.5 Przykład użycia dekoratora @login\_required, który powoduje, że zasób wymaga autoryzacji. (źródło: praca własna) 56

Rysunek 11.1 Schemat wzorca BLoC (źródło: https://pub.dev/packages/bloc - dostęp 2019-11-15) 57

Rysunek 11.2 Schemat architektury klient-serwer (źródło: http://architektura-oprogramowania.blogspot.com/p/architektura-klient-serwer.html - dostęp 2019-11-15) 58

# 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. *Flutter Documentation* (online) https://flutter.dev/docs (dostęp: 2019–11–10)
11. *PostgreSQL Tutorial* (online), http://www.postgresqltutorial.com/ (dostęp: 2019–11–13)