Państwowa Wyższa Szkoła Zawodowa w Tarnowie
Katedra Informatyki

Narzędzia i środowiska programistyczne

PIZZERIA HULKSMASH

Projekt 8 | Maksymalna ocena jaką można uzyskać za projekt: **5.0** | Temat:
System do zamawiania posiłków.

Prowadzący: Tomasz Gądek
Kierunek: Informatyka
Semestr: Letni
Rok: 2

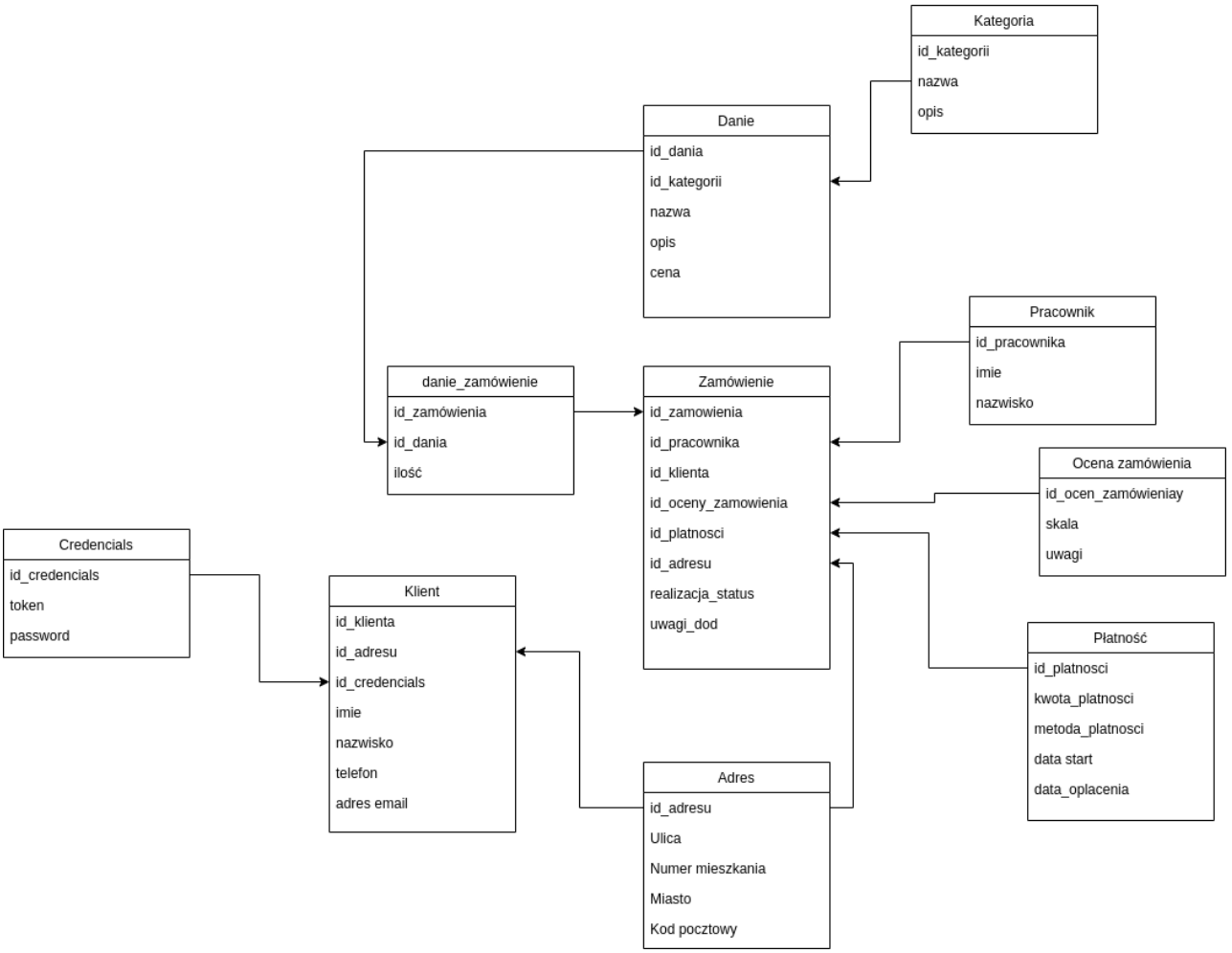
Autorzy:
Dominik Pająk
Bartosz Dymański
Robert Małek
Grzegorz Gancarz

1. Opis projektu

1.1 Restauracja Hulksmash

Restauracja Hulksmash to aplikacja webowa oparta o język JavaScript i framework Vue 3.
Umożliwia ona

- Zakładanie konta
- Logowanie się do konta
- Walidację formularzy
- Dokonywanie zamówień
- Historia zamówień użytkownika
- Aktualnie "ważne zamówienia" (dostępne dla administratorów)
- Możliwość zmiany statusu zamówienia
- Możliwość włączenia / wyłączenia obowiązywania promocji
- Dodawanie nowych produktów
- Dodawanie nowych kategorii
- Symulacja płatności internetowej
- Generowanie faktury w formacie PDF po dokonaniu zamówienia i automatyczne wysyłanie jej na email użytkownika



Screen 0: Struktura bazy danych

1.2 Serwer - endpointy opis

Serwer aplikacji Restauracja Hulksmash zawiera 54 endpointy. Dokładną dokumentację, można podglądnąć lokalnie na Swaggerze dostępnym pod linkiem http://localhost:8001/swagger-ui.html po odpaleniu serwera W tej dokumentacji opisane zostaną tylko te endpointy wykorzystywane przez aplikację webową (pozostała część jest zaimplementowana z myślą o aplikacji mobilnej i ewentualne rozwinięcie projektu [o np. kasowanie produktów])

GET /api/restaurant/category
BODY: {
  "desc": "string",
  "id": integer,
  "name": "string"
}
DESCRIPTION: Zwraca listę wszystkich kategorii

POST /api/restaurant/category
BODY: {
  "desc": "string",
  "id": integer,
  "name": "string"
}
DESCRIPTION: Tworzy nową kategorię i zwraca ją

POST /api/restaurant/order
BODY: {
  "client": "ClientObject",
  "comments": “string”,
  "paymentData": "PaymentObject"
  "rate": "RateObject"
  "realizationStatus": "string"
  "worker": "WorkerObject"
}
DESCRIPTION: Tworzy i zwraca obiekt nowego zamówienia

POST /api/restaurant/order-data/all
BODY: {
  "client": "ClientObject",
  "comments": “string”,
  "paymentData": "PaymentObject"
  "rate": "RateObject"
  "realizationStatus": "string"
  "worker": "WorkerObject"
}
DESCRIPTION: Tworzy i zwraca obiekt nowego zamówienia

POST /api/restaurant/order-rate
BODY: {
  "comments": "string",
  "id": integer,
  "points": integer
}
DESCRIPTION: Tworzy i zwraca pusty obiekt oceny zamówienia. Aplikacja nie obsługuje możliwości jej aktualizacji

---

POST /api/restaurant/payment'
BODY:{
  "amount": integer,
  "id": integer,
  "method": "string",
  "paymentData": "string",
  "startData": "string"
}
DESCRIPTION: Rozpoczyna i zwraca nową płatność, automatycznie ustawiając status na ROZPOCZĘTA. Jest to potem aktualizowane przez ten sam endpoint

---

POST /api/restaurant/food'
BODY: {
  "categoryDto": {
    "id": integer,
  },
  "description": "string",
  "id": integer,
  "imageHref": "string",
  "name": "string",
  "prize": integer
}
DESCRIPTION: Tworzy nowy produkt

GET /api/restaurant/food'
BODY: {
  "categoryDto": {
    "desc": "string",
    "id": integer,
    "name": "string"
  },
  "description": "string",
  "id": integer,
  "imageHref": "string",
  "name": "string",
  "prize": integer
}
DESCRIPTION: Zwraca listę wszystkich produktów

GET /api/restaurant//foodByCategory?categoryId={id}
BODY: {
  "categoryDto": {
    "desc": "string",
    "id": integer,
    "name": "string"
  },
  "description": "string",
  "id": integer,
  "imageHref": "string",
  "name": "string",
  "prize": integer
}
DESCRIPTION: Zwraca listę wszystkich produktów danej kategorii

---

GET /api/restaurant/promotion?id={id}
BODY: {
  "empty": true,
  "present": true
}
DESCRIPTION: Zwraca obiekt promocji

GET /api/restaurant/promotionStatus?id={id}
BODY: {
  "active": true,
  "id": integer
}
DESCRIPTION: Zmienia status promocji

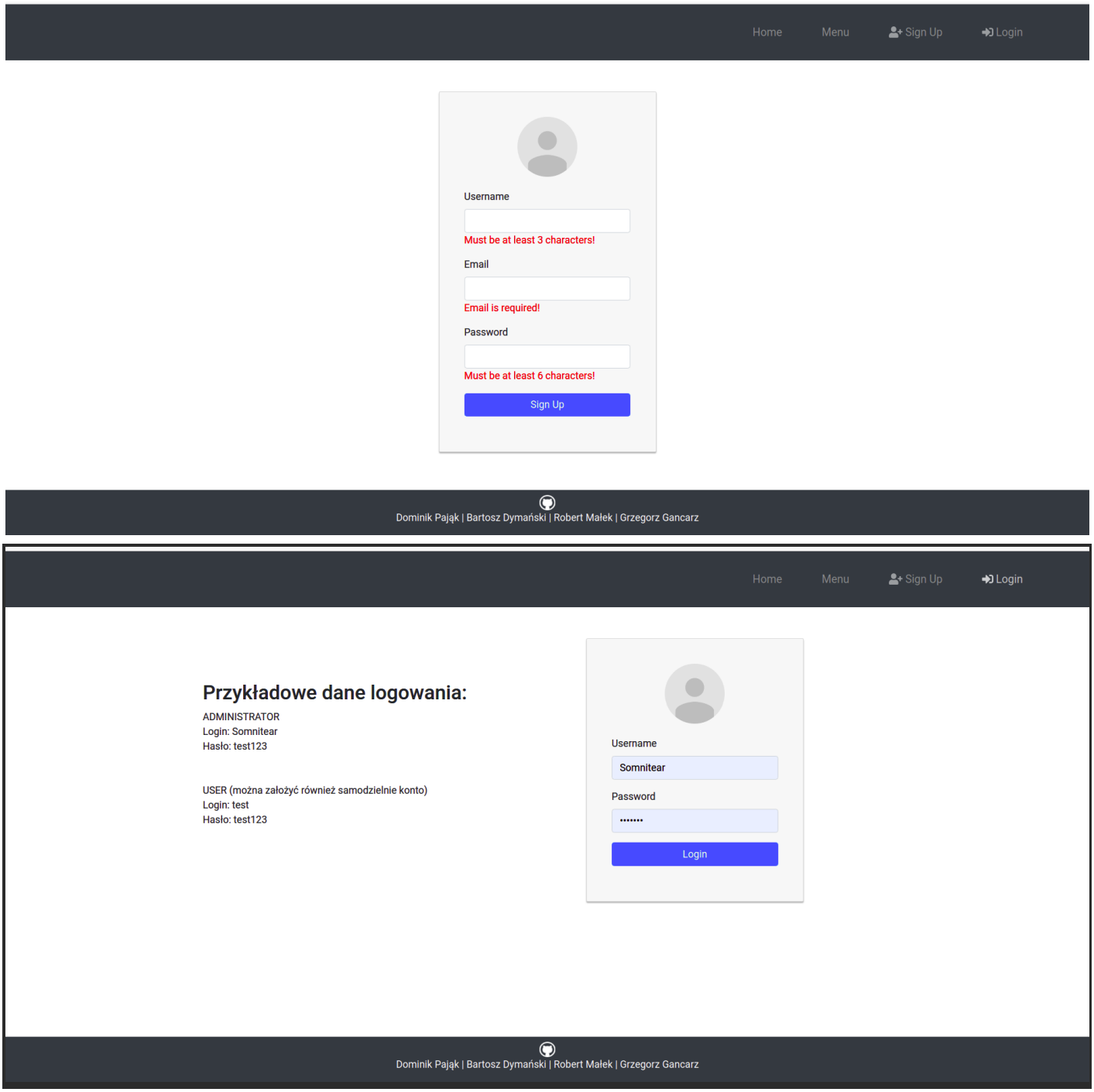
GET /api/restaurant/orders/actual
BODY: [{
  "client": {
    "address": {
      "city": "string",
      "homeNumber": "string",
      "id": integer,
      "postNumber": "string",
      "street": "string"
    }, "email": "string",
    "id": integer,
    "name": "string",
    "phone": "string",
    "surname": "string",
    "username": "string"
  }, "comments": "string",
  "id": integer,
  "paymentData": {
    "amount": integer,
    "id": integer,
    "method": "string",
    "paymentData": "string",
    "startData": "string"
  }, "rate": {
    "comments": "string",
    "id": integer,
    "points": integer
  }, "realizationStatus": "string",
  "worker": {
    "id": integer,
    "name": "string",
    "surname": "string"
  }}]

DESCRIPTION: Zwraca listę niezakończonych zamówień (aktualnych)

GET /api/restaurant/orders/user?id={id}
BODY: [{
"client": {
"address": {
"city": "string",
"homeNumber": "string",
"id": integer,
"postNumber": "string",
"street": "string"
},
"email": "string",
"id": integer,
"name": "string",
"phone": "string",
"surname": "string",
"username": "string"
},
"comments": "string",
"id": integer,
"paymentData": {
"amount": integer,
"id": integer,
"method": "string",
"paymentData": "string",
"startData": "string"
},
"rate": {
"comments": "string",
"id": integer,
"points": integer
},
"realizationStatus": "string",
"worker": {
"id": integer,
"name": "string",
"surname": "string"
}
}]
DESCRIPTION: Zwraca historię wszystkich zamówień użytkownika

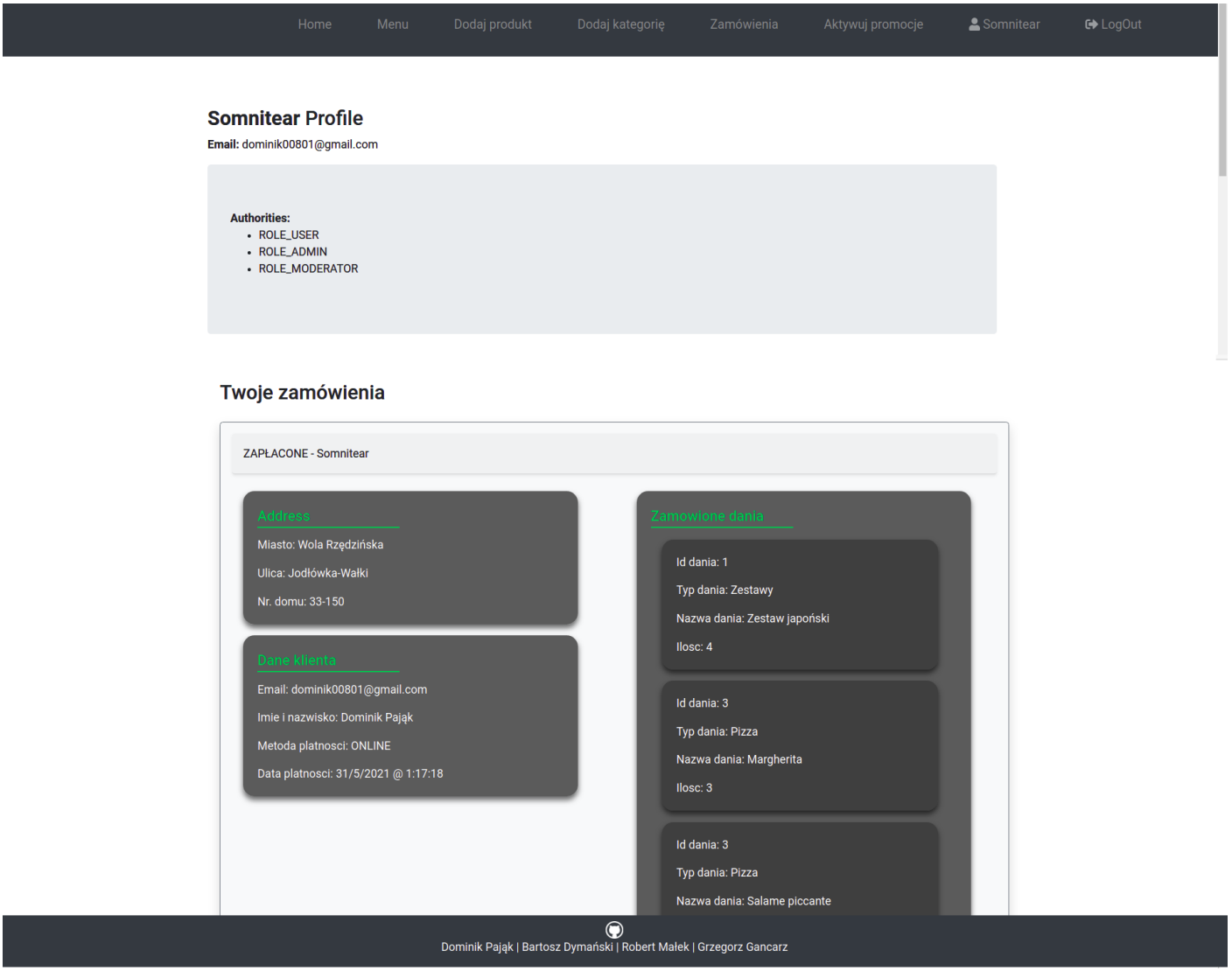
1.3 Klient - opis funkcjonalności

Użytkownik chcący skorzystać z usług restauracji, musi założyć konto w serwisie. Aplikacja obejmuje dwa typy kont - użytkownika - administratora. Autoryzacja w serwerze odbywa się na podstawie tokena bearer. Hasła w bazie są szyfrowane podstawowym UUID



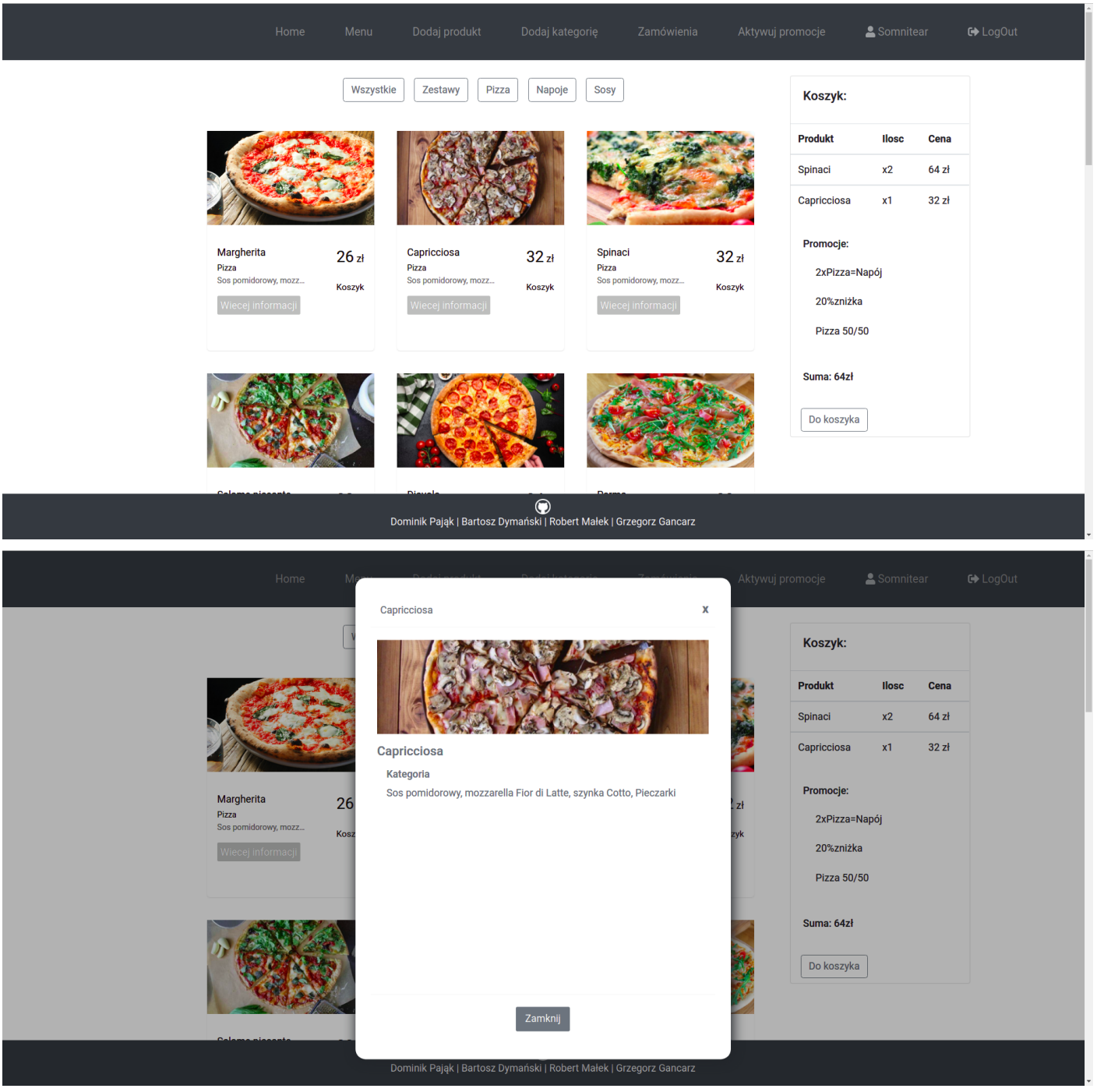
Screen 1: Przedstawienie panelu logowania / rejestracji

Po udanym logowaniu wita nas ekran z podsumowaniem profilu



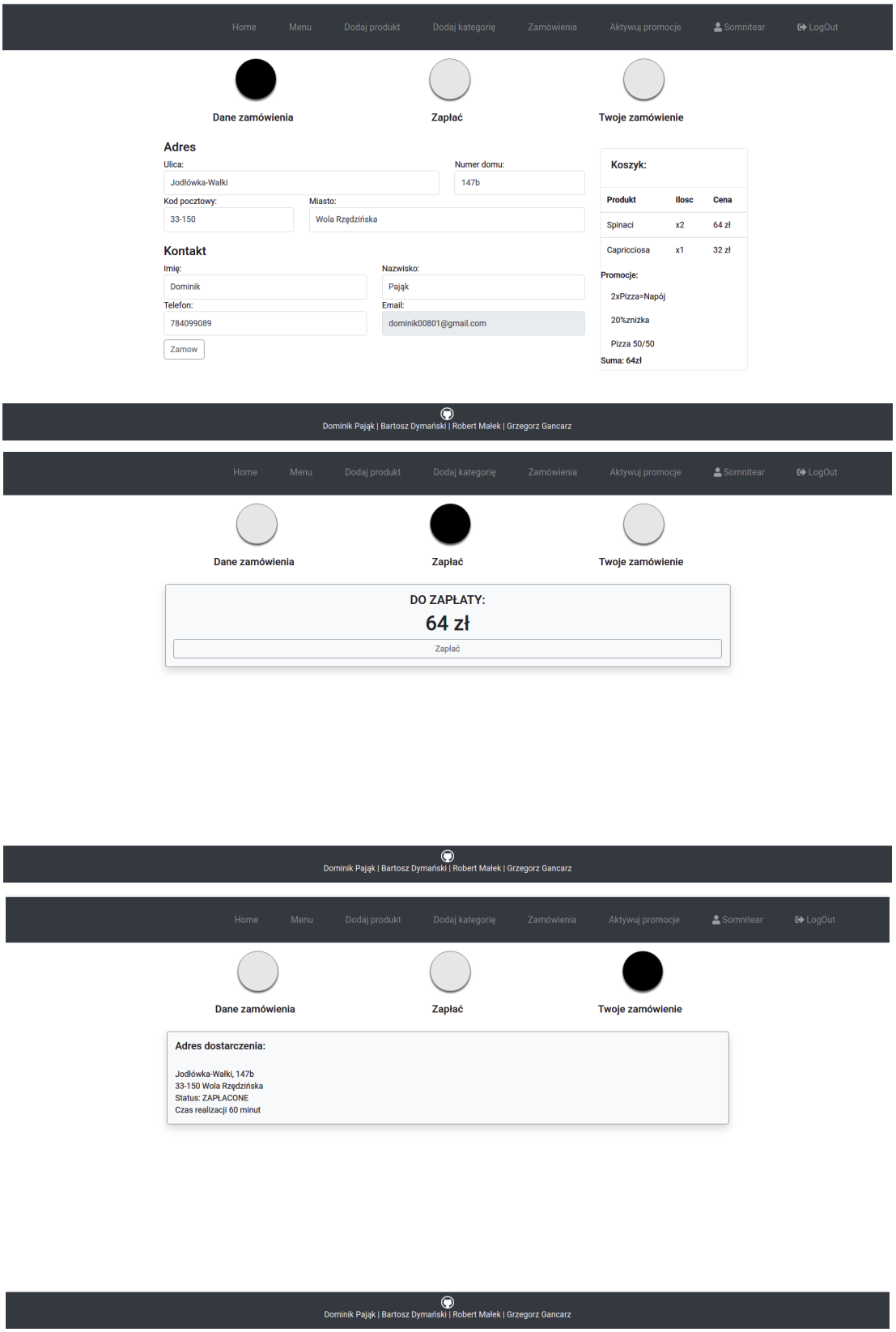
Screen 2: Ekran powitalny po zalogowaniu

Kolejną ścieżką dla zwykłego użytkownika jest panel z menu restauracji. Klikając w przycisk “Koszyk” dodajemy produkt do listy zakupów, którą możemy sfinalizować, klikając w przycisk “Do koszyka”



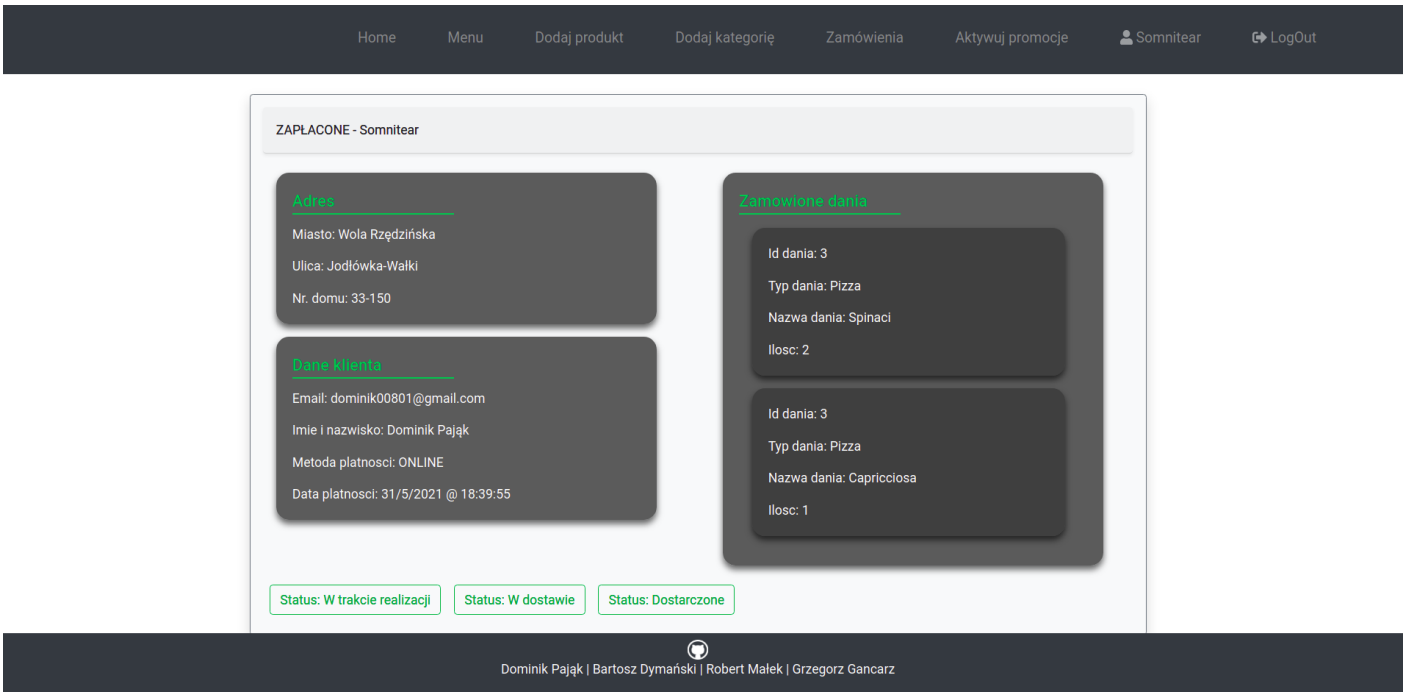
Screen 3: Widok zakupu produktów

Po zakończeniu wyboru produktów, przechodzimy do ścieżki płatności.



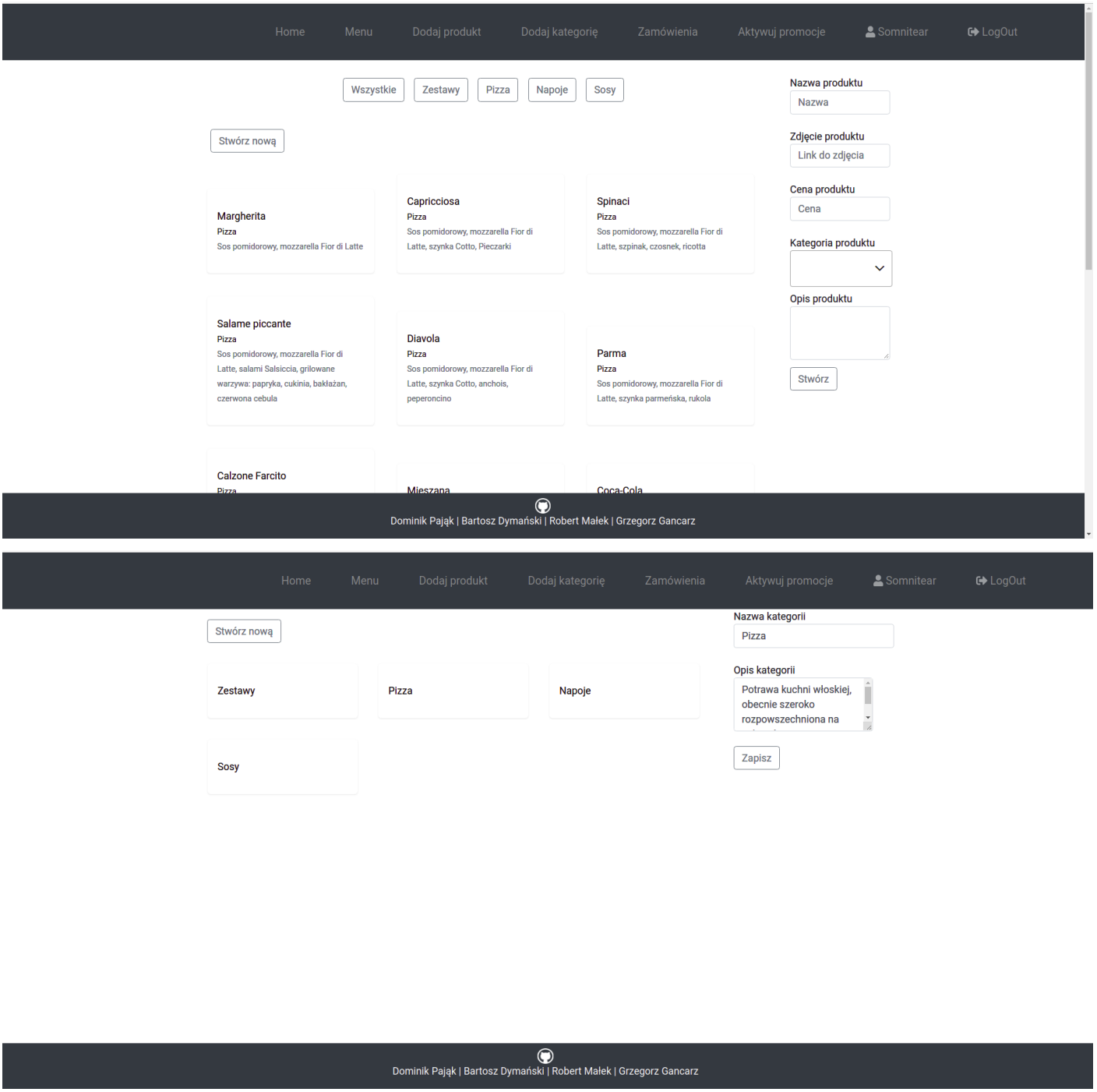
Screen 4: Ścieżka płatności za zamówienie

Po ukończeniu ścieżki płatności, zamówienie jest dodawane do listy oczekujących zamówień. Jako administratorzy, mamy dostęp do wglądu na wszystkie zamówienia, które nie mają statusu “DOSTARCZONE”. Status zamówienia zmieniamy, klikając w odpowiedni przycisk



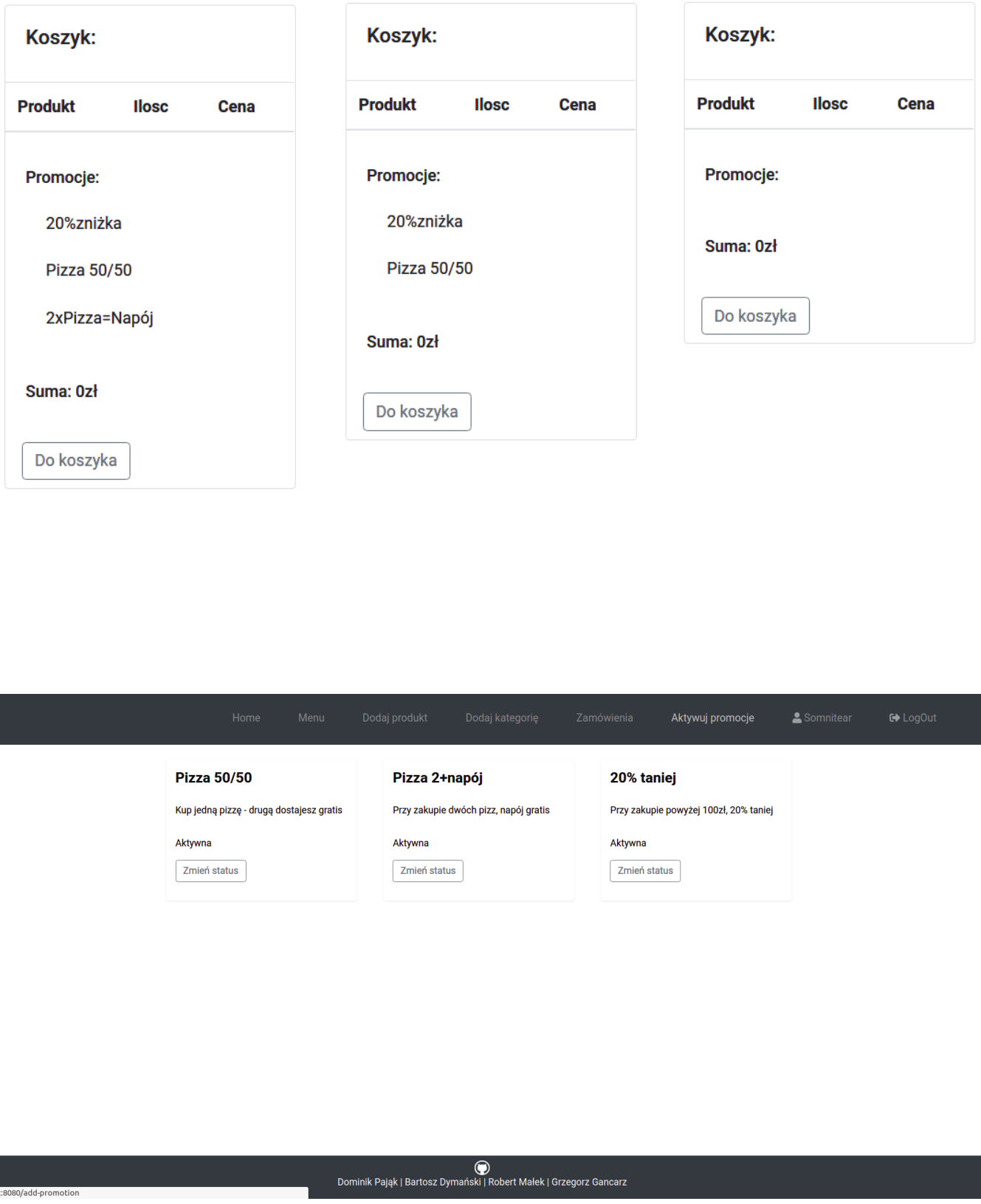
Screen 5: Widok aktualnych zamówień

Inną funkcjonalnością tylko dla administratorów, jest możliwość dodawania / edytowania nowych produktów i kategorii



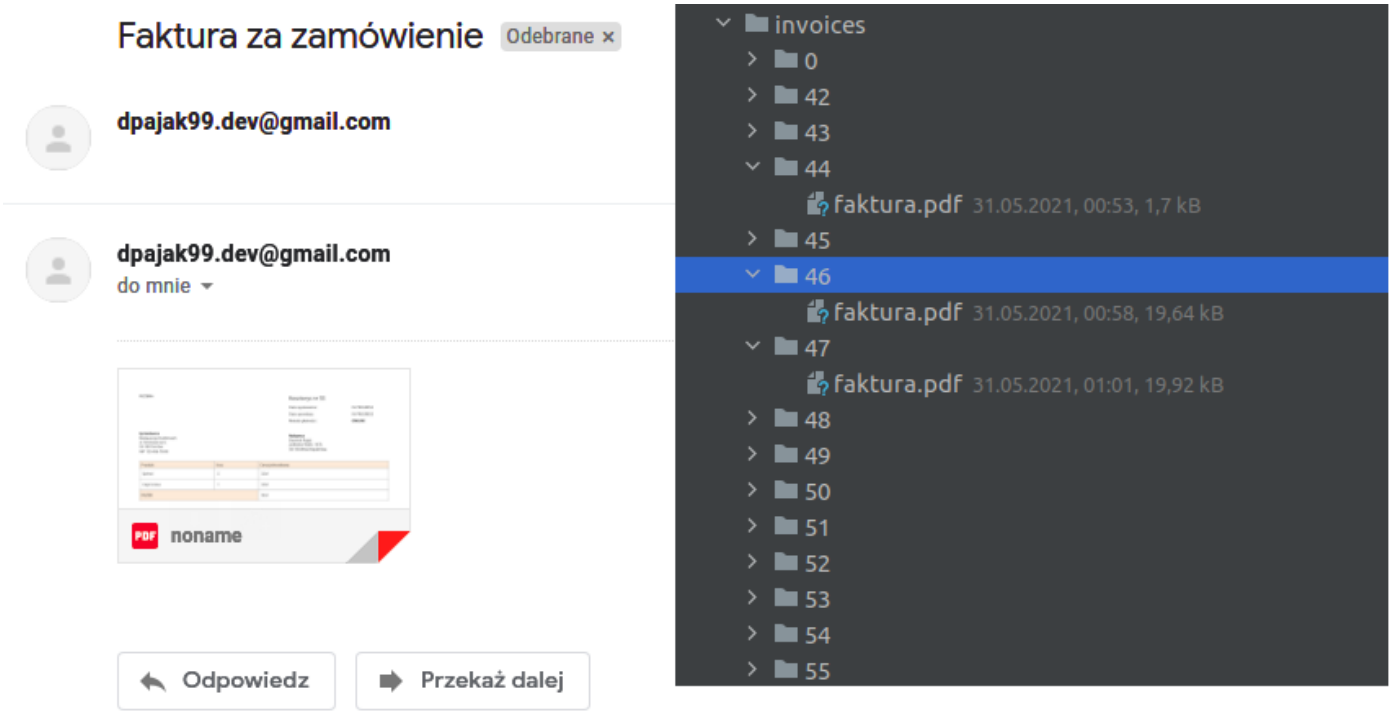
Screen 6: Panel edycji produktów / kategorii

Panel posiada również możliwość włączania / wyłączania poszczególnych promocji



Screen 7: Edycja promocji

Ostatnią ważną funkcjonalnością wspieraną przez Restaurację HulkSmash jest generowanie faktur / paragonów za dokonane zamówienie. Po opłaceniu zamówienia, dostajemy na maila podanego przy rejestracji wiadomość z fakturą. Dodatkowo, kopia faktury jest przechowywana również lokalnie na serwerze



PIZZERIA

Kosztorys nr 51

Data wystawienia: IN PROGRESS
Data sprzedaży: IN PROGRESS
Metoda płatności: **ONLINE**

Sprzedawca
Restauracja HulkSmash
ul. Mickiewicza 8,
33-100 Tarnów
NIP 123-456-78-89

Nabywca
Dominik Pająk
Jodłówka-Wałki, 147b
33-150-Wola Rzędzińska

Produkt	Ilosc	Cena jednostkowa
Zestaw japoński	4	90zł
Margherita	3	26zł
Salame piccante	1	32zł
Diavola	1	34zł
Coca-Cola	1	9zł
BBQ	1	2zł
Ice-tea	1	8zł
RAZEM		523zł

Screen 8: Faktura / paragon za wykonane zamówienie

1.4 Podział obowiązków w zespole

Dominik Pająk	Bartosz Dymański	Robert Małek	Grzegorz Gancarz
Implementacja systemu logowania / rejestracji na podstawie kodu stworzonego przez https://bezkoder.com/	Wstępny setup aplikacji klienckiej jak i serwerowej	Zaplanowanie, rozpisanie i stworzenie bazy danych w Postgresie (**Screen 0)**	Stworzenie layoutu faktury wysłanej do użytkownika.
Zaimplementowanie Swaggera do dokumentacji backendu	Stworzenie bazowego CRUD serwera dla wszystkich obiektów *(git push -f (użyty w kryzysowej sytuacji) przypadkowo skasował tą część historii commitów)*	Zaimplementowanie ścieżki dokonywania płatności za zamówienie	Główne stylowanie aplikacji webowej
Dopracowywanie części serwerowej do bieżących potrzeb	Implementacja Vuelidate do sprawdzania poprawności formularzy	Ręczne wypełnianie bazy danych danymi przed stworzeniem aplikacji CRUD	
Dokumentacja - opis serwera	Logika aplikacji - frontend		
Implementacja połączenia aplikacji klienckiej z serwerem	Stworzenie ścieżki nawigowania po aplikacji	-	Dokumentacja - instrukcja użytkowania

Bieżąca dokumentacja pracy na podstawie commitów:
Klient : https://bitbucket.org/Menji/project_01_hulksmash_client/commits/
Serwer: https://bitbucket.org/Menji/project_01_hulksmash_server/commits/

1.5 Użyte biblioteki - Źródła

- Serwer
    - springfox-swagger2
    - springfox-swagger-ui
    - spring-boot-starter-data-jpa
    - spring-boot-starter-web
    - pring-boot-devtools
    - postgresql
    - spring-boot-starter-test
    - jjwt
    - jakarta.validation-api
    - javax.mail
    - flying-saucer-core
    - flying-saucer-pdf-openpdf
    - jsoup
- Klient
    - fortawesome/fontawesome-svg-core
    - fortawesome/free-solid-svg-icons
    - fortawesome/vue-fontawesome
    - vuelidate/core
    - vuelidate/validators
    - axios
    - bootstrap
    - core-js
    - jquery
    - popper.js
    - primeicons
    - primevue
    - vee-validate
    - vue
    - vue-router
    - vuelidate
    - vuex
    - yup
- Inne
    - https://bezkoder.com/ - System logowania / rejestracji Vue3

- Wykorzystane narzędzia
    - **Bitbucket** jako system kontroli wersji
    - **InteliJJ IDEA / Webstorm / Visual Studio** jako IDE
    - **DBBreaver, PgAdmin -** ręczne połączenie z bazą danych
    - **Postman** do sprawdzania poprawności backendu przed zaimplementowaniem klienckiego CRUDa
    - **Draw.io** - projektowanie struktury bazy danych
    - **Figma** - do prostego wireframingu aplikacji