

Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

**Bartłomiej Kotarski
Marcin Krocza**

CAR CLEAN

Projekt zespołowy
na studiach stacjonarnych
o kierunku Informatyka

Kielce, 2020

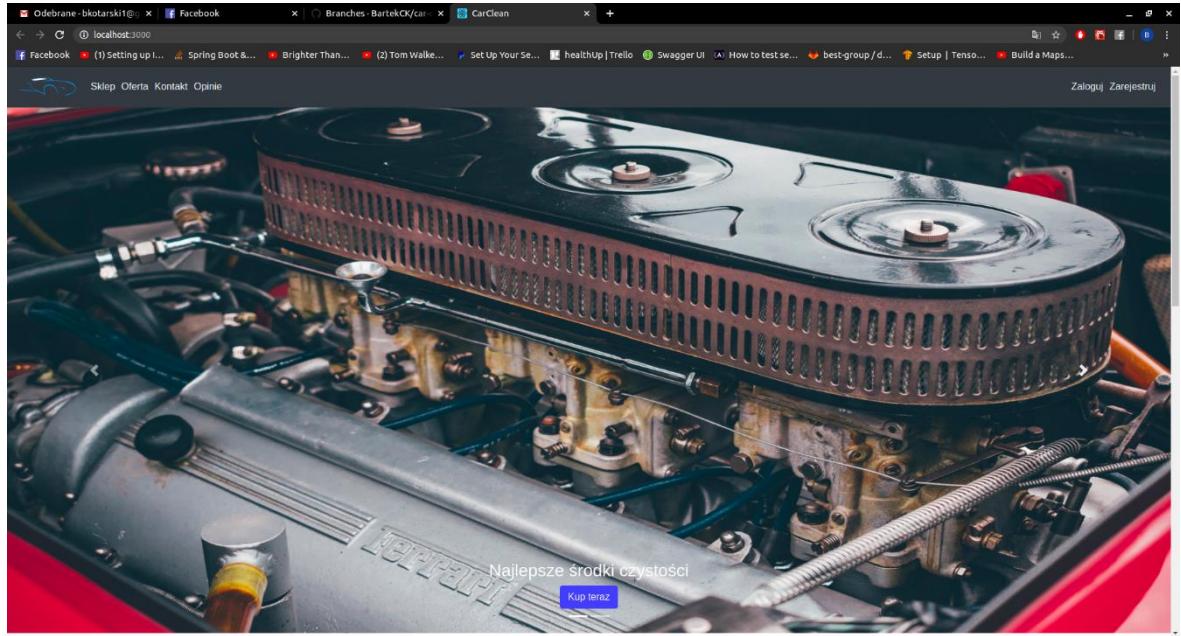
SPIS TREŚCI

1. Instrukcja obsługi.....	3
1.1. Rejestracja	3
1.2. Logowanie	5
1.3. Dodawanie pojazdu	5
1.4. Usuwanie pojazdu	7
1.5. Składanie zamówienia na serwis.....	7
1.6. Przeglądanie statusu serwisu.....	9
1.7. Płatność za serwis przy pomocy PayPal	10
1.8. Wysyłanie wiadomości	12
1.9. Dodawanie opinii	13
1.10.Zamiana statusu zamówienia przez pracownika.....	15
1.11.Błąd 404.....	17
1.12.Składanie zamówienia na produkty	18
1.13.Operacje wykonywane na koszyku zakupowym	19
1.14.Przeglądanie zamówień produktów	20
1.15.Płatność za produkty przy pomocy PayPal.....	21
1.16.Płatność za produkty przy odbiorze.....	23
2. Omówienie wykorzystanych warstw	24
2.1. Dodanie serwisu przez użytkownika	24
2.2. Dodanie opinii przez użytkownika.....	33
2.3. Logowanie zapisanego użytkownika w przeglądarce	37
2.4. Zmiana statusu usługi przez pracownika	40
2.5. Dodawanie produktów do koszyka przez zalogowanego użytkownika.....	45
3. Wnioski	49
Wykaz rysunków	50

1. INSTRUKCJA OBSŁUGI

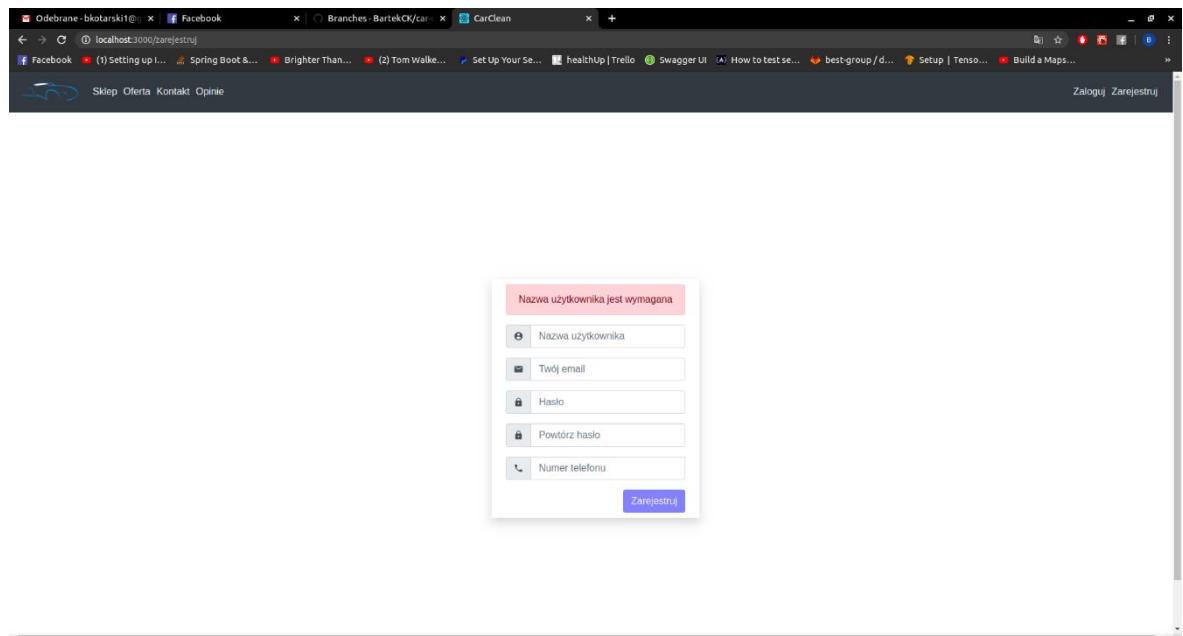
Pełną prezentację aplikacji można zobaczyć na stronie
https://www.youtube.com/watch?v=tIVzi2_GCf4&t

1.1. Rejestracja



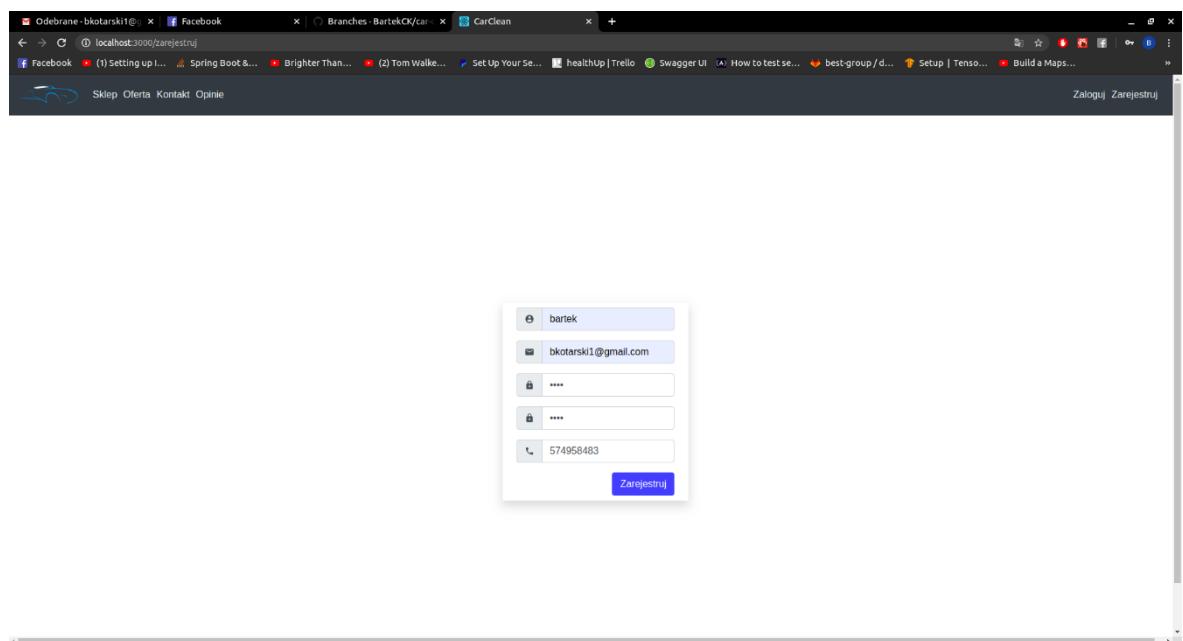
Rysunek 1.1.1 Strona główna aplikacji.

Aby założyć konto należy przejść na stronę główną aplikacji i wcisnąć przycisk zarejestruj.



Rysunek 1.1.2 Panel rejestracji.

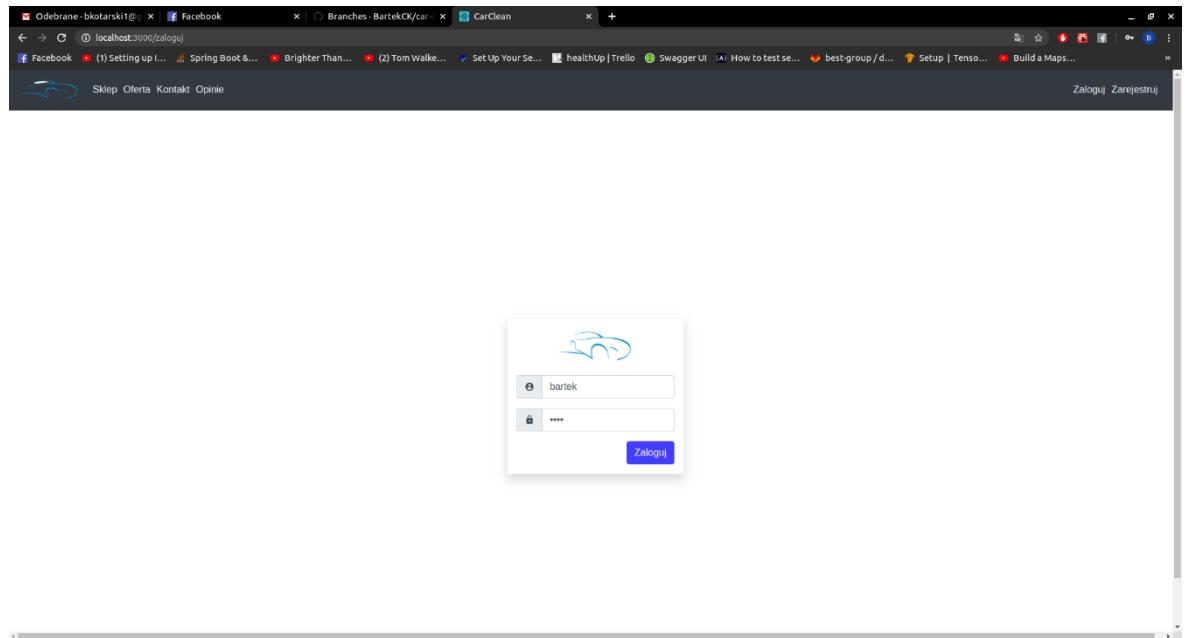
Następnie należy wprowadzić wymagane dane.



Rysunek 1.1.3 Wypełniony panel rejestracji.

Po prawidłowym wprowadzeniu danych należy kliknąć przycisk zarejestruj.

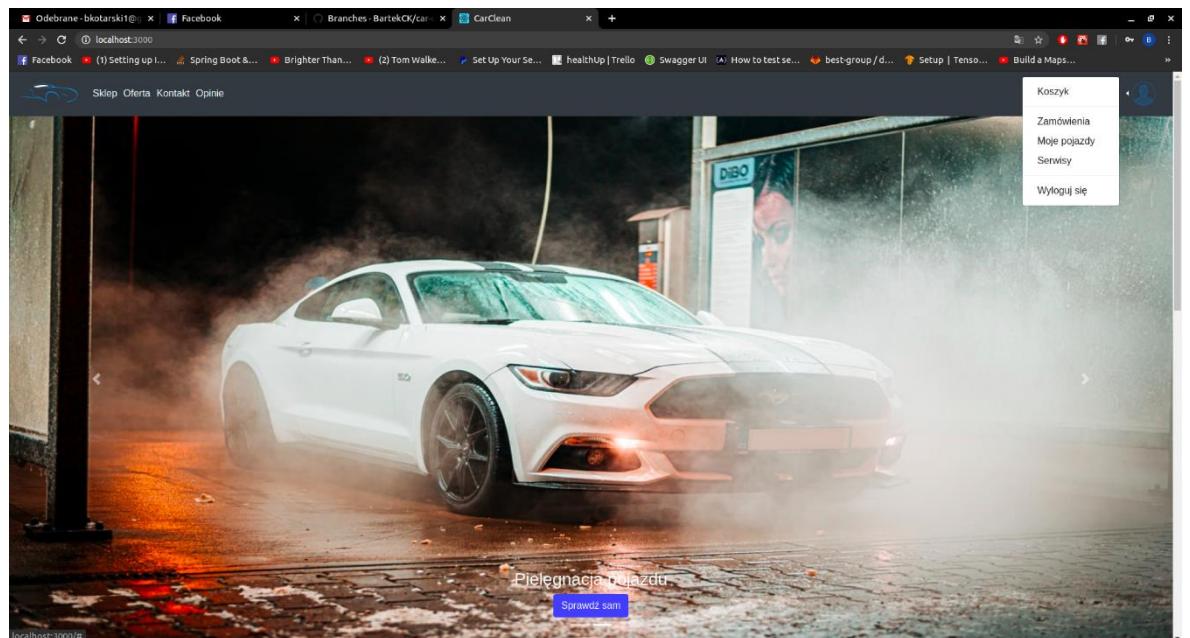
1.2. Logowanie



Rysunek 1.2.1 Panel logowania użytkownika.

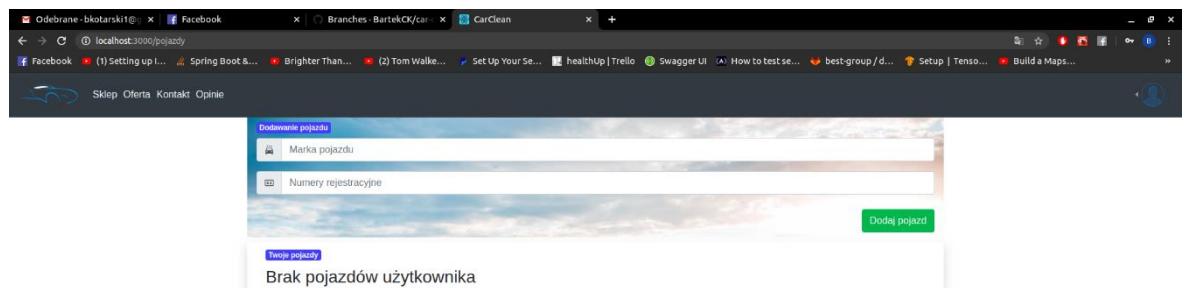
Należy przejść do strony logowania, wprowadzić swoje dane osobowe i wcisnąć przycisk zaloguj.

1.3. Dodawanie pojazdu



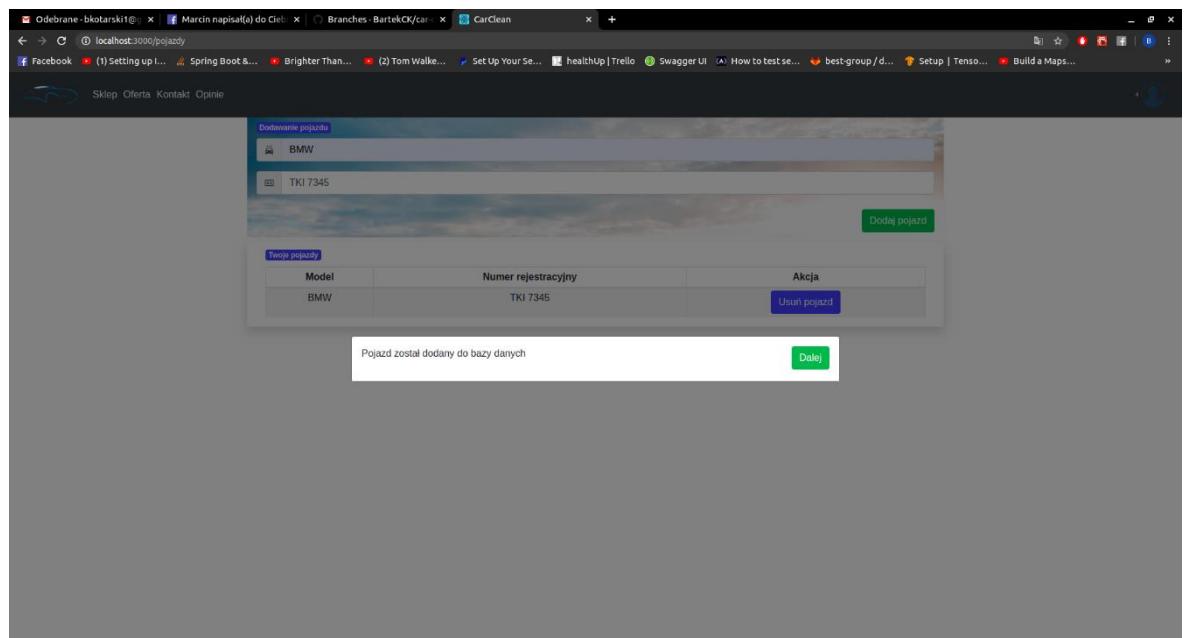
Rysunek 1.3.1 Strona główna wraz z panelem użytkownika.

Należy zalogować się, a następnie kliknąć ikonę użytkownika. Wyświetli się menu podręczne, z którego należy wybrać opcje „Moje pojazdy”.



Rysunek 1.3.2 Strona pojazdów użytkownika.

Następnie należy wprowadzić dane dotyczące marki pojazdu oraz jego numerów rejestracyjnych. Jeżeli wszystko przebiegnie poprawnie, użytkownik zostanie o tym poinformowanym stosownym komunikatem.

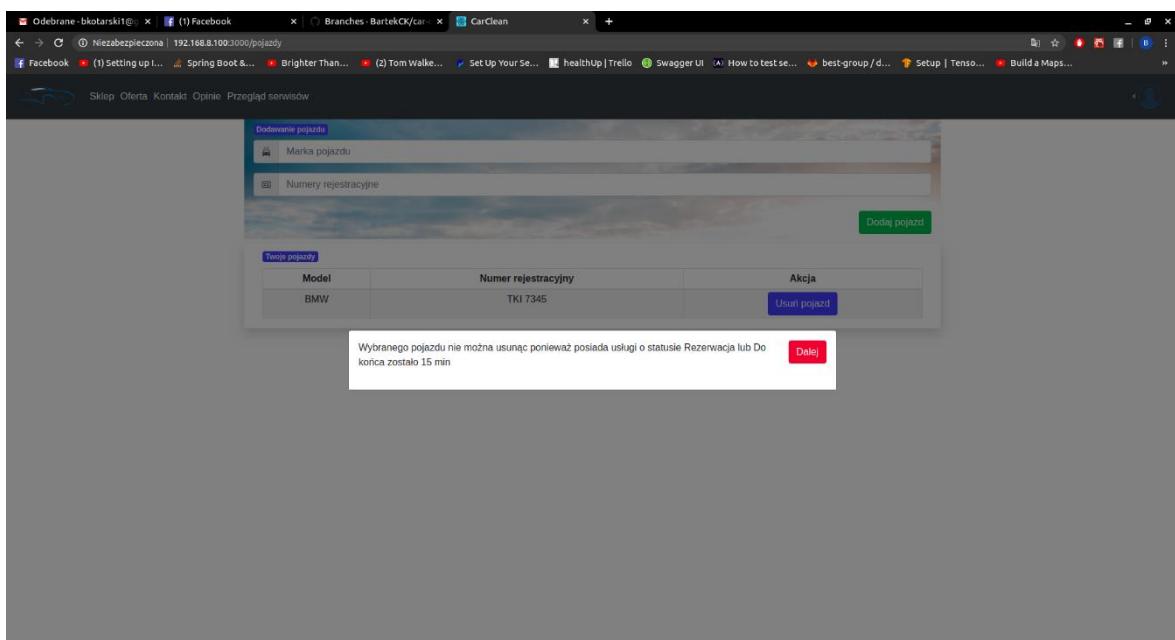


Rysunek 1.3.3 Informacja potwierdzająca dodanie pojazdu.

1.4. Usuwanie pojazdu

Usuwanie pojazdu może nastąpić jedynie wtedy, kiedy dany pojazd nie posiada rezerwacji na wybraną przez użytkownika usługę pielęgnacyjną. Jeżeli ten warunek jest spełniony wystarczy wcisnąć przycisk usuń pojazd.

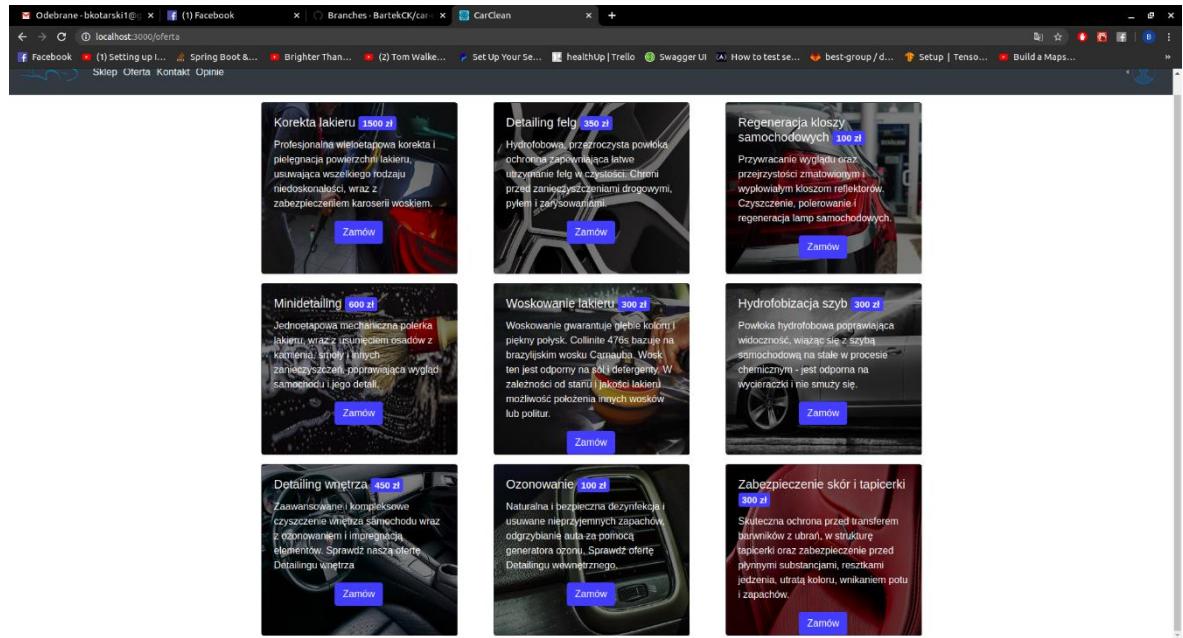
W przeciwnym wypadku, użytkownik zostanie poinformowany stosownym komunikatem.



Rysunek 1.4.1 Komunikat o błędzie podczas usuwania pojazdu.

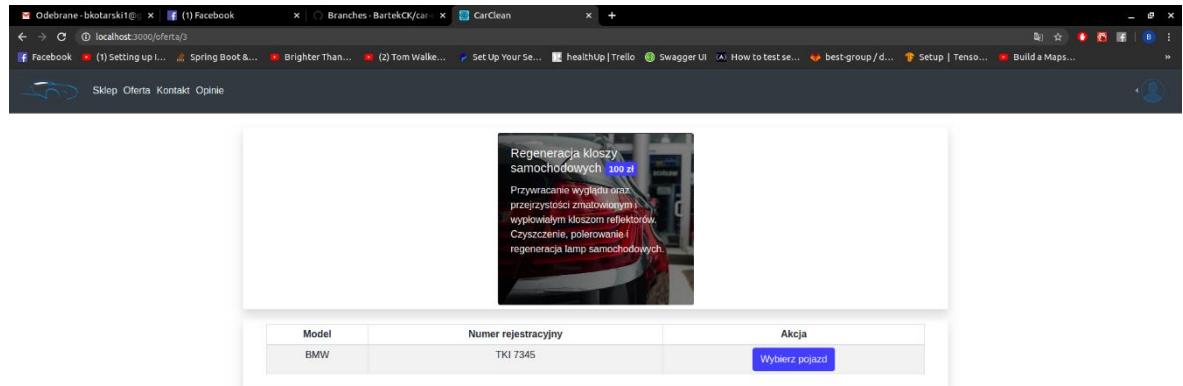
1.5. Składanie zamówienia na serwis

Złożenie zamówienia jest dostępne jedynie dla zalogowanych użytkowników, którzy posiadają dodane pojazdy. Należy przejść do zakładki „oferta” i wybrać odpowiednią usługę.



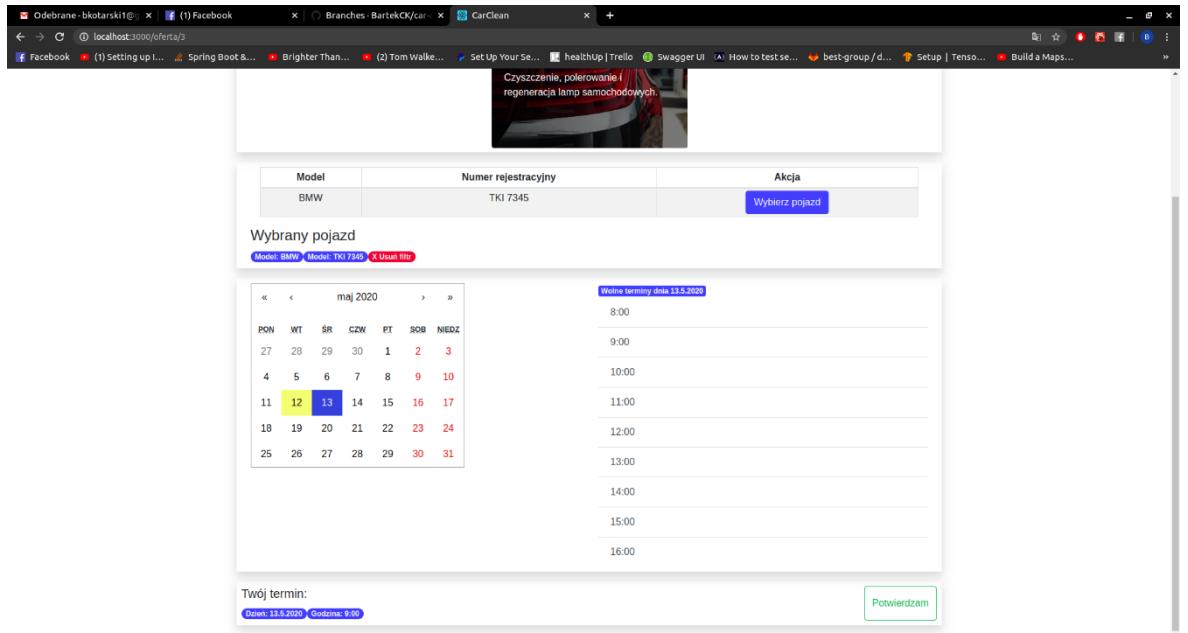
Rysunek 1.5.1 Strona z wyróżnieniem dostępnych usług na stronie.

Następnie pojawi się tabela, w której należy wybrać pojazd, na którym to usługa będzie wykonywana



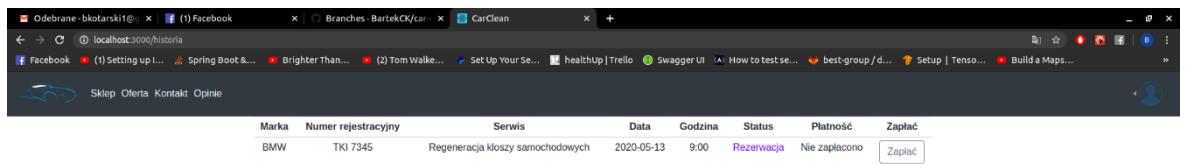
Rysunek 1.5.2 Rejestracja na usługę przed wyborem pojazdu

Ostatnim krokiem jest wybór daty i godziny odpowiadającej użytkownikowi.



Rysunek 1.5.3 Rejestracja na usługę po wyborze pojazdu, daty i godziny.

1.6. Przeglądanie statusu serwisu

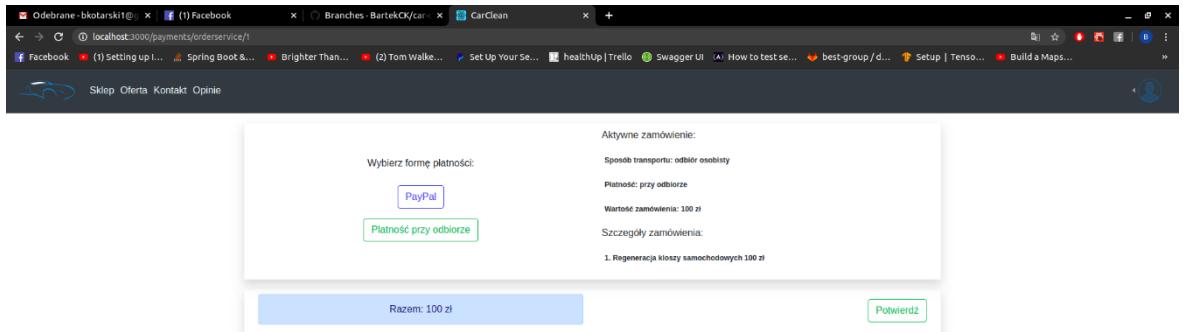


Rysunek 1.6.1 Historia serwisów danego użytkownika.

Opcja dostępna tylko dla zalogowanych użytkowników. Z menu podręcznego należy wybrać Serwisy. Następnie następuje przekierowanie na stronę wyświetlającą historię serwisów danego użytkownika.

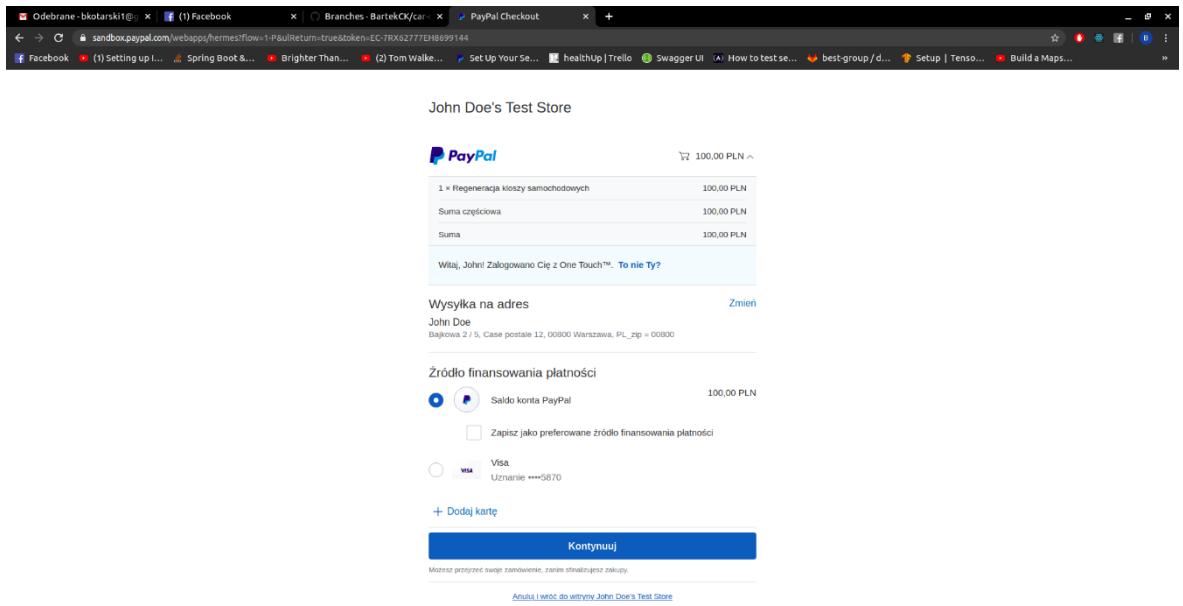
1.7. Płatność za serwis przy pomocy PayPal

Zalogowany użytkownik może za pomocą karty Serwisy, która jest dostępna w menu podręcznym użytkownika, zapłacić za usługę przy pomocy platformy płatniczej PayPal. Należy przy konkretnej usłudze wcisnąć przycisk zapłać, a potem nastąpi przekierowanie do strony potwierdzającej zamówienie.



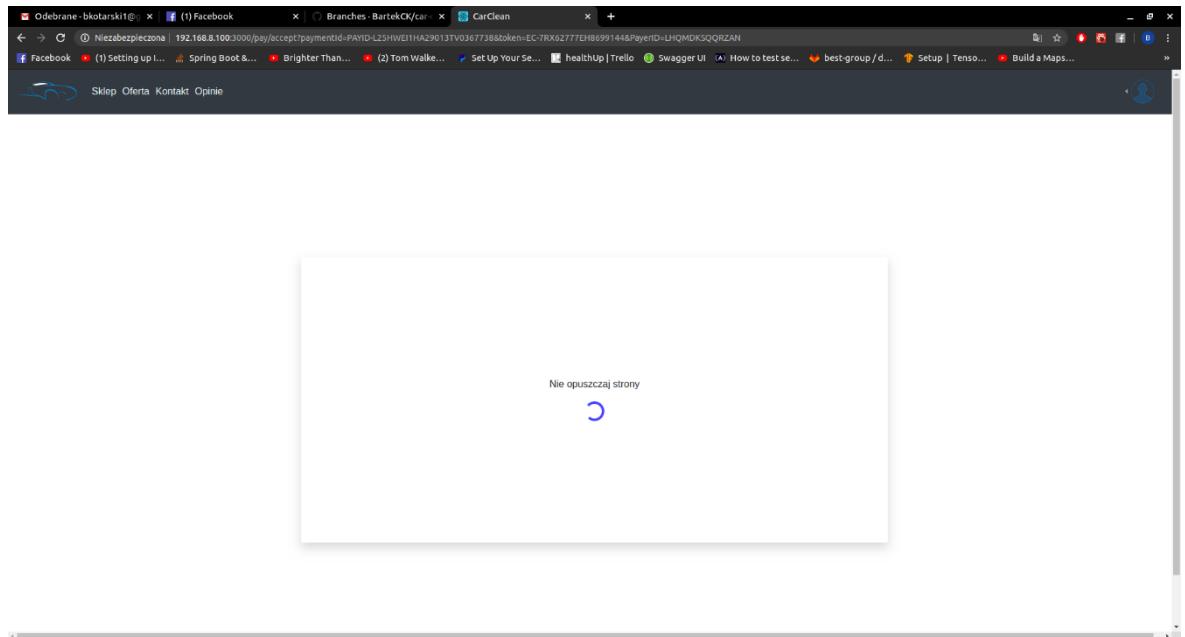
Rysunek 1.7.1 Strona pośrednicząca pomiędzy płatnością.

Jeżeli użytkownik potwierdzi oraz wybierze formę płatności jako „PayPal” zostanie przekierowany na platformę płatniczą wraz z informacjami o danym zamówieniu.



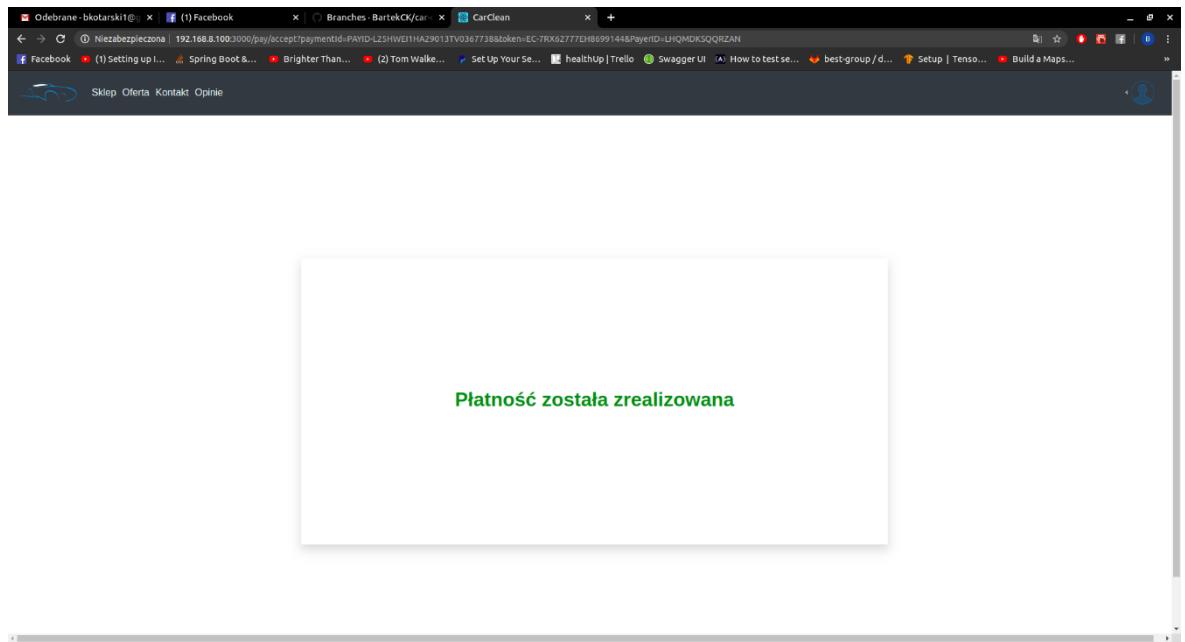
Rysunek 1.7.2 Rachunek w PayPal.

Po akceptacji następuje przekierowanie na stronę potwierdzającą.



Rysunek 1.7.3 Oczekiwanie na akceptację systemu.

O pozytywnym lub negatywnym ukończeniu akcji, użytkownik zostanie poinformowany stosownym komunikatem.



Rysunek 1.7.4 Akceptacja płatności.

W zakładce Serwisy widnieje informacja o zapłaconej usłudze.

Marka	Numer rejestracyjny	Serwis	Data	Godzina	Status	Płatność	Zapłać
BMW	TKI 7345	Regeneracja kloszy samochodowych	2020-05-13	9:00	Rezerwacja	Zapłacono	

Rysunek 1.7.5 Historia serwisów wraz z zapłaconą usługą.

1.8. Wysyłanie wiadomości

Gdzie się mieścimy
25-656 Kielce
Ul. Czysta 23
Polska

Telefon
Stacjonarny: +41 45 36 234
Komórkowy: 567 345 234

E-mail
bkotarski1@gmail.com
marcink1303@gmail.com

Facebook
Instagram

Masz pytania? Napisz do nas

Imię
Bartłomiej

Adres E-mail
bkotarski1@gmail.com

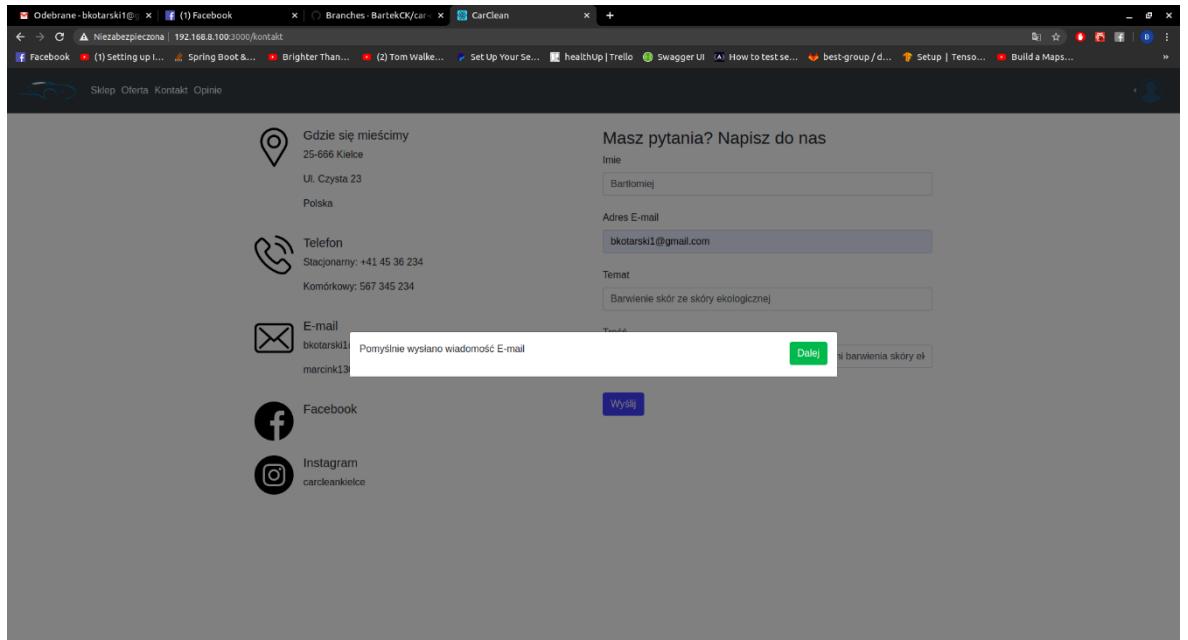
Temat
Barwienie skór ze skóry ekologicznej

Treść
Dzien dobry, czy zakład zajmuje się również usługami barwienia skóry el

Wyslij

Rysunek 1.8.1 Wysyłanie wiadomości do administracji.

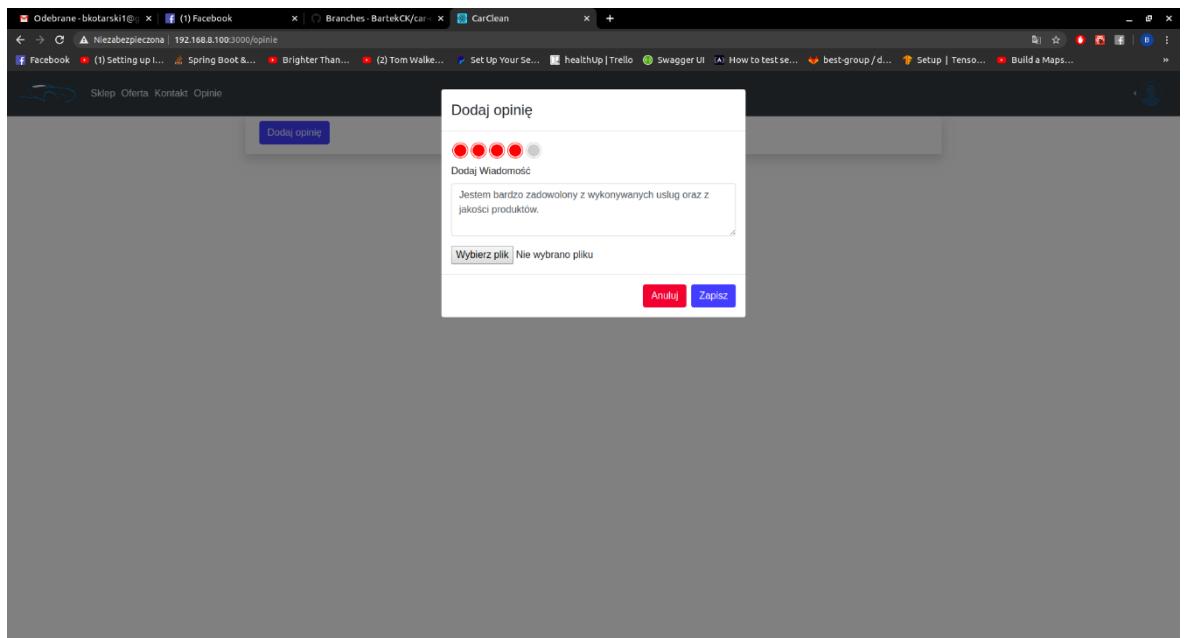
Każdy odwiedzający aplikacje, ma możliwość wysłania pytania do osoby odpowiedzialnej za prawidłowe działanie działalności gospodarczej. W tym celu osoba ta musi przejść do zakładki „Kontakt”, wpisać odpowiednie dane, a następnie zaakceptować przyciskiem „wyślij”.



Rysunek 1.8.2 Potwierdzenie wysłania wiadomości.

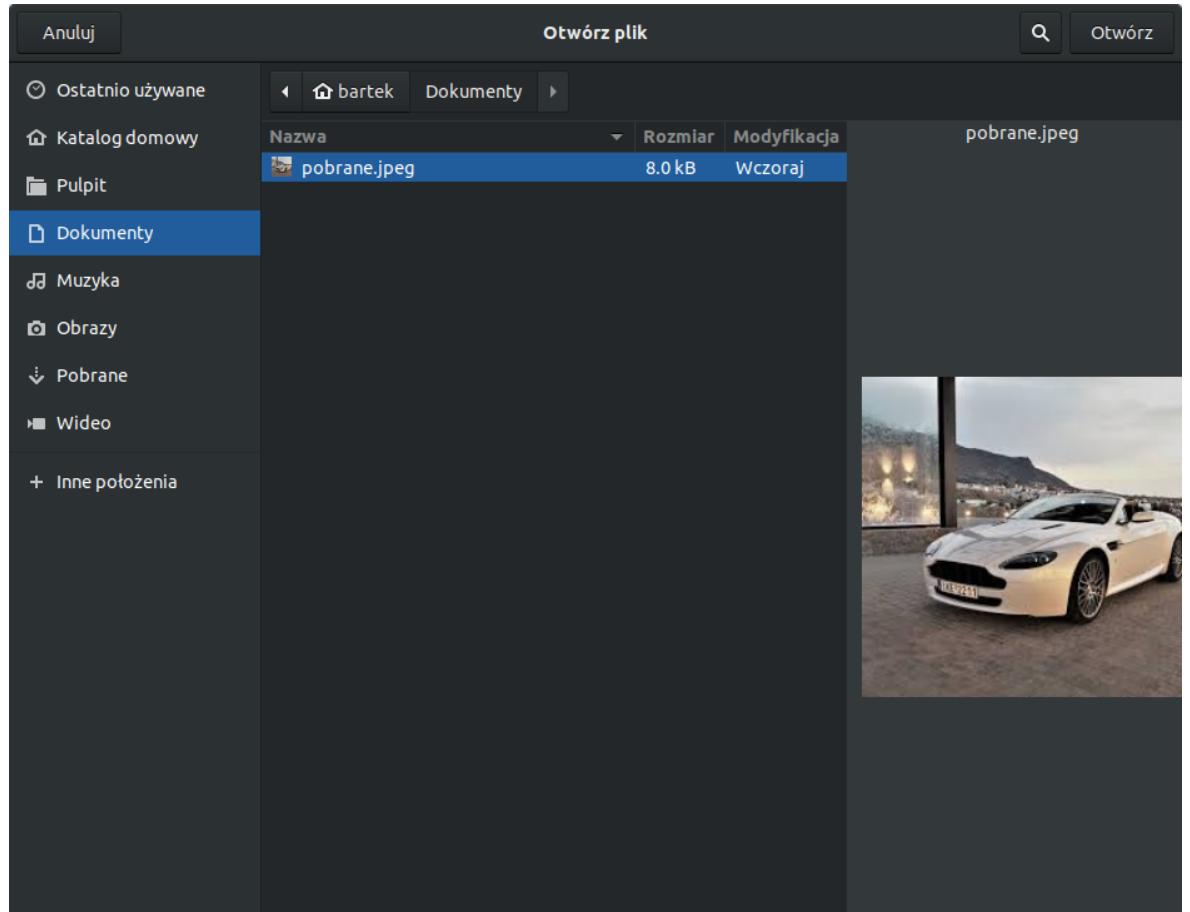
1.9. Dodawanie opinii

Zalogowany użytkownik może dodać jedną opinię o działalności „Car-Clean”. W tym celu należy przejść do zakładki „Opinie”, a następnie wcisnąć przycisk „Dodaj opinię”.

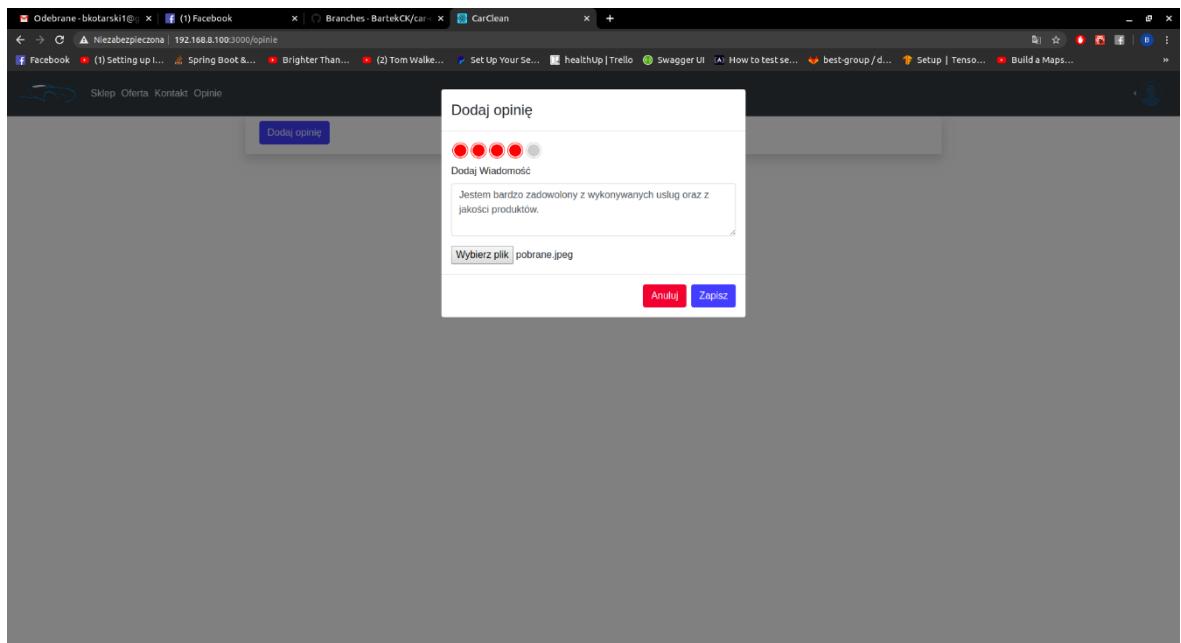


Rysunek 1.9.1 Panel dodawania opinii.

Należy wybrać ocenę według skali, wpisać wiadomość tekstową oraz wysłać fotografię wraz ze stanem wizualnym swojego pojazdu, po wykonaniu przez nas danej usługi.

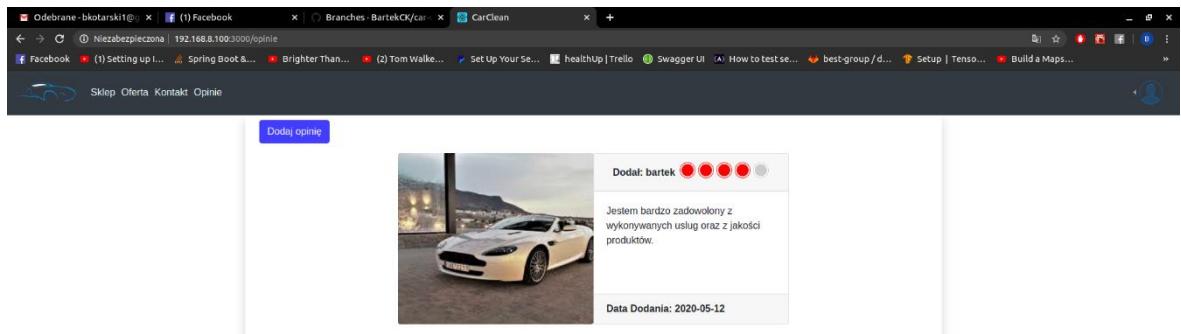


Rysunek 1.9.2. Okno dialogowe wyboru fotografii do opinii.



Rysunek 1.9.3 Kompletny uzupełniony formularz.

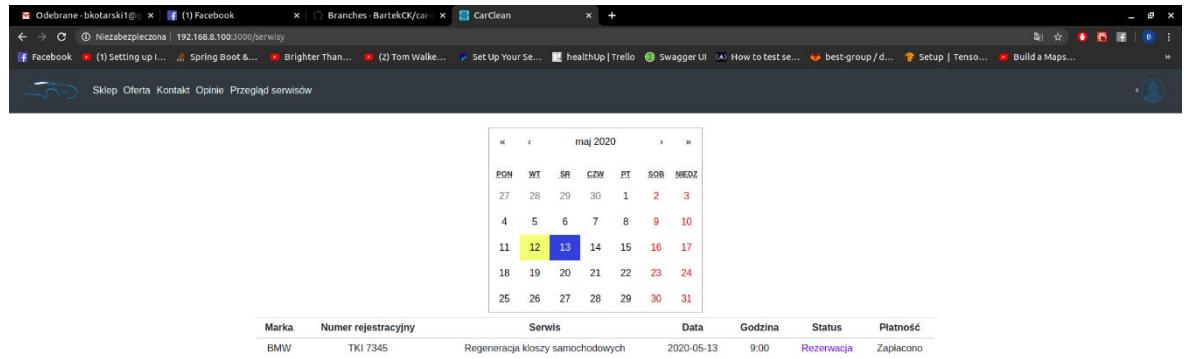
Po wybraniu zdjęcia z dysku użytkownika, należy zatwierdzić wybór.



Rysunek 1.9.4 Przedstawienie dodanej opinii

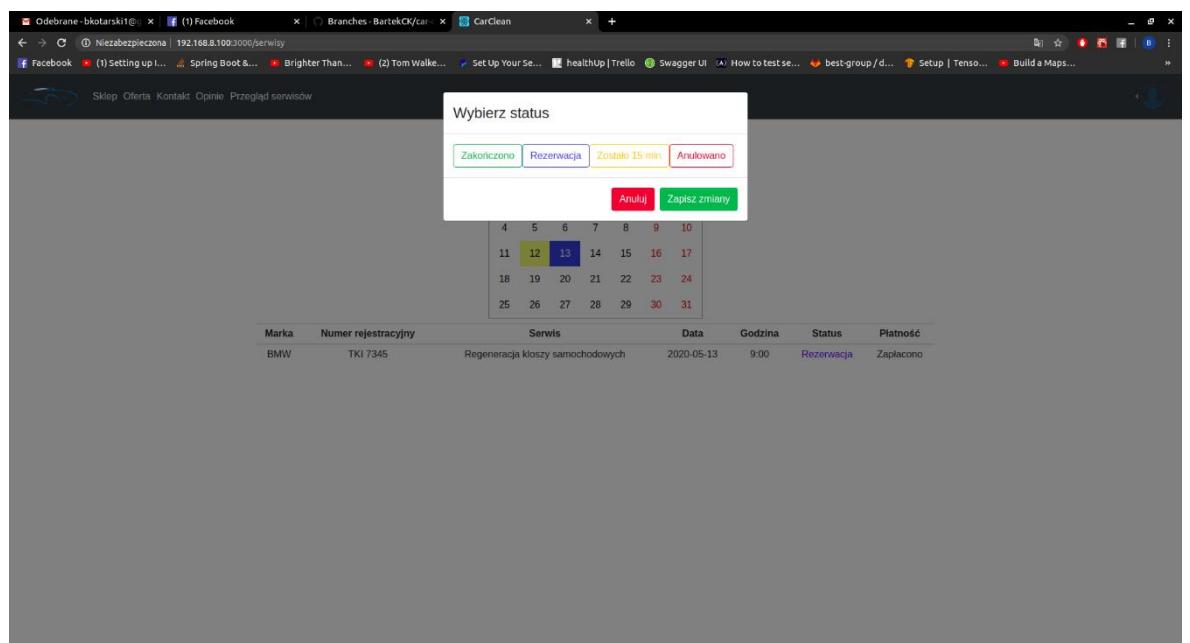
1.10. Zamiana statusu zamówienia przez pracownika

Aby wykonać daną usługę należy być użytkownikiem zalogowanym jak również posiadać odpowiednie uprawnienia dostępu do aplikacji. Jeżeli oba warunki są spełnione, należy przejść do zakładki „Przegląd serwisów” oraz wybrać odpowiednią datę.



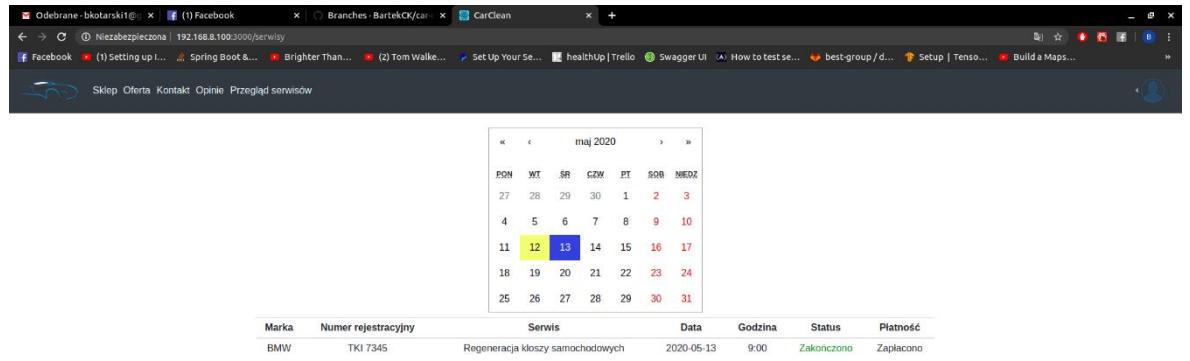
Rysunek 1.10.1 Panel pracowniczy.

Następnie należy wybrać odpowiedni serwis, aby otworzyć dodatkowe okno dialogowe.



Rysunek 1.10.2 Okno modalne zmiany statusu zamówienia

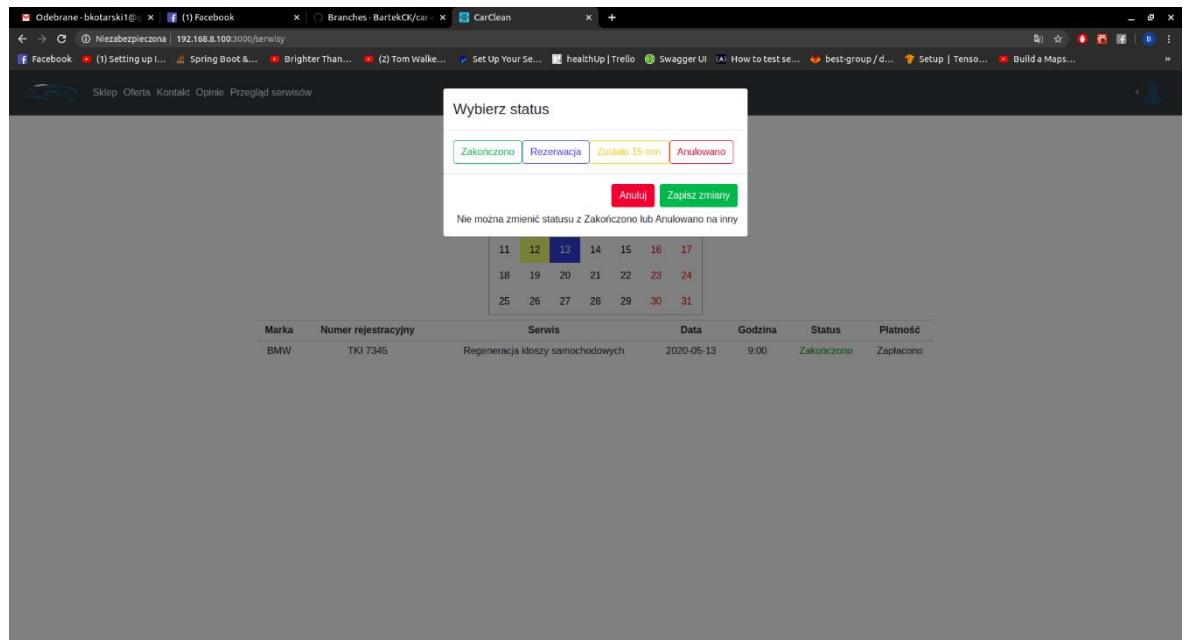
Z otwartego okna pracownik może wybrać interesujący go do zmiany status i zatwierdzić.



Rysunek 1.10.3 Potwierdzenie zmiany statusu.

Uwaga!

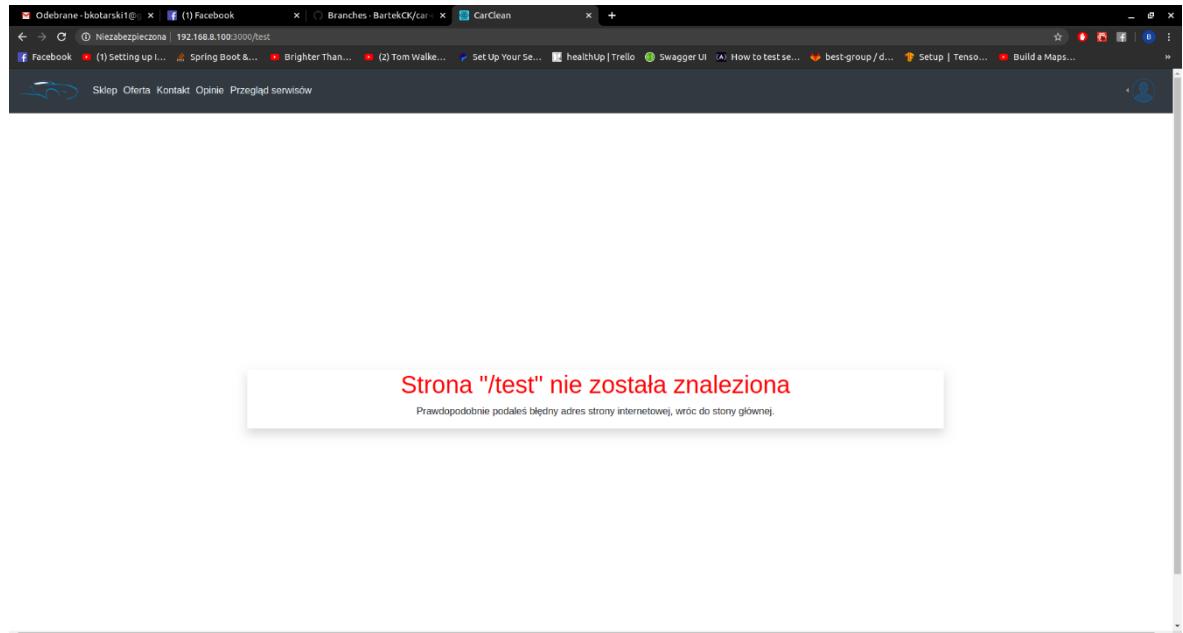
Jeżeli pracownik chce zmienić serwis o statusie Anulowano lub Zakończono na inny, zostanie poinformowany o błędzie stosownym komunikatem.



Rysunek 1.10.4 Informacja o błędzie podczas zmiany statusu na błędny.

1.11. Błąd 404

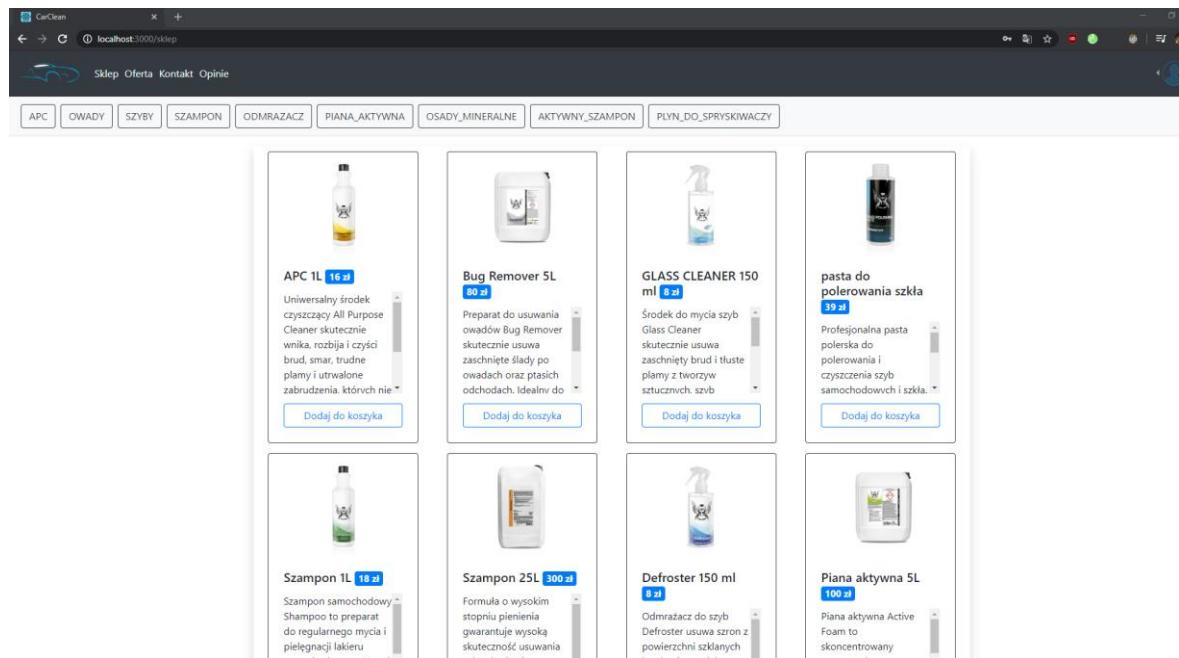
Jeżeli gość aplikacji, wprowadzi błędny adres URL zostanie o tym poinformowany stosownym komunikatem.



Rysunek 1.11.1 Strona 404 wyświetlająca nieprawidłowy adres URL.

1.12. Składanie zamówienia na produkty

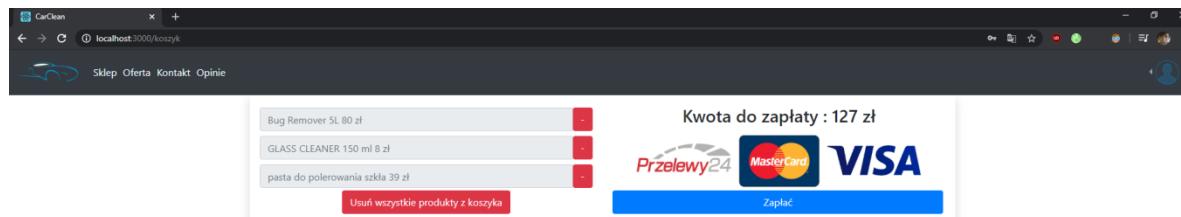
Sklep z produktami dostępny jest dla wszystkich. Jednak zakupów może w nim dokonywać tylko zalogowany użytkownik. Po pomyślnym zalogowaniu się należy przejść do zakładki „Sklep”.



Rysunek 1.12.1 Sklep z produktami

U góry dostępne są kategorie produktów. Użytkownik może po nich filtrować klikając na wybraną kategorię. Celem ponownego wyświetlenia wszystkich produktów

należy nacisnąć „Usuń filtry”. Zakupów dokonuje się najpierw poprzez dodawanie produktów do koszyka. Aby dodać produkt należy nacisnąć przy wybranym artykule „Dodaj do koszyka”. Po wybraniu produktów należy przejść do zakładki „Koszyk” widocznej po naciśnięciu na ikonę użytkownika po prawej stronie paska górnego.

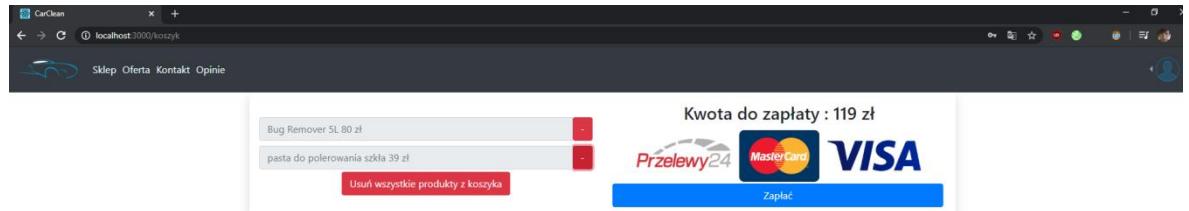


Rysunek 1.12.2 Widok koszyka

Po lewej widoczne są wybrane produkty. Naciśnięcie przycisku „Zapłać” spowoduje utworzenie zamówienia i przejście do płatności, które zostaną omówione w punktach 1.15 i 1.16.

1.13. Operacje wykonywane na koszyku zakupowym

Zalogowany użytkownik, po wyborze produktów może przejść do zakładki „Koszyk”. Wówczas ma do wyboru kilka funkcji na nim. Pierwszym z nich jest usuwanie pojedynczych produktów z koszyka. Dokonuje się tego naciskając obok produktu przycisk “-”. Lista zostaje po tym odświeżona. Drugą funkcją jest czyszczenie koszyka, czyli usunięcie z niego wszystkich produktów. Dokonuje się tego przyciskiem „usuń wszystkie produkty z koszyka”.



Rysunek 1.13.1 Koszyk.

Ostatnią operacją jest Tworzenie zamówienia. Dokonuje się tego naciskając przycisk „Zapłać”. Powyżej zaprezentowana jest cena przyszłego zamówienia.

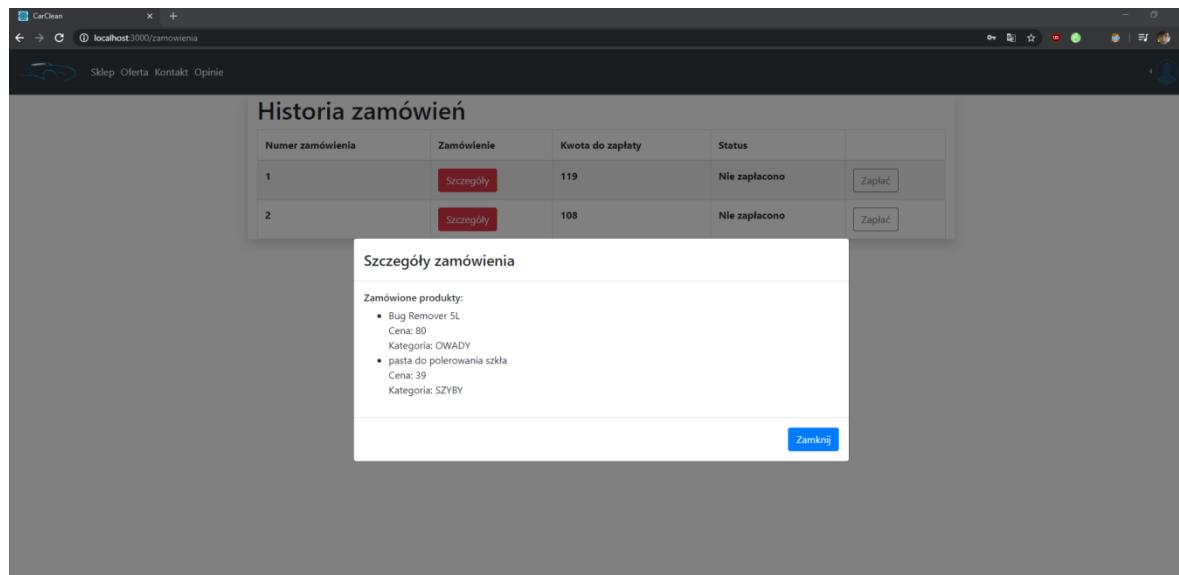
1.14. Przeglądanie zamówień produktów

Każde dokonane przez użytkownika zamówienia umieszczane jest w zakładce „Zamówienia”. Znajduje się ona pod ikoną użytkownika po prawej stronie menu górnego.

Historia zamówień				
Numer zamówienia	Zamówienie	Kwota do zapłaty	Status	
1	Szczegóły	119	Nie zapłacono	Zapłać
2	Szczegóły	108	Nie zapłacono	Zapłać

Rysunek 1.14.1 Historia zamówień

Ukazuje się tabela z historią zamówień. Numer zamówienia to jego identyfikator, obok znajduje się przycisk „Szczegóły”. Zawiera on szczegółowe dane o zamówieniu. Obok łącznej kwoty zapłaty i statusu jest przycisk „Zapłać” dostępny tylko jeśli nie opłacono jeszcze zamówienia.



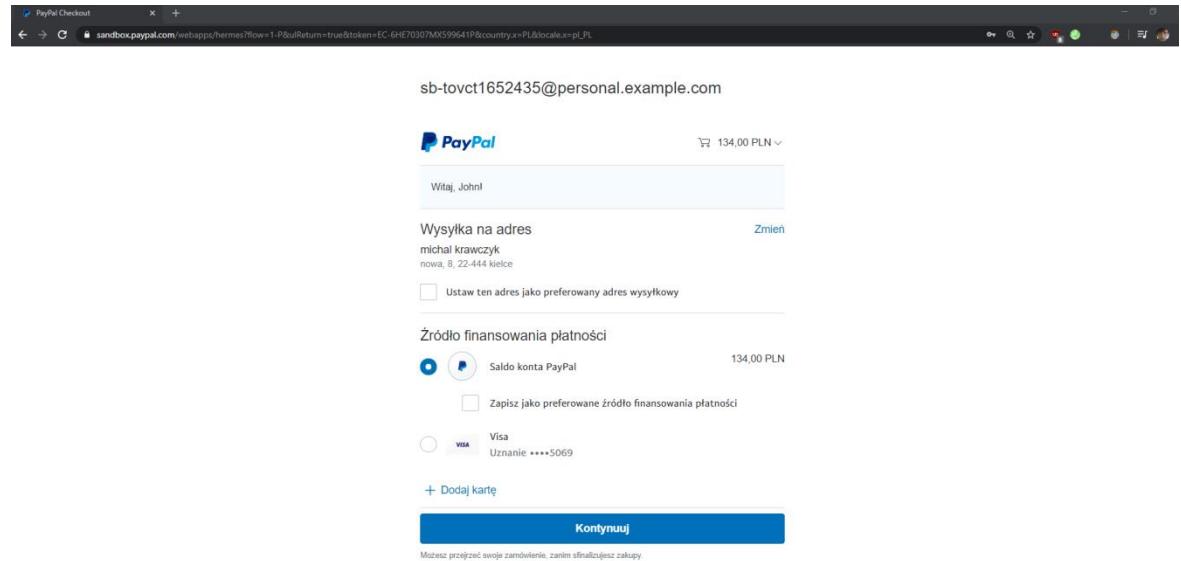
Rysunek 1.14.2 Szczegóły zamówienia

1.15. Płatność za produkty przy pomocy PayPal

Po zatwierdzeniu koszyka (po naciśnięciu przycisku „Zapłać”) użytkownik zostaje przekierowany do okna płatności. Wówczas, aby zrealizować płatność przez PayPal musi Podać adres, pod który zostanie wysłane zamówienie. Należy też wybrać rodzaj płatności „PayPal”. Po wprowadzeniu danych nacisnąć „Potwierdź”.

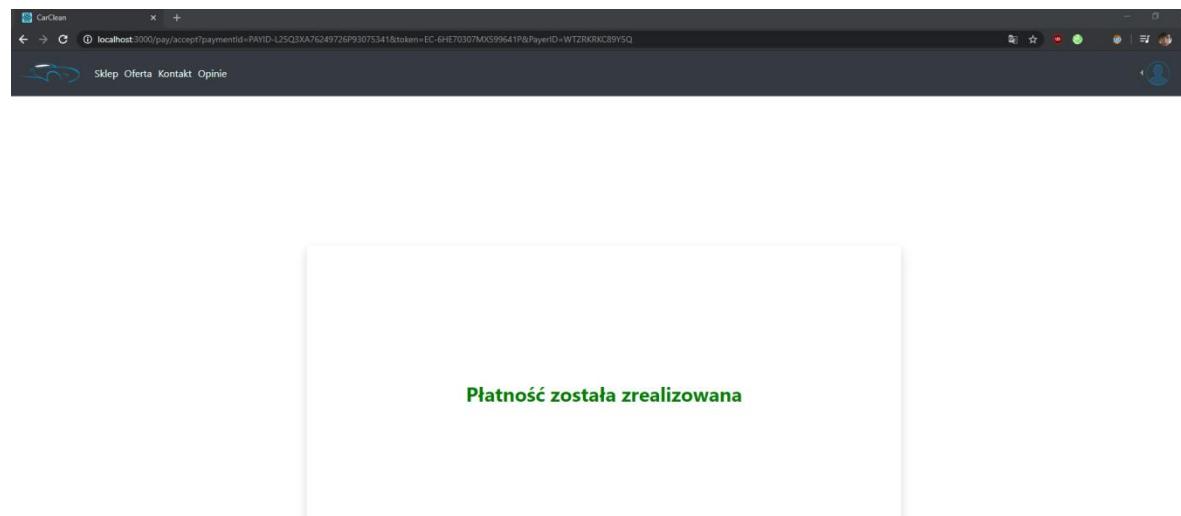
Rysunek 1.15.1 Okno szczegółów płatności

Użytkownik zostanie przekierowany na stronę płatności. Wymagane jest logowanie.



Rysunek 1.15.2 Realizacja płatności w PayPal

Adres do wysyłki powinien pokazać się w formularzu płatności

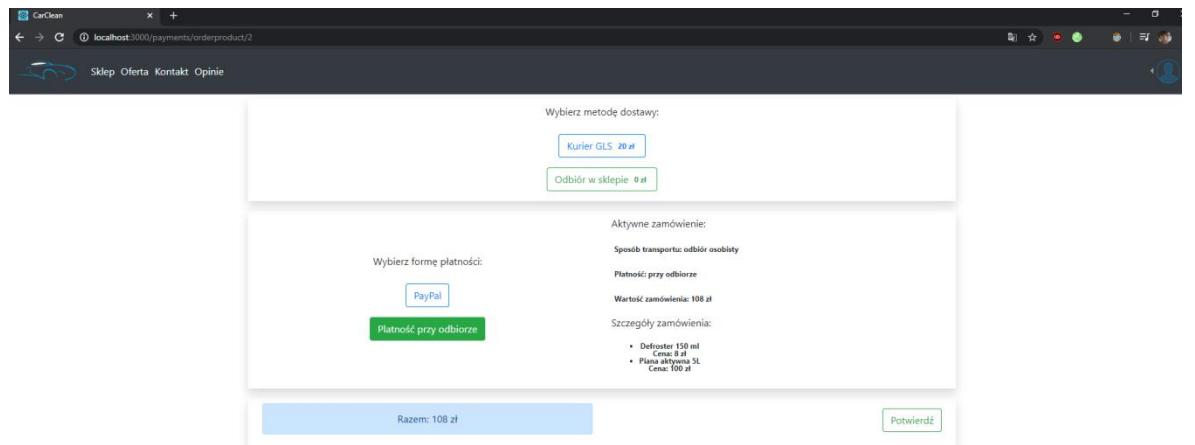


Rysunek 1.15.3 Koniec płatności

Po pomyślnym zrealizowaniu płatności status zamówienia w zakładce „Zamówienia” powinien się zmienić na „Zapłacono”.

1.16. Płatność za produkty przy odbiorze

Gdy użytkownik chce zapłacić za produkty przy odbiorze powinien on po utworzeniu zamówienia Zaznaczyć opcję „Odbiór w sklepie” i przy formie płatności wybrać „Płatność przy odbiorze”. Wówczas po zatwierdzeniu przekierowany on zostanie na stronę główną.



Rysunek 1.16.1 Okno szczegółów płatności

2. OMÓWIENIE WYKORZYSTANYCH WARSTW

Aplikacja została podzielona na trzy warstwy, a konkretnie na warstwę:

- prezentacji,
- biznesową,
- danych.

Za warstwę prezentacji odpowiada aplikacja napisana za pomocą biblioteki React, która pozwala na reaktywne tworzenie stron internetowych.

Warstwie biznesowej odpowiada logika całej aplikacji, która została stworzona przy pomocy Spring Boot.

Baza danych, z którą łączy się aplikacja backendowa zajmuje się warstwą danych. PostgreSQL został wybrany pod to zadanie.

2.1. Dodanie serwisu przez użytkownika

- 1) Użytkownik wchodzi w podstronę „Oferta” co wiąże się z automatycznym zapytaniem GET do serwera.

```
componentDidMount = async () => {
  try {
    const result = await getAllServices();
    this.setState({ offers: result });
  } catch (e) {
    const error = { message: 'Brak dostępnych ofert' };
    this.setState({ error: error });
  }
};
```

- 2) Serwer dostaje żądanie, wysyła zapytanie do bazy danych, aby pobrać wszystkie oferty dostępne w bazie.

```
@GetMapping //controllers/ServicesController.java
public List<ServicesDto> getAllServices() {
  return servicesService.getAllServices();
}
```

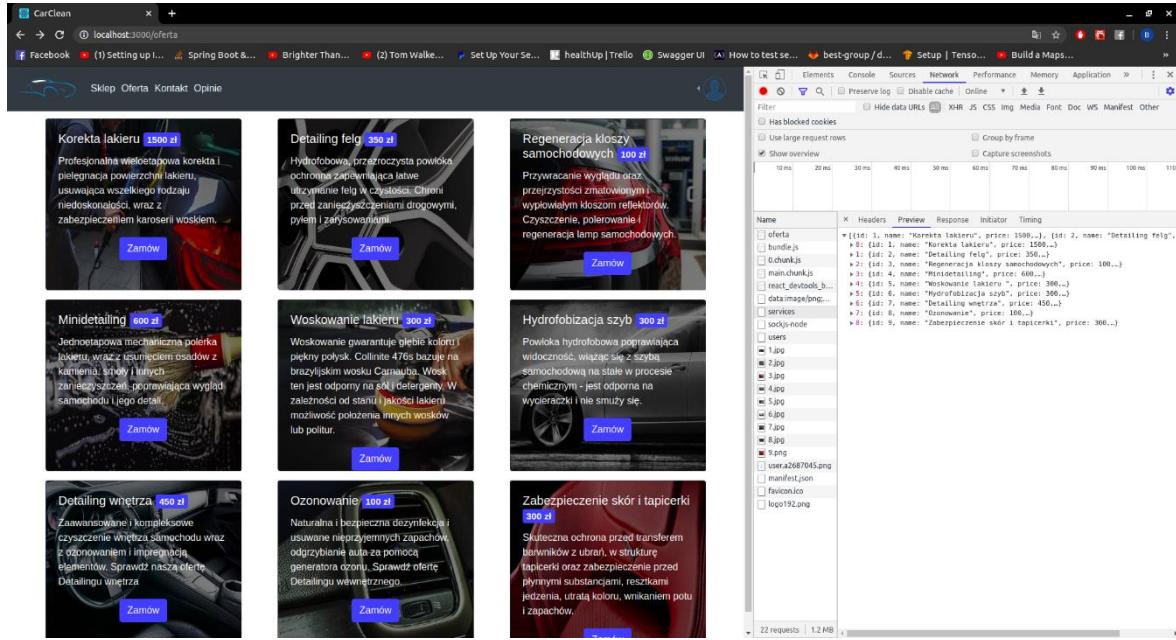
```

@Override//services/ ServicesServiceImpl.java

    public List<ServicesDto> getAllServices() {
        return servicesRepository.findAll().stream().map(
service-> ServicesDto.build(service)).collect(Collectors.toList());
    }

```

- 3) Baza danych zwraca na serwer wszystkie oferty wraz z własnościami jak nazwa, cena, id i opis.
- 4) Serwer zwraca otrzymane dane jako listę do warstwy prezentacji.
- 5) Warstwa prezentacji wyświetla otrzymane dane według zaprogramowanego szablonu



Rysunek 2.1.1 Odpowiedź serwera z dostępnymi ofertami

- 1) Użytkownik kliką przycisk zamów na wybranej ofercie, wysyłając automatycznie dwa żądania na serwer. Pierwsze z nich dotyczy wybranej oferty, drugie wszystkich pojazdów użytkownika.

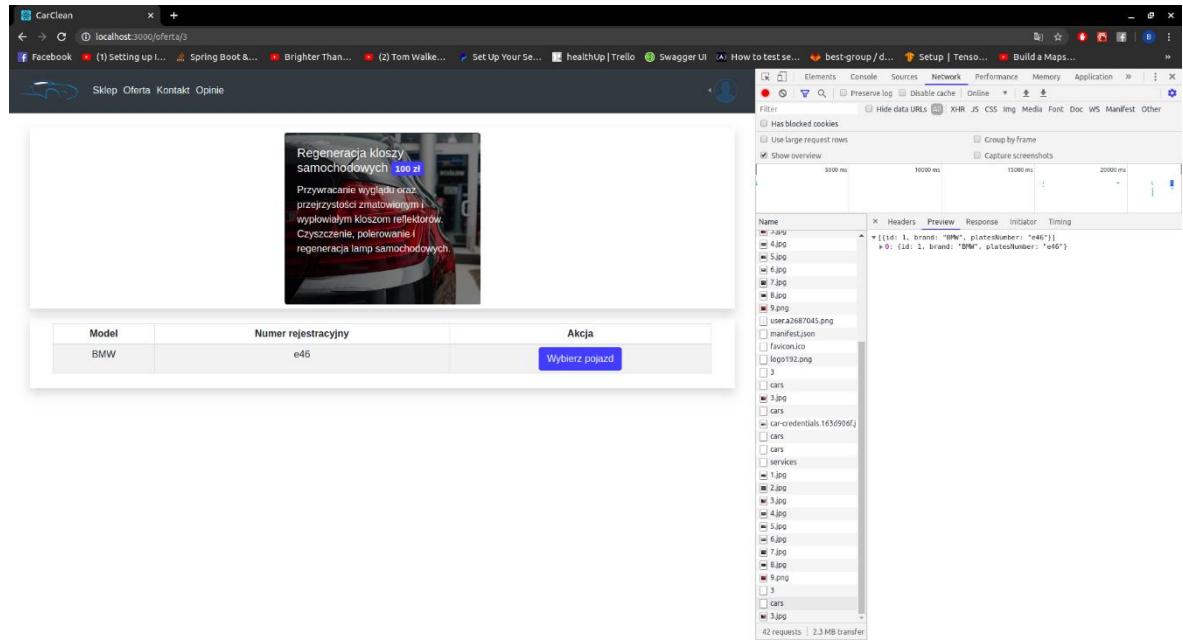
```
componentDidMount = async () => { // /pages/ ChosenOffer.js

  try {
    const result = await getServiceById(this.props.match.params.id);
    const cars = await getAllUserCars();
    this.setState({ cars: cars.data, showModal: true, offer: result });
  } catch (e) {
    if (e.response) this.setState({ error: e.response.data });
    else this.setState({ error: 'Brak oferty' });
  }
};
```

- 2) Serwer przetwarza żądania i komunikuje się z bazą danych.

```
@Override // /services/ CarServicesImpl.java
public List<CarDto> getAllUserCars(String username) {
  return carRepository.findByUserUsername(username)
    .stream()
    .map(car ->
      CarDto.build(car.orElseThrow(() ->
        new RuntimeException("Pojazd nie został znaleziony")))
    )
    .collect(Collectors.toList());
}
```

- 3) Baza danych zwraca wyniki według zapytania.
- 4) Serwer sprawdza otrzymane dane, jeżeli użytkownik nie ma pojazdów to zwraca błąd, jeżeli ma to wysyła listę z pojazdami użytkownika.
- 5) Warstwa prezentacji wyświetla listę jako tabelę, a jeżeli dostanie informacje o braku pojazdów to ukazuje stosowany komunikat.

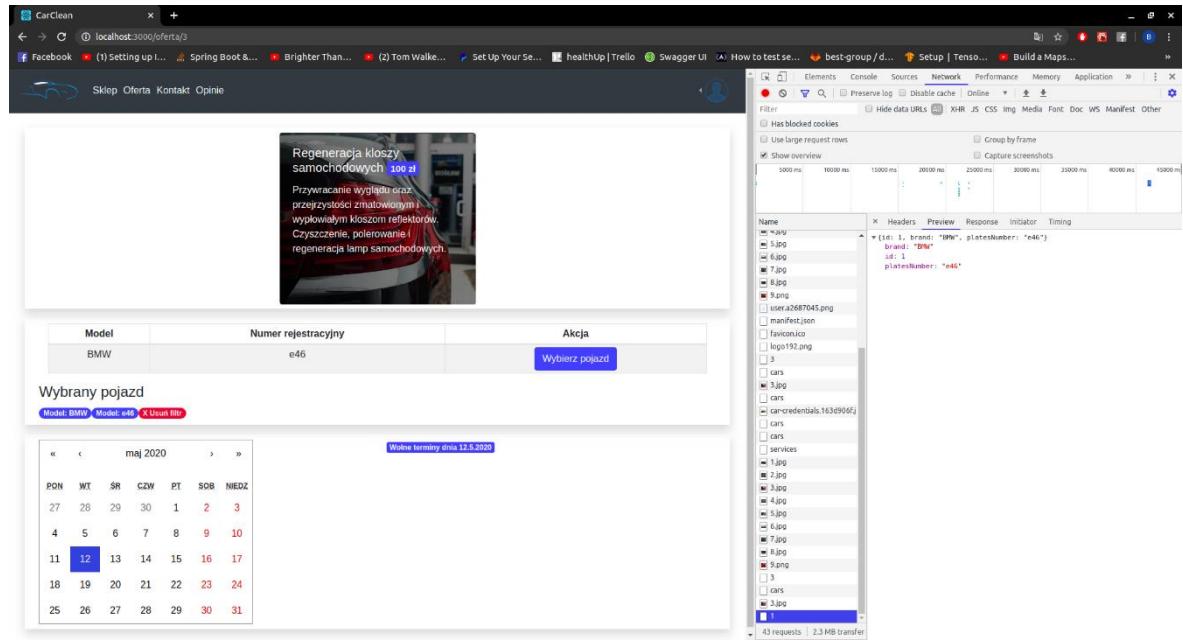


Rysunek 2.1.2 Odpowiedź zawierająca listę pojazdów użytkownika.

- 1) Użytkownik kliką wybierz pojazd po stronie warstwy prezentacji wysyłając żądanie na serwer sprawdzające czy pojazd na pewno istnieje i należy do niego.
- 2) Serwer dostaje nazwę użytkownika oraz dane pojazdu. Wysyła zapytanie do bazy danych wyciągające pojazd o tych danych dla tego rodzaju użytkownika.

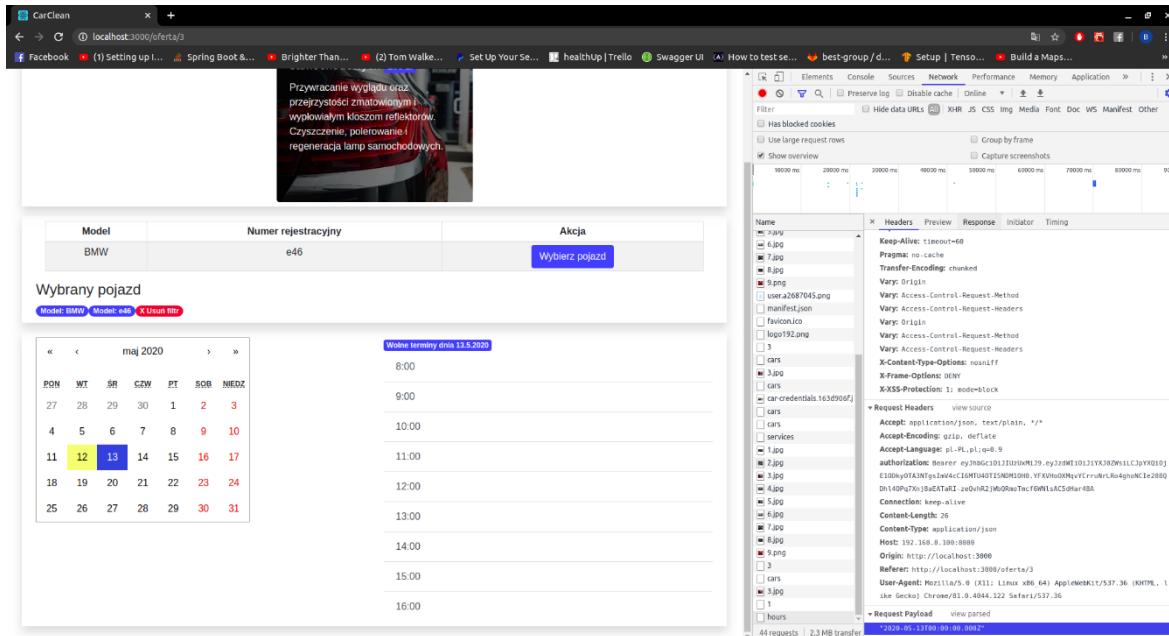
```
@Override//services/ CarServicesImpl.java
public CarDto getUserCar(String username, int carId) {
    Car car = carRepository.findByUserUsernameAndId(username, carId).orElseThrow(() -
> new RuntimeException("Pojazd nie został znaleziony"));
    return CarDto.build(car);
}
```

- 3) Baza danych zwraca stosowane dane.
- 4) Serwer porównuje dane i wysyła pojazd wraz z potwierdzeniem na warstwę prezentacji.
- 5) Warstwa prezentacji dostaje status 200 oraz pojazd, więc wyświetla kalendarz z wyborem daty.



Rysunek 2.1.3 Odpowiedź zawierająca wybrany pojazd użytkownika.

- Użytkownik po stronie prezentacji kliką w wybrany dzień na kalendarzu, co wiąże się z wysłaniem tej daty na serwer.



Rysunek 2.1.4 Zapytanie zawierające wybraną datę.

- Serwer dostaje żądanie i formułuje zapytanie do bazy danych, aby zwróciła wszystkie serwisy tego dnia.
- Baza danych zwraca serwisy według polecenia.
- Serwer iteruje po liście i sprawdza występujące godziny pracy warsztatu. Oznacza to, że sprawdza czy jest wystąpienie godziny 8, jeżeli nie to dodaje 8 do

listy wolnych godzin i sprawdza dla 9, sytuacja powtarza się aż do ostatniej określonej godziny pracy. Godziny pracy są określane we właściwościach aplikacji, dzięki czemu są łatwe to zmiany. Serwer wysyła listę wolnych godzin na warstwę prezentacji.

```
@Override//services/ OrderServiceServiceImpl.java
public ResponseEntity<List<Integer>> getFreeHoursByDay(LocalDate localDate) {
    List<Integer> freeHours = new ArrayList<>();
    try {
        List<OrderService> orderServiceList = orderServiceRepository.findByDate(localDate).orElseThrow(() -> new Exception("Wybrany dzień nie ma zaplanowanych wizyt"));
        for (int i = startWorkHour; i <= endWorkHour; i++) {
            int temp = i;
            boolean decision = orderServiceList.stream().anyMatch(el -> el.getTime() == temp);
            if (!decision)
                freeHours.add(i);
        }
    } catch (Exception e) {
        logger.info("{}" , e.getMessage());
        for (int i = startWorkHour; i <= endWorkHour; i++)
            freeHours.add(i);
    }

    if (freeHours.isEmpty())
        return ResponseEntity.notFound().build();
}
return ResponseEntity.ok(freeHours);
}
```

Rysunek 2.1.5 Odpowiedź w formie listy ukazująca dostępne godziny wybranego dnia.

5) Warstwa prezentacji dostaje wynik i wyświetla w postaci tabelki.

- Użytkownik wybiera interesującą go datę i zatwierdza. Wysyła automatycznie żądanie z zapisem na usługę, konkretnego dnia i godziny wraz z wybranym pojazdem.

Rysunek 2.1.6 Wysłanie żądania o rezerwacji.

- 2) Serwer dostaje dane przetwarza je na swoje modele i wysyła żądanie do bazy danych, aby zapisać dane.
- 3) Baza danych zapisuje dane lub zwraca błąd.
- 4) Serwer dostaje informację o zapisie. Jeżeli zapis udał się pozytywnie zwraca status operacji wraz z danymi dotyczącymi zapisu.

```

@Override //services/ OrderServiceServiceImpl.java
@Transactional
public ResponseEntity<? extends Object> addReservationService(String username, CreateOrderService
ceDto createOrderServiceDto) {
    try {
        if (orderServiceRepository.existsByDateEqualsAndTimeEquals(LocalDate.parse(createOrderS
erviceDto.getDate()), createOrderServiceDto.getTime()))
            throw new RuntimeException("Wybrany termin jest niedostępny");
    } catch (DateTimeException e) {
        throw new RuntimeException("Błędny format daty, poprawny to YYYY-MM-DD");
    }

    Car car = carRepository.findByUserUsernameAndId(username, createOrderServiceDto.getCarId())
.orElseThrow(() -> new RuntimeException("Błędny pojazd"));
    User user = userRepository.findByUsername(username).orElseThrow(() -
> new UsernameNotFoundException("Brak osoby rozpoczynającej"));

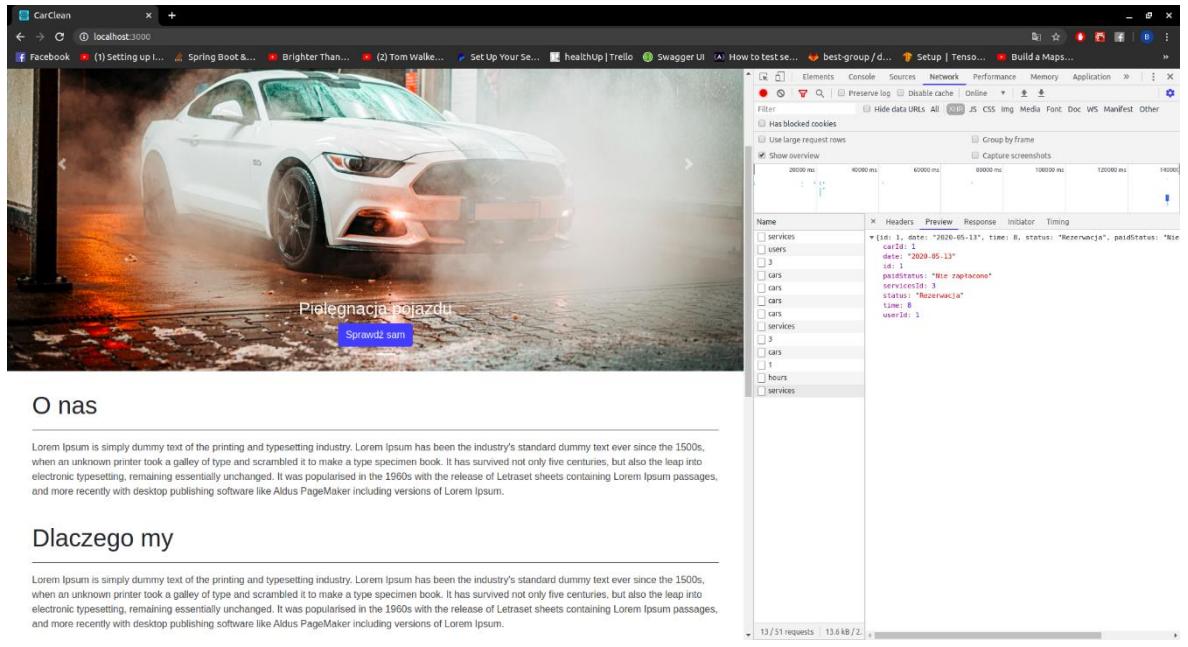
    Services services = servicesRepository.findById(createOrderServiceDto.getServicesId()).orEl
seThrow(() -> new RuntimeException("Błędny wybrany serwis"));

    OrderService orderService = OrderService.builder()
        .date(LocalDate.parse(createOrderServiceDto.getDate()))
        .time(createOrderServiceDto.getTime())
        .status(OrderServiceStatus.WAITING)
        .paidStatus(PaidStatus.NOT_PAID)
        .car(car)
        .user(user)
        .serviceid(services)
        .build();

    orderServiceRepository.save(orderService);
    return ResponseEntity.ok(CreateOrderServiceDto.build(orderService));
}

```

- 5) Warstwa prezentacji, jeżeli dostanie wiadomość i status 200 to przekierowuje użytkownika na stronę główną aplikacji.



O nas

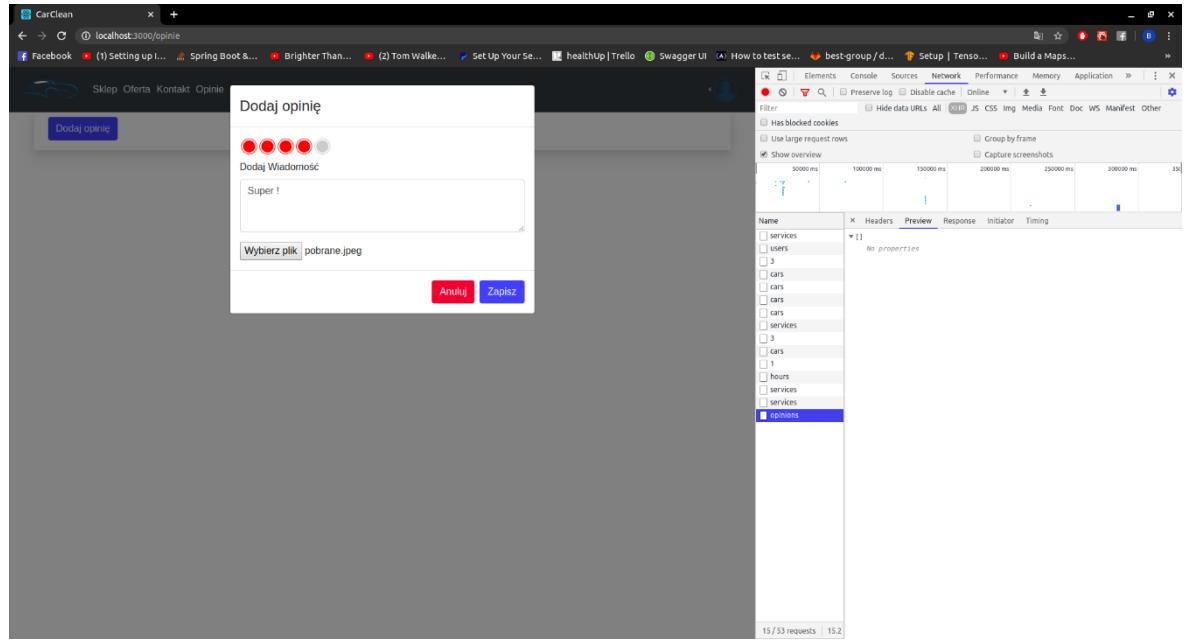
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Dlaczego my

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Rysunek 2.1.7 Odpowiedź serwera na rezerwacje wykonaną przez użytkownika.

2.2. Dodanie opinii przez użytkownika



Rysunek 2.2.1 Zwrócona pusta lista opinii i okno modalne do dodania nowej.

- 1) Użytkownik po stronie prezentacji dodaje opinię. Wybiera ocenę, wiadomość tekstową i fotografię. Wybrane dane wysyła na serwer.

```
const safeOpinion = async () => {//pages/Opinions/index.js
    let formData = new FormData();
    formData.append('file', file);
    formData.append(
        'createOpinionDto',
        new Blob([
            JSON.stringify({
                ...opinion,
            }),
        ],
        { type: 'application/json' }
    )
);

try {
    await addOpinionByUser(formData);
    hide();
} catch (e) {
    console.log(e);
}
};
```

- 2) Serwer otrzymuje dane w JSON oraz dane binarne. Sprawdza rozszerzenie pliku, jeżeli jest dobre to konwertuje dane binarne na plik fizyczny i zapisuje go w udostępnianym katalogu media/opinions. Następnie wysyła dane ze ścieżka do pliku, oceną i wiadomością tekstową do bazy danych w celu zapisu.

```
//services/OpinionServiceImpl.java
private String saveFile(Long userId, MultipartFile file) throws IOException {

    String extension = FilenameUtils.getExtension(file.getOriginalFilename());
    if (!(extension.equals("png") || extension.equals("jpg") || extension.equals("jpeg")))
        throw new RuntimeException("Błędne rozszerzenie pliku");

    String imageName = new StringBuilder().append(userId).append(".").append(extension).toString();

    byte[] bytes = file.getBytes();
    Path path = Paths.get(folder + imageName);
    Files.write(path, bytes);
    return imageName;
}

@Override
public ResponseEntity<CreateOpinionDto> createOpinion(String username, MultipartFile file, CreateOpinionDto createOpinionDto) throws IOException {

    User user = userRepository.findByUsername(username).orElseThrow(() -> new UsernameNotFoundException("Użytkownik " + username + " nie został odnaleziony"));

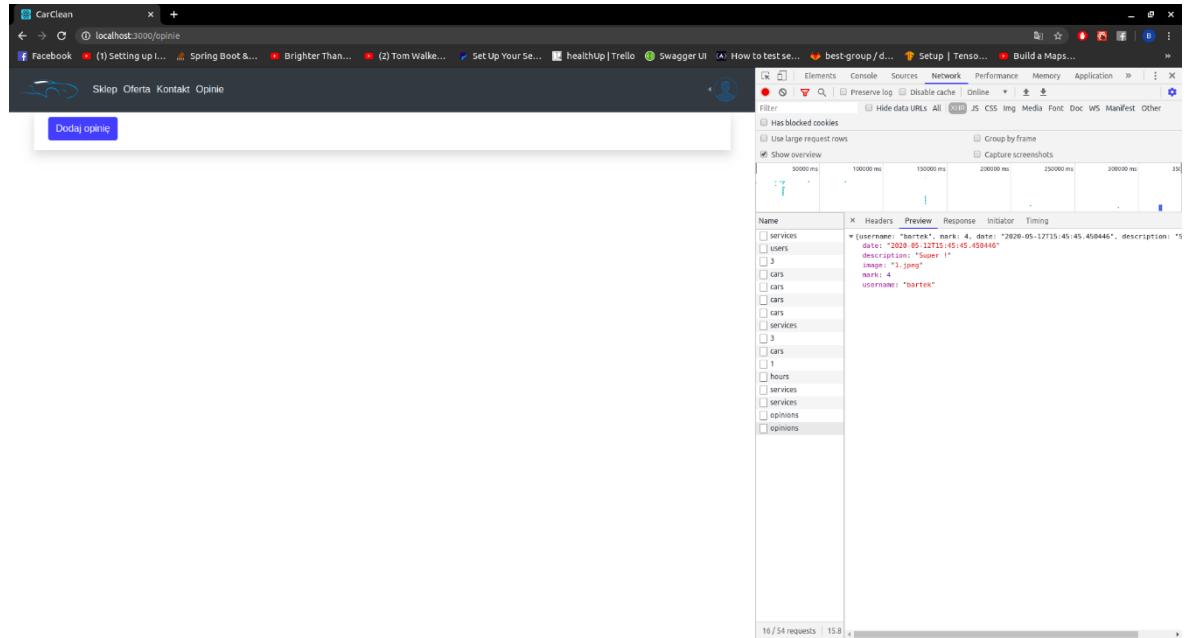
    if (user.getOpinion() != null)
        this.deleteOpinion(user.getUsername());

    String image = saveFile(user.getId(), file);
    Opinion opinion = Opinion.builder()
        .user(user)
        .date(LocalDateTime.now())
        .description(createOpinionDto.getDescription())
        .mark(createOpinionDto.getMark())
        .image(image)
        .build();

    opinionRepository.save(opinion);
    return ResponseEntity.ok(CreateOpinionDto.build(opinion));
}
```

- 3) Baza danych zapisuje dane.

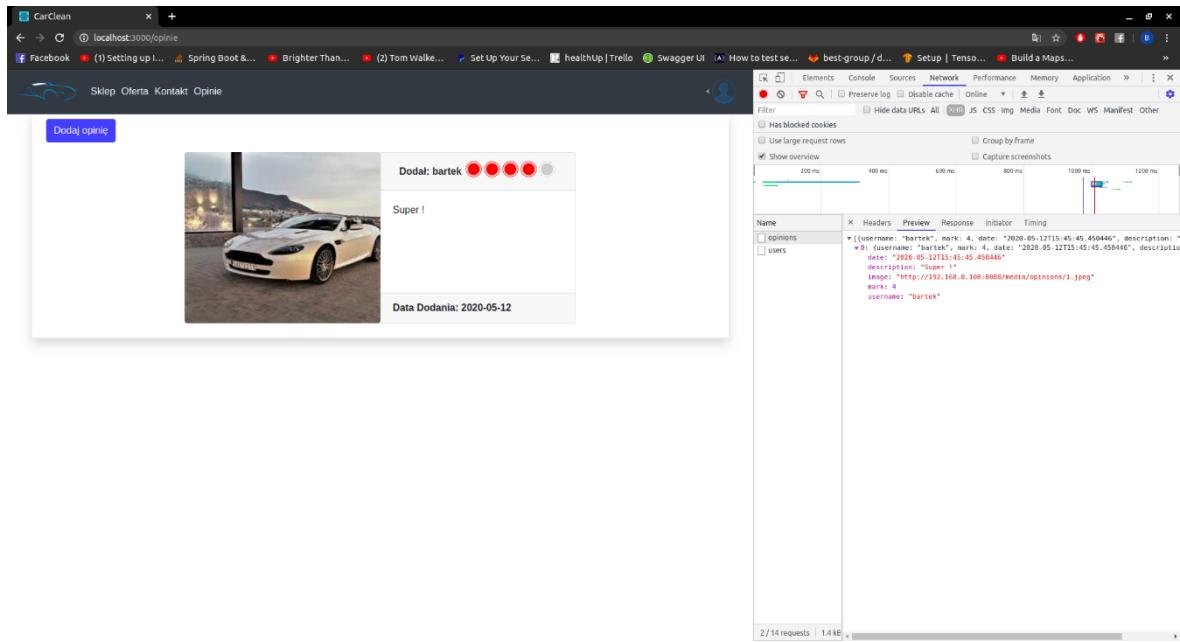
- 4) Jeżeli serwer nie dostanie błędu oznacza, że dane zostały zapisane, więc zwraca status operacji na warstwę prezentacji.
- 5) Warstwa prezentacji dostaje status pozytywny, zamyka dzięki temu okno modalne dotyczące dodawania opinii.



Rysunek 2.2.2 Odpowiedź zamk傢ca okno modalne, informuj膮ca o pozytywnym dodaniu opinii.

W celu wyświetlenia dodanej opinii, użytkownik musi wysłać zapytanie GET poprzez odświeżenie strony. Serwer w tym wypadku konwertuje ścieżkę zdjęć dla aktualnej domeny.

```
@Override//services/OpinionServiceImpl.java
public ResponseEntity<List<CreateOpinionDto>> getAllOpinions() {
    List<Opinion> opinions = opinionRepository.findAll();
    List<CreateOpinionDto> createOpinionDto = opinions.stream().map(opinion ->
        CreateOpinionDto.build(opinion)).collect(Collectors.toList());
    createOpinionDto.forEach(opinion -> opinion.setImage(
        new StringBuilder()
            .append(this.domainName)
            .append("/")
            .append(this.folder)
            .append(opinion.getImage())
            .toString()
    ));
    return ResponseEntity.ok(createOpinionDto);
}
```

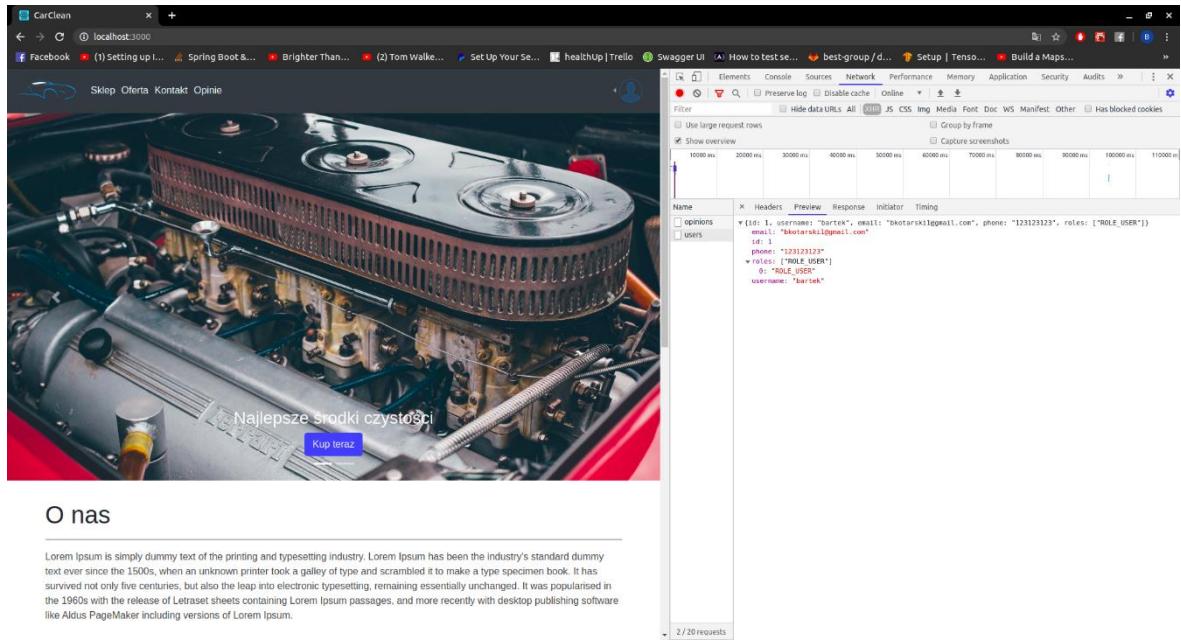


Rysunek 2.2.3 Odpowiedź zawierają listę wszystkich opinii.

2.3. Logowanie zapisanego użytkownika w przeglądarce

Jeżeli użytkownik jest już zalogowany to jego dane przechowywane są w 2 miejscach. LocalStorage zawiera zakodowany token, a reduxowy wzorzec projektowy pozwala na przechowywanie stanu, który posiada dane o roli, tokenie i czy użytkownik został zautoryzowany. W momencie odświeżania karty lub zamknięcia przeglądarki, stan reduxowy jest czyszczony i zostaje jedynie token w przeglądarce, wraz z zakodowaną datą ważności. Poniższy przypadek ukazuje „nieświadome” logowanie użytkownika, który odświeżył przeglądarkę lub ją zamknął.

```
export const authReducer = (state, action) => {//context.reducer.js
  switch (action.type) {
    case LOGIN:
      localStorage.setItem('@token', action.user.token);
      const temp = {
        isAuthenticated: true,
        token: action.user.token,
        roles: action.user.roles,
      };
      return {
        ...state,
        ...temp,
      };
    case LOGOUT:
      localStorage.clear();
      const temp2 = {
        isAuthenticated: false,
        token: '',
        roles: [],
      };
      return {
        ...state,
        ...temp2,
      };
    default:
      return state;
  }
};
```



Rysunek 2.3.1 Odpowiedź uzyskana poprzez wysłanie tokenu na serwer. Odpowiedź zawiera dane o użytkowniku.

- 1) Warstwa prezentacji wysyła żądanie na serwer wraz z tokenem zapisanym w localstorage.

```

useEffect(() => {//context/index.js
  const searchUser = async () => {
    const token = localStorage.getItem('@token');
    if (token) {
      try {
        const user = (await getUserByToken()).data;
        user.token = token;
        dispatch({ type: LOGIN, user: { ...user } });
      } catch (e) {
        console.log(e);
      }
    }
  };
  searchUser();
}, []);

```

- 2) Serwer dostaje token. Dekoduje go, sprawdzając jego ważność oraz przynależność. Przynależność jest sprawdzana za pomocą sekretnego klucza, który zakodował podpis tokenu jwt. Jeżeli ważność i przynależność są prawidłowe to system wyciąga z tokenu nazwę użytkownika, którą

wysyła do bazy danych w celu pobrania większej ilości danych (rola, id, email, telefon).

```
public boolean validateToken(String token) { //JwtProvider.java
    try {
        Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
        return true;
    } catch (SignatureException e) {
        logger.error("Invalid signature: {}", e.getMessage());
    } catch (MalformedJwtException e) {
        logger.error("Invalid token: {}", e.getMessage());
    } catch (ExpiredJwtException e) {
        logger.error("Expired token: {}", e.getMessage());
    } catch (UnsupportedJwtException e) {
        logger.error("Unsupported token: {}", e.getMessage());
    } catch (IllegalArgumentException e) {
        logger.error("Claims string is empty: {}", e.getMessage());
    }
    return false;
}
```

```
@Override //AuthorizationServiceImpl.java
public ResponseEntity loginUser(SignInDto signInDto) {
    Authentication authentication = manager.authenticate(
        new UsernamePasswordAuthenticationToken(
            signInDto.getUsername(),
            signInDto.getPassword()
        )
    );
    SecurityContextHolder.getContext().setAuthentication(authentication);
    String token = provider.generateToken(authentication);

    return ResponseEntity.ok(new JwtTokenDto(token));
}
```

- 3) Baza danych zwraca dane według nazwy użytkownika.
- 4) Serwer formatuje te dane według określonego schematu dto, żeby zwrócić na warstwę prezentacji w bezpieczny sposób, odpowiednie informacje.

```
@Override //AuthorizationServiceImpl.java
public UserDto getUser(Authentication authentication) {
    return UserDto.build((UserPrincipal) authentication.getPrincipal());
}
```

- 5) Warstwa prezentacji dostaje dane uwierzytelniające i zapisuje je w stanie kontekstu. Umożliwia to dostęp do części aplikacji, które są niewidoczne dla użytkowników np. o innej roli.

2.4. Zmiana statusu usługi przez pracownika

W celu zmiany statusu najpierw musi zostać podjęta akcja wyboru serwisu do edycji.

- 1) Pracownik wysyła zapytanie (jako wybraną datę) na serwer.

```
getAllEmployeeServicesByDay(new Date(formatDate(chosenDate)))

export const getAllEmployeeServicesByDay = (body) =>
  postSafe(servicesEmployeeApiUrl(), body);
```

- 2) Serwer przetwarza datę i tworzy odpowiednie zapytanie do bazy danych.

```
List<OrderService>orders=
orderServiceRepository.findAllByDate(localDate).orElseThrow(() -> new
RuntimeException("Brak rezerwacji tego dnia"));
```

- 3) Baza danych zwraca listę serwisów tego dnia.
- 4) Serwer sprawdza tą listę, jeżeli jest pusta to zwraca błąd 404. W przeciwnym wypadku zwraca listę serwisów tego dnia.

```
List<OrderService>orders=
orderServiceRepository.findAllByDate(localDate).orElseThrow(() -> new
RuntimeException("Brak rezerwacji tego dnia"));
return ResponseEntity.ok(orderServiceDtoList);
...
public ResponseEntity<List<GetOrderServiceDto>> getAllServiceByDay(@RequestBody
LocalDate localDate) {
  return orderServiceService.getAllServiceByDay(localDate); }
```

- 5) Warstwa prezentacji dostaje dane i wyświetla je w formacie tabelki.

```
useEffect(() => {
  getAllEmployeeServicesByDay(new Date(formatDate(chosenDate)))
    .then((res) => {
      if (res.status === 200) setServiceData(res.data);
    })
    .catch((err) => console.log(err));
}, [chosenDate, isModalVisible]);
```

The screenshot shows a browser window for 'CarClean' at localhost:3000/serwisy. The main content includes a calendar for May 2020 and a table with columns: Marka, Numer rejestracyjny, Servis, Data, Godzina, Status, and Platność. Below the table is a navigation bar with Sklep, Oferta, Kontakt, Opinie, and Przegląd serwisów.

In the developer tools Network tab, a POST request to `/api/v1/employees/services` is shown. The response status is 404, and the response body contains the message: `Brak referencji tego dnia.`

Rysunek 2.4.1 Odpowiedź z błędem 404 na żądanie z datą serwisów.

This screenshot is identical to Rysunek 2.4.1, showing the same browser interface and developer tools output for a 404 error on the service data endpoint.

Rysunek 2.4.2 Treść błędu 404 w przypadku pobrania serwisów danego dnia.

The screenshot shows a web application interface for 'CarClean' on a local host. At the top, there's a navigation bar with links like 'Facebook', 'Spring Boot', 'Brighter Than...', 'Tom Walk...', 'healthUp | Trello', 'Swagger UI', and several others. Below the navigation is a menu bar with 'Sklep', 'Oferta', 'Kontakt', 'Opinie', and 'Przegląd serwisów'. A sidebar on the left contains a calendar for May 2020, with the 13th highlighted in blue. The main content area displays a table for a service appointment:

Marka	Numer rejestracyjny	Serwis	Data	Godzina	Status	Płatność
BMW	e46	Regeneracja kloszy samochodowych	2020-05-13	8:00	Rezerwacja	Nie zapłacono

In the bottom right corner of the main content area, there's a small message: 'Nie zapłacono'.

On the right side of the screen, the developer tools Network tab is open. It shows a single request to 'localhost:3000/serwisy'. The Headers section includes standard HTTP headers like 'Content-Type: application/json', 'Accept: application/json', and 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122 Safari/537.36'. The Response tab shows the JSON response from the server:

```

{
  "id": 1,
  "carInfo": {
    "id": 1,
    "brand": "BMW",
    "plateNumber": "e46"
  },
  "carInfo": {
    "id": 1,
    "brand": "BMW",
    "plateNumber": "e46"
  },
  "date": "2020-05-13",
  "id": 1,
  "paidStatus": "Nie zapłacono",
  "serviceDetails": [
    {
      "id": 3,
      "name": "Regeneracja kloszy samochodowych",
      "price": 100
    }
  ],
  "status": "Rezerwacja",
  "time": 0
}

```

Rysunek 2.4.3 Przykład żądania z wybraną przez użytkownika datą.

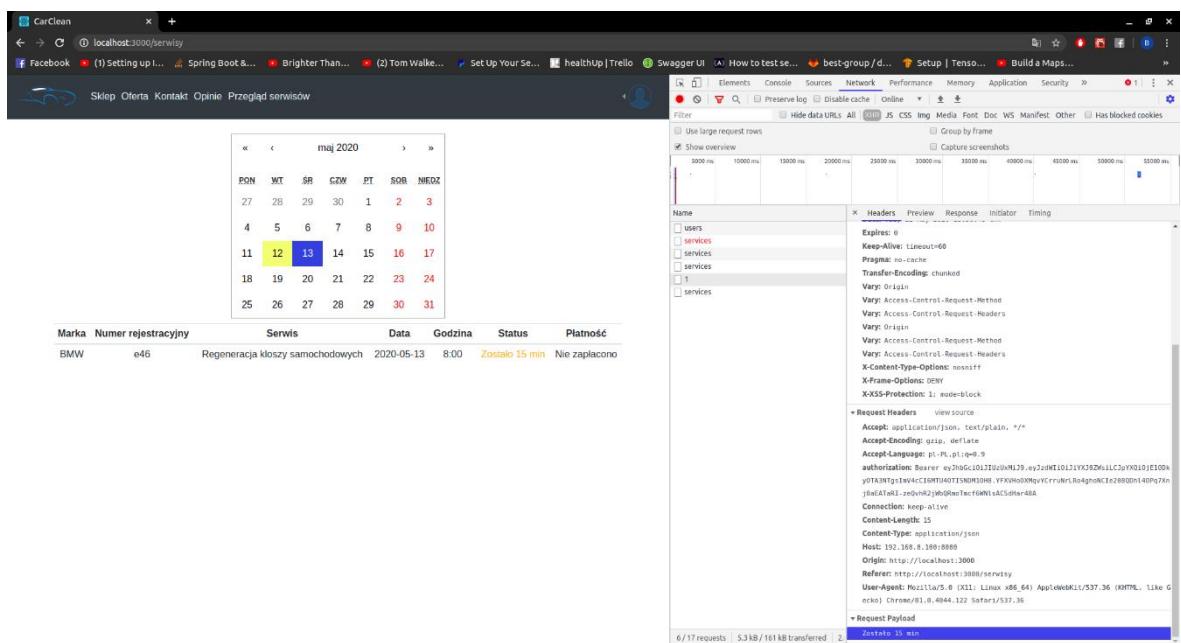
This screenshot is nearly identical to Rysunek 2.4.3, showing the same web application interface for 'CarClean'. The calendar highlights the 13th of May, and the main content area shows the same service appointment table. The developer tools Network tab also shows a single request to 'localhost:3000/serwisy' with the same JSON response as in the previous screenshot.

Rysunek 2.4.4 Odpowiedź serwera wraz z wizualizacją w warstwie prezentacji.

W momencie, kiedy lista serwisów jest dostępna pracownik kliką w wybrane zadanie następnie otwiera się specjalne okno modalne, aby zmienić status.

- Pracownik po stronie warstwy prezentacji kliką przycisk zmiany statusu o nazwie „Zostało 15 min”.

```
<Button
    onClick={setStatusByMethod}
    name='Zostało 15 min'
    variant='outline-warning'
>
const saveStatusChanges = async () => {
  if (status) {
    try {
      await putServiceStatusById(serviceId, status);
    ...
  }
}
```



Rysunek 2.4.5 Wysłanie żądania wraz z wybranym nowym statusem.

- Serwer dostaje wiadomość o nowym statusie. Pobiera z bazy ten serwis sprawdza jego aktywny status. Jeżeli aktywny status to „Zakończono” lub „Anulowano” to nie można zmienić statusu i zwraca błąd na warstwę prezentacji. W innym przypadku po pobraniu obiektu, aktualizuje go i zapisuje w bazie danych.

```

OrderService order = orderServiceRepository.findById(idService).orElseThrow(() -> new
RuntimeException("Brak serwisu"));
if (order.getStatus() == OrderServiceStatus.DONE || order.getStatus() ==
OrderServiceStatus.CANCEL)
    return ResponseEntity.badRequest().body(GetOrderServiceDto.build(order));

```

- 3) Baza danych aktualizuje wybraną pozycję.

```
orderServiceRepository.save(order);
```

- 4) Serwer sprawdza, czy wystąpił jakiś błąd, jeżeli nie to zwraca status 200 i zaktualizowane dane.

```

@PutMapping("{idService}")
@PreAuthorize("hasRole('EMPLOYEE')")
public ResponseEntity<GetOrderServiceDto>
changeServiceStatus(@PathVariable("idService") Long idService, @RequestBody String
status) {
    return orderServiceService.changeServiceStatus(idService, status);
}

```

- 5) Warstwa prezentacji wyświetla zaktualizowane dane w formie tabelki.

```
{serviceData.length > 0 &&
serviceData.map((service) => (
    ...

```

The screenshot shows a web application interface for 'CarClean'. At the top, there's a navigation bar with links like 'Skip', 'Oferta', 'Kontakt', 'Opinie', and 'Przegląd serwisów'. Below the navigation is a calendar for May 2020, with specific dates highlighted in yellow (11, 12, 13) and blue (14, 15, 16, 17). Below the calendar is a table with columns: 'Marka', 'Numer rejestracyjny', 'Serwis', 'Data', 'Godzina', 'Status', and 'Płatność'. One row shows 'BMW' with license plate 'e46' and service 'Regeneracja kloszy samochodowych' scheduled for '2020-05-13' at '8:00' with status 'Zakończono' and payment status 'Nie zapłacono'.

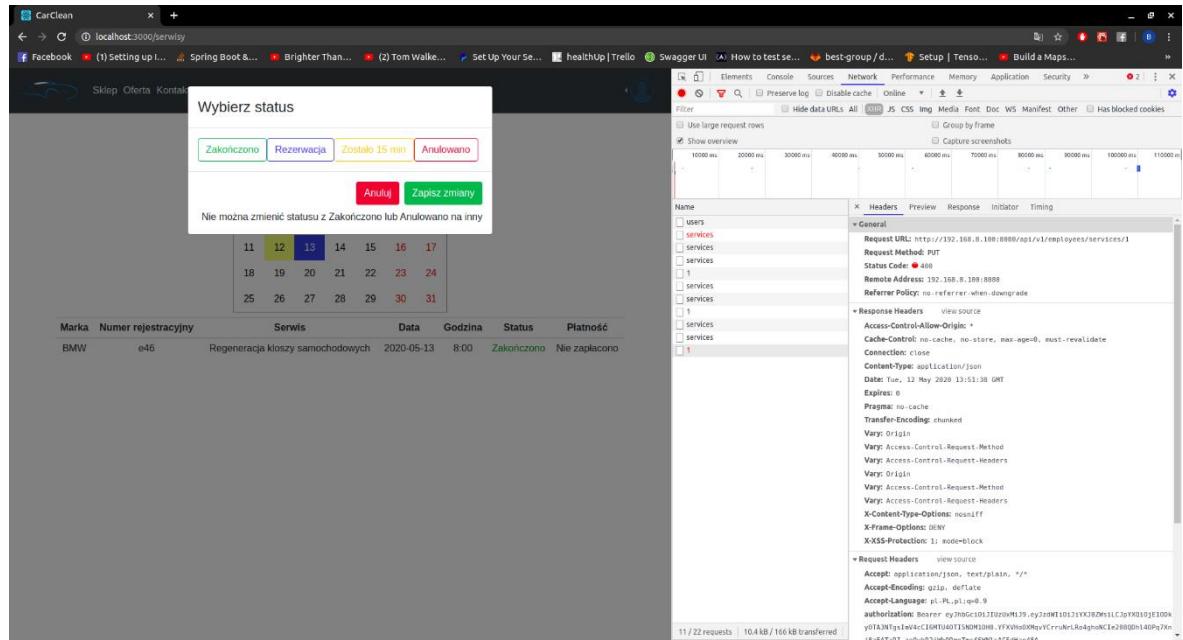
On the right side of the screen, the browser's developer tools are open, specifically the Network tab. It shows a list of requests. One request is expanded, showing its details:

- Name:** services
- Headers:** (empty)
- Preview:** (empty)
- Response:**

```

{
  "id": 1,
  "carId": 1,
  "brand": "BMW",
  "plateNumber": "e46",
  "date": "2020-05-13",
  "status": "Zakończono"
}
```
- Initiator:** (empty)
- Timing:** 8ms

Rysunek 2.4.6 Odpowiedź serwera na zmianę statusu.



Rysunek 2.4.7 Błąd 400 podczas niewłaściwej akcji pracownika.

2.5. Dodawanie produktów do koszyka przez zalogowanego użytkownika

- Klient wysyła żądanie o pobranie produktów z serwera.

```
componentDidMount = async () => {
  try {
    const result = await getAllProducts();
    ...
  };
  ...
  export const getAllProducts = () => get(shopApiUrl());
```

- Serwer otrzymuje i obsługuje żądanie, wysyła zapytanie do bazy danych celem pobrania produktów.

```
@GetMapping
public List<ProductDto> getAllProducts() {
  ...
}

@Override
public List<ProductDto> getAllProducts() {
  return productRepository.findAll().stream().map(product
    ->
  ProductDto.build(product)).collect(Collectors.toList());
}
```

- 3) Baza danych zwraca na serwer produkty i ich własności.

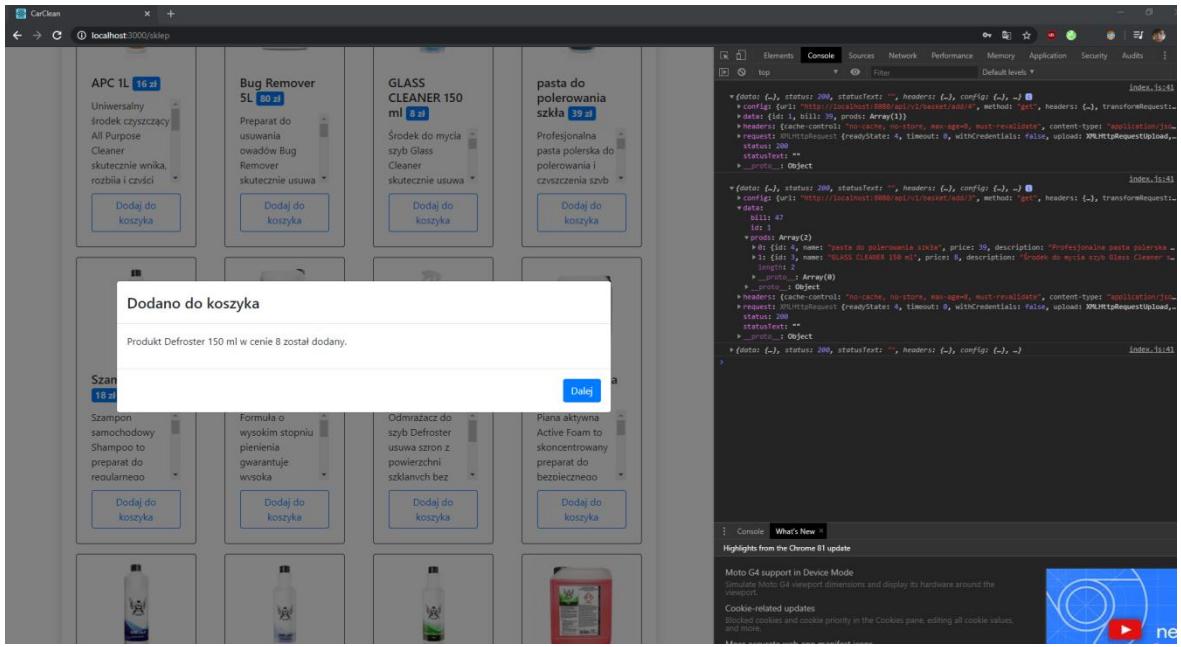
```
@Repository  
public interface ProductRepository extends JpaRepository<Product, Integer> {  
    ...  
    List<Product> findAllByCategory(ProductCategory category);  
    ...  
}
```

- 4) Serwer zwraca do warstwy prezentacji dane (listę produktów).

```
@GetMapping  
public List<ProductDto> getAllProducts() {  
    return productService.getAllProducts();  
}
```

- 5) Warstwa prezentacji wyświetla dane.

```
componentDidMount = async () => {  
    ...  
    this.setState({ products: result });  
}  
};  
  
<Container className='shadow my-3'>  
    <Row xs={1} sm={2} md={3} lg={4}>  
        {products.map((product) => (  
            <ProductCard  
                ...  
            />  
        ))}  
    </Row>  
</Container>
```



Rysunek 2.5.1 Odpowiedź serwera na dodanie produktów do koszyka

- 1) Użytkownik wybiera z listy żądzany produkt. Klikając przycisk "Dodaj do koszyka" co powoduje wysłanie żądania GET na serwer.

```

<Button onClick={addToBasket} variant='outline-primary'>
  Dodaj do koszyka
</Button>

const addToBasket = async () => {
  ...
  await addToUserBasket(id)
  ...
};
  
```

- 2) W serwerze dla każdego nowo utworzonego użytkownika tworzony jest koszyk. Koszyk ten jest modyfikowany po każdym dodaniu lub usunięciu produktu. Jest on też czyszczony po utworzeniu zamówienia. W tej sytuacji do listy z produktami dodawany jest nowy produkt i aktualizowana jest suma. Następnie serwer wysyła do bazy danych informacje o stanie koszyka.

```

@GetMapping("add/{productId}")
@PreAuthorize("hasRole('USER')")
public ResponseEntity<BasketDto> addProductToBasket(Authentication authentication,
@PathVariable("productId") int productId) {
  ...
}
  
```

```

int newPrice = basket.getBill() + product.getPrice();
basket.setBill(newPrice);
basketRepository.save(basket);

```

- 3) W bazie danych przypisywany jest ID produktu dla ID koszyka. Baza danych zwraca wyniki.

```

public interface BasketRepository extends JpaRepository<Basket, Integer> {

    Optional<Basket> findByUserUsername(String username);
}

```

- 4) Serwer sprawdza wynik. Przekazuje listę z zamówionymi dotychczas produktami.

```

public ResponseEntity<BasketDto> addProductToBasket(String username, int productId) {
    ...
    return ResponseEntity.ok(BasketDto.build(basket));
}

@GetMapping("add/{productId}")
@PreAuthorize("hasRole('USER')")
public ResponseEntity<BasketDto> addProductToBasket(Authentication authentication,
@PathVariable("productId") int productId) {
    return basketService.addProductToBasket(AuthMiner.getUsername(authentication),
    productId);
}

```

- 5) Warstwa prezentacji informuje o powodzeniu wyświetlając okno modalne.

```

const MyVerticallyCenteredModal = ({ name, price, onHide, show }) => (
    ...
        <Modal.Title>Dodano do koszyka</Modal.Title>
        ...
        Produkt {name} w cenie {price} został dodany.
        ...
        <Button onClick={onHide}>Dalej</Button>
        ...
    );

```

3. WNIOSKI

Celem sprawdzenia jakości działania kodu wykonane zostały testy jednostkowe i integracyjne przy wykorzystaniu MockMVC oraz Junit Jupiter. Pokrycie kodu zostało sprawdzone za pomocą Coverage w IntelliJ Idea Ultimate. Modele i klasy dto nie podlegały pod priorytet testów. Łączne pokrycie linii obejmuje 78 % (z modelami i dto). Natomiast poniżej przedstawiamy najistotniejsze wyniki poszczególnych częściach kodu:

- kontrolery - 96 %,
- konfiguracja – 82 %,
- serwisy – 79 %.

The screenshot displays the IntelliJ IDEA interface with the Coverage tool window open. The coverage report indicates 100% classes and 78% lines covered across various packages. The main editor shows a Java file for the `AuthMiner` class. Below the editor, the Test Runner tool window shows a list of tests that have passed, with a total execution time of 16 seconds and 712 milliseconds.

Rysunek 3.0.1 Pokrycie kodu.

This screenshot shows a detailed view of the Coverage tool window in IntelliJ IDEA. It lists the percentage of classes, methods, and lines covered for various package components. The table highlights several components with 100% coverage: controllers (12/12), dbresources (4/4), dto (17/17), models (25/25), repositories (0/0), services (11/11), and utilities (1/1). The overall summary at the top states 100% classes and 78% lines covered in the package 'com.carwash.server'.

Coverage: All in server			
Element	Class, %	Method, %	Line, %
configurat...	100% (7/7)	94% (16/17)	82% (69/84)
controllers	100% (12/12)	96% (53/55)	96% (56/58)
dbresources	100% (4/4)	100% (6/6)	78% (22/28)
dto	100% (17/17)	94% (107/113)	94% (161/171)
models	100% (25/25)	69% (130/187)	62% (174/278)
repositories	100% (0/0)	100% (0/0)	100% (0/0)
services	100% (11/11)	86% (64/74)	79% (369/466)
utilities	100% (1/1)	100% (1/1)	100% (1/1)
ServerApp...	100% (1/1)	0% (0/0)	33% (1/3)

Rysunek 3.0.2 Pokrycie kodu w pakietach.

WYKAZ RYSUNKÓW

Rysunek 1.1.1 Strona główna aplikacji.	3
Rysunek 1.1.2 Panel rejestracji.....	4
Rysunek 1.1.3 Wypełniony panel rejestracji.	4
Rysunek 1.2.1 Panel logowania użytkownika.	5
Rysunek 1.3.1 Strona główna wraz z panelem użytkownika.	5
Rysunek 1.3.2 Strona pojazdów użytkownika.....	6
Rysunek 1.3.3 Informacja potwierdzająca dodanie pojazdu.	6
Rysunek 1.4.1 Komunikat o błędzie podczas usuwania pojazdu.	7
Rysunek 1.5.1 Strona z wyróżnieniem dostępnych usług na stronie.....	8
Rysunek 1.5.2 Rejestracja na usługę przed wyborem pojazdu.....	8
Rysunek 1.5.3 Rejestracja na usługę po wyborze pojazdu, daty i godziny.	9
Rysunek 1.6.1 Historia serwisów danego użytkownika.	9
Rysunek 1.7.1 Strona pośrednicząca pomiędzy płatnością.....	10
Rysunek 1.7.2 Rachunek w PayPal.	10
Rysunek 1.7.3 Oczekiwanie na akceptacje systemu.....	11
Rysunek 1.7.4 Akceptacja płatności.	11
Rysunek 1.7.5 Historia serwisów wraz z zapłaconą usługą.	12
Rysunek 1.8.1 Wysyłanie wiadomości do administracji.	12
Rysunek 1.8.2 Potwierdzenie wysłania wiadomości.	13
Rysunek 1.9.1 Panel dodawania opinii.	13
Rysunek 1.9.2. Okno dialogowe wyboru fotografii do opinii.	14
Rysunek 1.9.3 Kompletny uzupełniony formularz.....	15
Rysunek 1.9.4 Przedstawienie dodanej opinii	15
Rysunek 1.10.1 Panel pracowniczy.	16
Rysunek 1.10.2 Okno modalne zmiany statusu zamówienia	16
Rysunek 1.10.3 Potwierdzenie zmiany statusu.....	17
Rysunek 1.10.4 Informacja o błędzie podczas zmiany statusu na błędny.....	17
Rysunek 1.11.1 Strona 404 wyświetlająca nieprawidłowy adres URL.....	18
Rysunek 1.12.1 Sklep z produktami	18
Rysunek 1.12.2 Widok koszyka	19
Rysunek 1.13.1 Koszyk.	20

Rysunek 1.14.1 Historia zamówień	20
Rysunek 1.14.2 Szczegóły zamówienia.....	21
Rysunek 1.15.1 Okno szczegółów płatności	21
Rysunek 1.15.2 Realizacja płatności w PayPal	22
Rysunek 1.15.3 Koniec płatności	22
Rysunek 1.16.1 Okno szczegółów płatności	23
Rysunek 2.1.1 Odpowiedź serwera z dostępnymi ofertami	25
Rysunek 2.1.2 Odpowiedź zawierająca listę pojazdów użytkownika.	27
Rysunek 2.1.3 Odpowiedź zawierająca wybrany pojazd użytkownika.....	28
Rysunek 2.1.4 Zapytanie zawierające wybraną datę.	28
Rysunek 2.1.5 Odpowiedź w formie listy ukazująca dostępne godziny wybranego dnia.	30
Rysunek 2.1.6 Wysłanie żądania o rezerwacji.	30
Rysunek 2.1.7 Odpowiedź serwera na rezerwacje wykonaną przez użytkownika..	32
Rysunek 2.2.1 Zwrócona pusta lista opinii i okno modalne do dodania nowej.	33
Rysunek 2.2.2 Odpowiedź zamkająca okno modalne, informującą o pozytywnym dodaniu opinii.	35
Rysunek 2.2.3 Odpowiedź zawierającą listę wszystkich opinii.	36
Rysunek 2.3.1 Odpowiedź uzyskana poprzez wysłanie tokenu na serwer. Odpowiedź zawiera dane o użytkowniku.	38
Rysunek 2.4.1 Odpowiedź z błędem 404 na żądanie z datą serwisów.....	41
Rysunek 2.4.2 Treść błędu 404 w przypadku pobrania serwisów danego dnia.	41
Rysunek 2.4.3 Przykład żądania z wybraną przez użytkownika datą.	42
Rysunek 2.4.4 Odpowiedź serwera wraz z wizualizacją w warstwie prezentacji. ..	42
Rysunek 2.4.5 Wysłanie żądania wraz z wybranym nowym statusem.	43
Rysunek 2.4.6 Odpowiedź serwera na zmianę statusu.	44
Rysunek 2.4.7 Błąd 400 podczas niewłaściwej akcji pracownika.....	45
Rysunek 2.5.1 Odpowiedź serwera na dodanie produktów do koszyka.....	47
Rysunek 3.0.1 Pokrycie kodu.	49
Rysunek 3.0.2 Pokrycie kodu w pakietach.	49