|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Państwowa Wyższa Szkoła Informatyki i Przedsiębiorczości**  **w Łomży**  **Wydział Informatyki i Nauk O Żywności** | | | | | |
| **Kierunek:**  Informatyka | **Przedmiot:**  Projekt zespołowy II | **Rok:**  III | **Semestr:**    6 | **Prowadzący:**  Janusz Rafałko | **Data:**  29.05.2021 |
| **Skład zespołu:**  Sebastian Brodziak  Krzysztof Jaśkowski  Sebastian Kucharczyk | | | | | |

Projekt Systemu Zarządzania

Parkingiem Strzeżonym „Carceres”

Wstęp

Ogólna charakterystyka

System zarządzania parkingiem strzeżonym „Carceres” ułatwiający pracę stróża parkingowego oraz obsługę klienta. Zawiera panel logowania, podgląd mapy parkingu, zakładanie użytkowników z podziałem na role, możliwość rejestracji pojazdów, możliwość generowania paragonów/faktur, obsługę płatności oraz cennik. Wszystkie wymienione funkcje dostępne są z poziomu zarówno wersji web jak i desktop. Wersja mobile przeznaczona jest dla klienta końcowego i służy do monitorowania stanu rezerwacji, daje możliwość wykupu nowego miejsca. Cechą charakterystyczną „Carceres” jest jego lekkość oraz uniwersalność – uruchomić go można niezależnie od posiadanej platformy sprzętowej jak i posiadanego systemu operacyjnego.

Przyczyny wyboru tematu

Temat okazał się ciekawym wyzwaniem dzięki hobbystycznemu podejściu do tematu oprogramowania do zarządzania flota samochodów. Z racji zbliżonej tematyki nasza grupa postanowiła spróbować stworzyć własną wersję. Dodatkowym założeniem było stworzenie systemu prostego w wdrożeniu jak i obsłudze, skierowanego głównie do mniejszych parkingów strzeżonych, gdzie nie jest wymagane posiadanie „ciężkiego” i zaawansowanego systemu z rozwiązań konkurencyjnych.

Cel projektu

Celem projektu jest stworzenie systemu zarządzania parkingiem strzeżonym, ułatwiającego pracownikom parkingu codzienną pracę oraz nie używającego sporych zasobów sprzętowych. Kluczowa tu ma być mobilność systemu, umożliwiająca uruchomienie go bez względu na zastosowany sprzęt czy system operacyjny.

Przegląd istniejących rozwiązań

Na rynku istnieją rozwiązania tego typu. Charakteryzują się zróżnicowaną złożonością, możliwością konfiguracji, współpracą z peryferiami typu bramki, wpłatomaty, pętle indukcyjne – rozwiązania zaawansowane technologicznie i wymagające wyższych konfiguracji sprzętowych, szczególnie od strony backendowej.

PROPARK – <https://www.propark.pl/parkingi/parkingi-bezobslugowe-automatyczne/>

TAAB – <https://taab.pl/systemy-parkingowe-taab/>

DEJW – <https://www.dejw.eu/systemy-parkingowe/>

Zakres projektu

Opis biznesowy

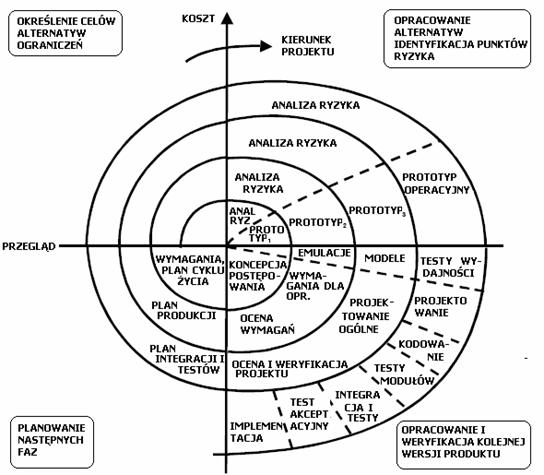
Carceres to oprogramowanie do zarządzania parkingiem strzeżonym, stworzone specjalnie dla małych parkingów prowadzonych przez mikro i małych przedsiębiorców, nie posiadających dużego budżetu na zakup i obsługę oprogramowania. Uniwersalność, prostota oraz lekkość – to cechy wyróżniające go na tle konkurencyjnych rozwiązań.

Wymagania funkcjonalne i niefunkcjonalne

|  |  |
| --- | --- |
| **WYMAGANIA FUNKCJONALNE** | **WYMAGANIA NIEFUNKCJONALNE** |
| tworzenie użytkowników systemu, nadawanie im uprawnień oraz logowanie | prosta, intuicyjna obsługa |
| ewidencja miejsc parkingowych (sprawdzanie ogólnej liczby, liczby miejsc wolnych/zajetych, strefy parkingowe) | Niskie wymagania sprzętowe |
| poglądowa mapa parkingu | Możliwość uruchomienia na każdym komputerze |
| rejestracja i ewidencja klientów | Możliwość uruchomienia na smartfonie klasy niższej i w górę |
| rejestracja i ewidencja pojazdów korzystających z parkingu |  |
| generowanie biletów/kart parkingowych (wjazd/wyjazd) |  |
| możliwość rejestracji online oraz płatności online |  |
| duplikaty biletów/kart |  |
| generowanie rachunków |  |
| kontrola płatności |  |
| przypominanie o końcu abonamentu mail/sms |  |
| generowanie raportów okresowych |  |
| generowanie grafików (tworzenie, ewidencja pracowników, urlopy, drukowanie) |  |

Metodyka

Metodyka pracy

Model spiralny – model opracowany w 1988 roku. Każde okrążenie spirali reprezentuje planowanie i wytworzenie pewnego elementu produktu projektu. Cały projekt składa się z szeregu obrotów po rozkręcającej się spirali.  


Rys. <https://www.cri.agh.edu.pl/uczelnia/tad/PSI6/wyklady_html/wyklad12_pliki/image008.jpg>

Środki implementacji

Do tworzenia oprogramowania zastosowano następujące technologie:

Backend:

* Python + framework Flask
* MariaDB do obsługi bazy danych

Frontend wersji web:

* Node.js
* Vue
* Bootstrap

Docker do łatwej implementacji na różnych maszynach

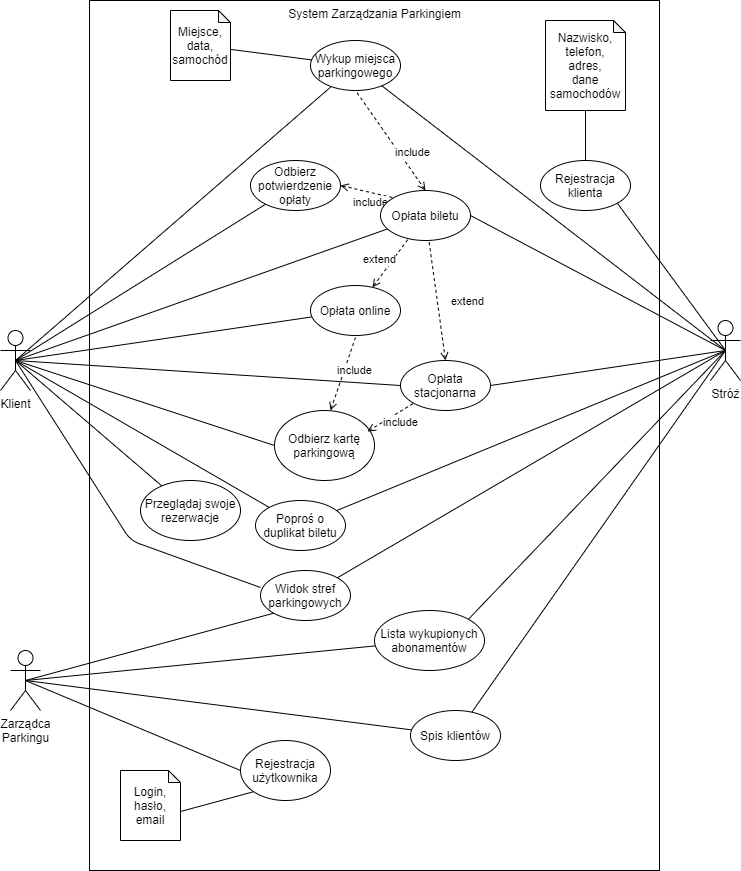
Wersja desktop:

* Electron

Wersja mobile:

* Xamarin Forms

Przypadki użycia

****

**Aktor – Klient:**

Jestem klientem. Chciałbym móc wykupić miejsce na parkingu online lub na miejscu, z możliwością wcześniejszego wybrania go na mapce. Chciałbym mieć możliwość wyboru czy zapłacę online czy gotówką na miejscu, a także na jaki czas chciałbym miejsce: godzinowo, 1 dzień, kilka dni, tydzień, miesiąc, rok. Gdy będzie kończył mi się abonament chciałbym dostać powiadomienie. Po wykupieniu online chciałbym dostać potwierdzenie zakupu oraz bilet parkingowy w formie PDF. Na miejscu potwierdzenie chciałbym otrzymać wydrukowane. Gdybym zgubił bilet to chciałbym móc poprosić o duplikat.

**Aktor – Stróż:**

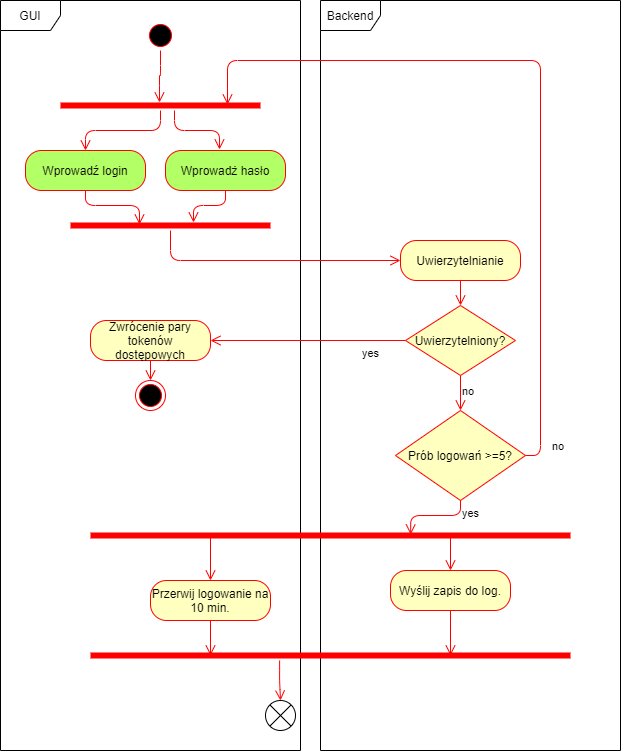
Jestem stróżem. Jako stróż muszę widzieć ile miejsc mam w całości, ile zajętych i ile wolnych. Muszę widzieć które miejsce należy do kogo. Nowy klient musi zostać zarejestrowany, zarejestrowany samochód oraz przypisane miejsce parkingowe i czas abonamentu. Po zarejestrowaniu klient otrzymuje potwierdzenie zakupu i bilet uprawniający do wjazdu i wyjazdu. W momencie wjazdu powinienem widzieć czy klient ma opłacony abonament. Gdy klient zgubi bilet, mogę mu wygenerować duplikat.

**Aktor – Zarządca parkingu:**

Jestem zarządcą parkingu. Chciałbym mieć wgląd na strefy parkingowe oraz widzieć ile miejsc mam zajętych a ile dostępnych. Chciałbym mieć możliwość wglądu do spisu klientów oraz samochodów, wglądu do wykupionych abonamentów i wiedzieć czy każdy zapłacił na czas. Chciałbym też mieć możliwość sporządzenia raportów: dziennych, tygodniowych, miesięcznych, kwartalnych, rocznych, zarówno jeżeli chodzi o statystykę miejsc jak i kasowe oraz pojazdów.

Diagramy czynności

Diagram aktywności – Logowanie



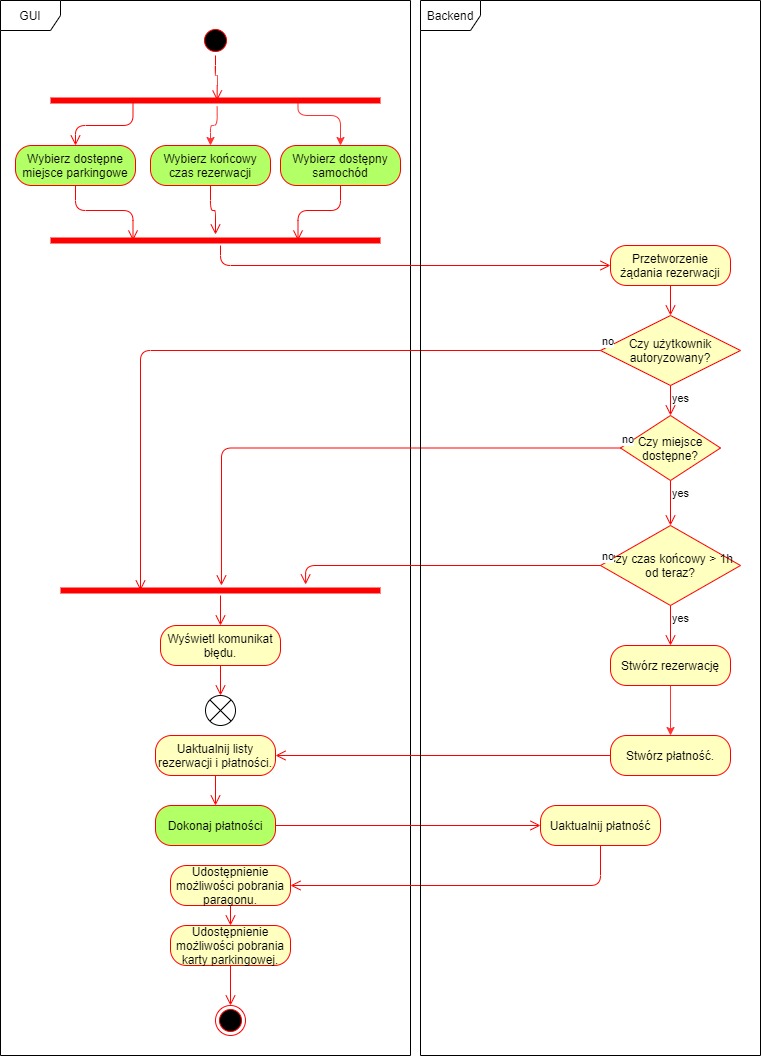
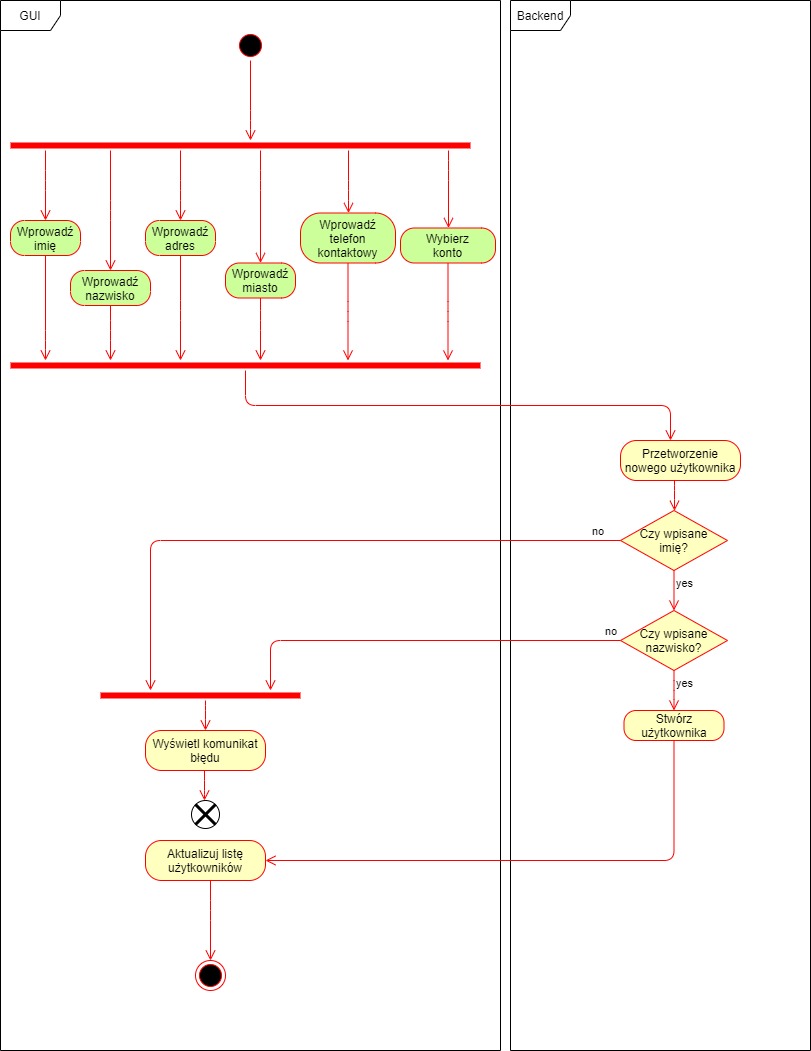
Diagram aktywności – Rezerwacja miejsca parkingowego

Diagram aktywności – Dodanie nowego klienta



Diagramy interakcji

Diagram interakcji – Logowanie

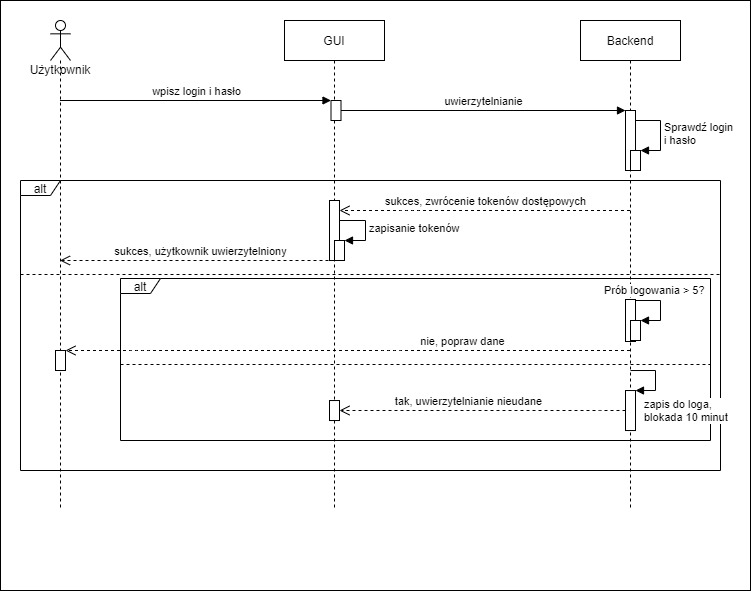


Diagram interakcji – Rezerwacja miejsca parkingowego

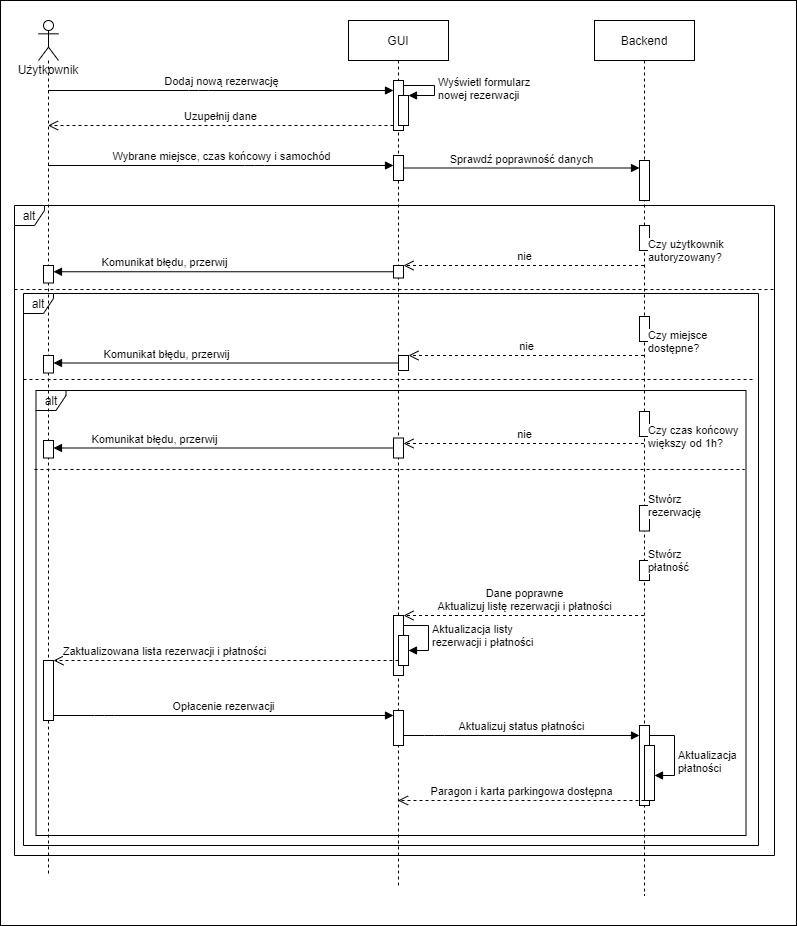
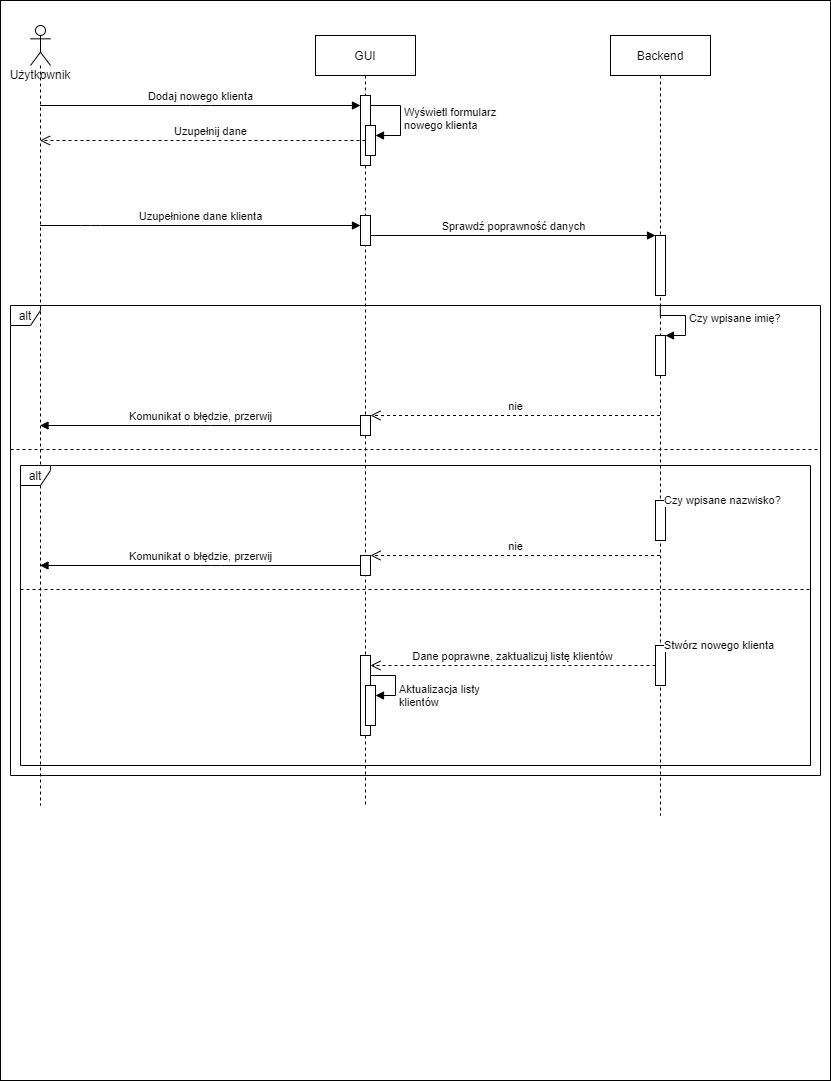
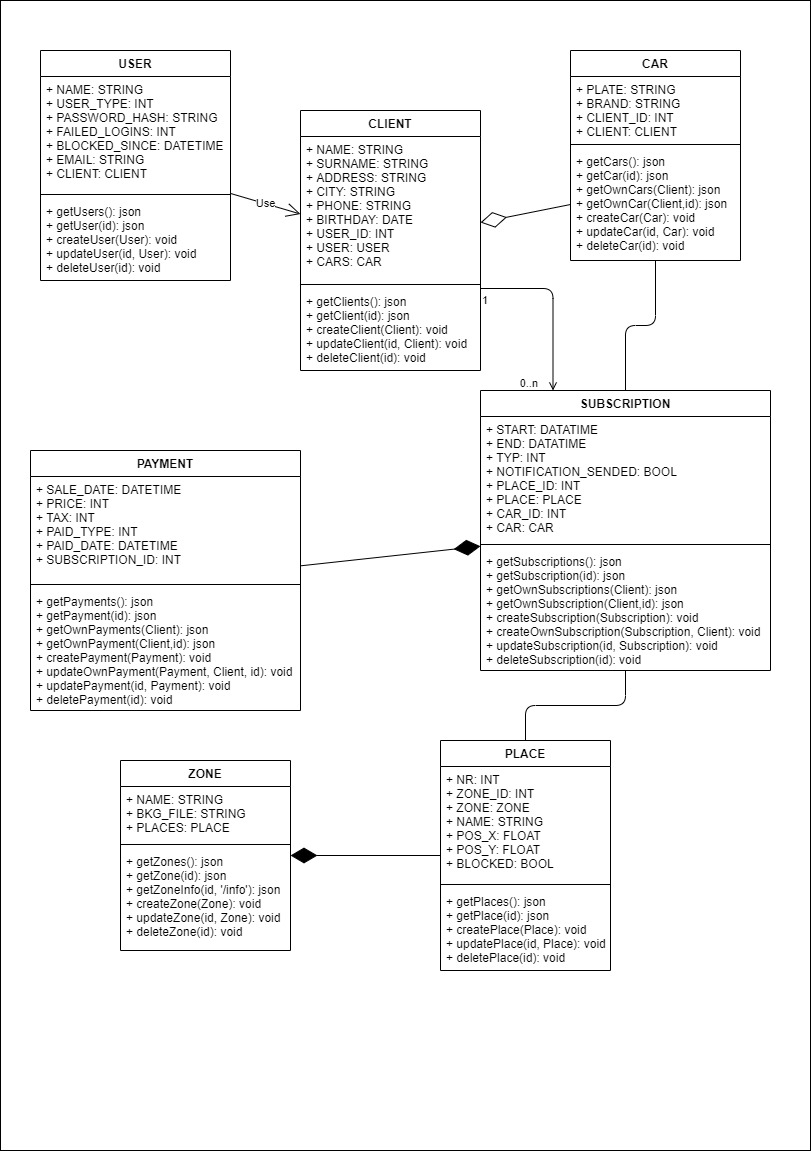


Diagram interakcji – Dodanie nowego klienta



Perspektywa przypadków użycia

Diagram klas



### Diagramy stanów

Diagram stanu – Autoryzacja użytkownika

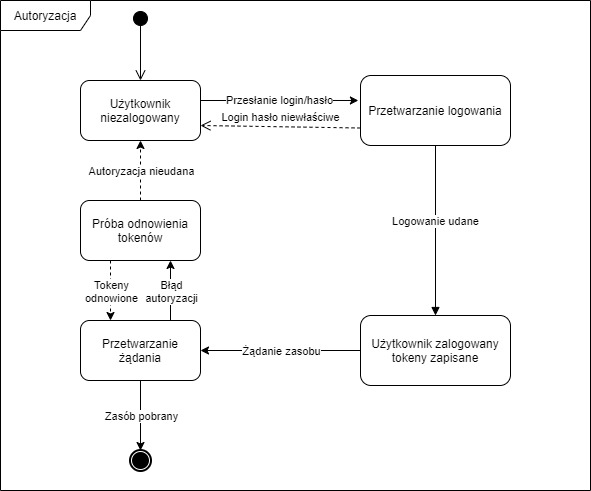
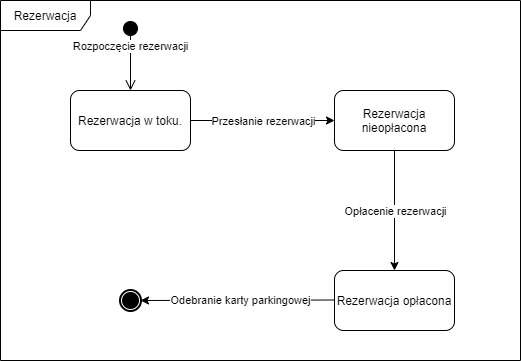


Diagram stanu – Rezerwacja miejsca parkingowego



Baza danych

Schemat bazy danych w aplikacji Carceres. Baza składa się z 7 tabel.

# 

Diagramy wdrożenia

Diagram wdrożenia z podziałem na wiele maszyn

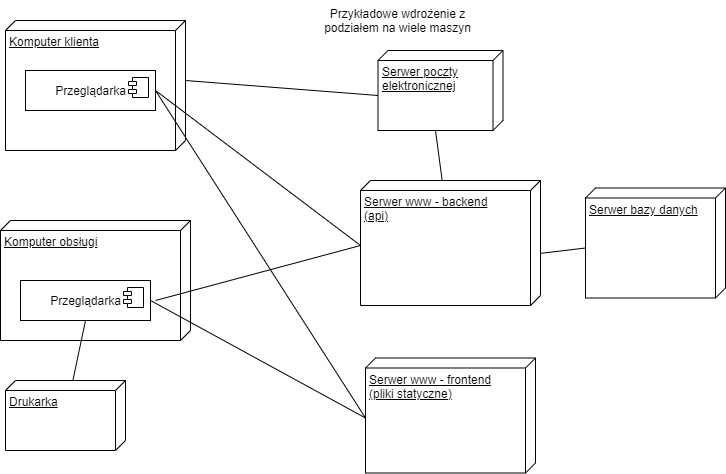
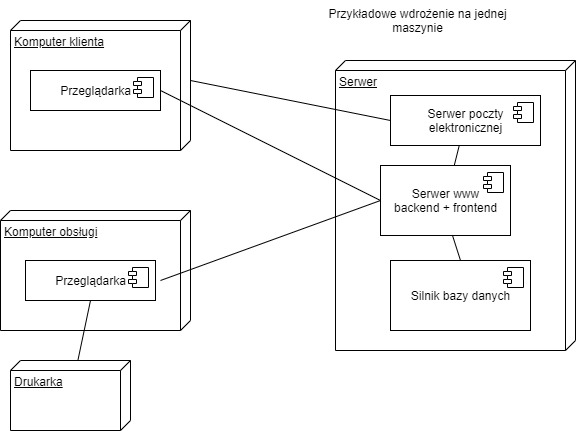


Diagram wdrożenia na jednej maszynie



Członkowie zespołu i harmonogram

Odpowiedzialność poszczególnych członków zespołu:

Sebastian Brodziak – Project Manager, GUI

Krzysztof Jaśkowski – projekt dokumentacji, mobile, testy

Sebastian Kucharczyk – desktop, mobile

Przewidywany harmonogram prac:

* 31.03 – wykonanie mockupów (szkiców) GUI
* 10.04 – testy TDD
* 17.04 – działające GUI
* 24.04 – wersje 1.0 mobile/desktop
* Do 29.05 – dopracowanie ewentualnych dodatkowych błędów i wykonanie dokumentacji

Model systemu

Architektura

# Cały system działa w architekturze klient-serwer. Serwer odpowiada przede wszystkim za dostarczenie API potrzebnego do komunikacji pomiędzy poszczególnymi aplikacjami (web/mobile - serwer) oraz przechowywanie bazy danych.

# Po stronie klienta wykonywane są poniższe możliwości dostępu do aplikacji:

# Dostęp za pomocą przeglądarki internetowej – logika napisana w node.js oznacza wykonywanie kodu po stronie serwera i przesyłanie gotowych wyników do klienta (eliminuje obciążenie hosta użytkownika)

# Dostęp za pomocą wieloplatformowej aplikacji desktopowej – aplikacja korzysta z Electrona. Jest to proste rozwiązanie, wykorzystujące Node.js i Chromium, w którym tak naprawdę mamy wersję web aplikacji zamkniętą w zewnętrznym programie. Aplikacja wymaga dostępu do internetu ponieważ musi skomunikować się z serwerem w celu pobrania danych z bazy danych. Wersja desktopowa działa zarówno na Windowsie, jak i Linuxie oraz MacOS.

# Aplikacja mobile - aplikacja działająca na systemie Android. Napisana w C# w technologii Xamarin Forms. Wymagany jest dostęp do internetu w celu komunikacji z serwerem danych.

Opis modułów

W Carceres można wyróżnić cztery moduły:

Ewidencja klientów – służy do wprowadzania danych klientów, chcących korzystać z parkingu.

Ewidencja pojazdów – zawiera listę pojazdów klientów

Ewidencja parkingu – zawiera listę miejsc parkingowych, listę stref oraz mapę parkingu

Płatności – lista należności klientów, które miejsce zostało opłacone, które jeszcze nie i czy płatność jest przeterminowana. Zawierać może także integrację z systemem płatności internetowych lub integrację z systemami płatności kartą na miejscu.

Nie są one niezależne, gdyż razem tworzą cały system i są wzajemne powiązane.

Endpoint’y

Najbardziej aktualne dane routowania powinny być w tym pliku: https://github.com/Delor4/CarceresBE/blob/master/app.py

Metody dostępne przy endpoint'cie (jeżeli nie zaznaczono inaczej):

GET, GET /<id> dane wszystkich/pojedyńczego obiektu

POST dodanie obiektu

PUT /<id> uaktualnienie danych obiektu

DELETE /<id> usunięcie obiektu

HEAD, HEAD /<id> informacje o endpoint'cie

/api/users [użytkownicy systemu](https://github.com/Delor4/CarceresBE/blob/master/resources/users.py)

Pola:

'id': Integer,

'name': String, UQ login

'user\_type': Integer, 1 - admin, 2 - mod, 3 - klient, 4 - inne

'email': String,

'password': String WO

'failed\_logins' Integer, RO, aktualna liczba nieudanych logowań

'blocked\_since' DateTime, RO, do kiedy zablokowane logowanie (null jeżeli nie było blokad)

'client': (client\_fields),

'uri': Url RO

----

/api/clients [klienci](https://github.com/Delor4/CarceresBE/blob/master/resources/clients.py)

Pola:

'id': Integer,

'name': String, lastname? Imię,

'surname': String,

'address': String,

'city': String,

'phone': String,

'birthday': Date,

'cars': (car\_fields), RO

'user\_id' Integer,

'user' (user\_fields) , RO

'uri': Url, RO

---

/api/cars [samochody](https://github.com/Delor4/CarceresBE/blob/master/resources/cars.py)

Pola:

'id': Integer,

'plate': String,

'brand': String,

'client\_id': Integer,

'client': (client\_fields),

'uri': Url, RO

----

/api/zones [strefy parkowania](https://github.com/Delor4/CarceresBE/blob/master/resources/zones.py)

Pola:

'id': Integer,

'name': String,

'bkg\_file': String,

'places':place\_fields, RO

'uri': Url, RO

---

/api/places [miejsca parkowania](https://github.com/Delor4/CarceresBE/blob/master/resources/places.py)

Pola:

'id': Integer,

'nr': Integer, NN

'zone\_id': Integer, NN

'name': String,

'pos\_x': Float,

'pos\_y': Float,

'occupied' Boolean, RO

'uri': Url,RO

---

/api/subscriptions [subskrypcje](https://github.com/Delor4/CarceresBE/blob/master/resources/subscriptions.py)

Pola:

'id': Integer,

'start': DateTime,

'end': DateTime,

'type': Integer,

'place\_id': Integer,

'place':place\_fields, RO

'car\_id': Integer,

'car': (car\_fields), RO

'payment': (payment\_fields), RO

'uri': Url,RO

---

/api/payments [płatności](https://github.com/Delor4/CarceresBE/blob/payments/resources/payments.py)

Pola:

'id': Integer,

'price': Integer, cena netto (w groszach)

'tax': Integer,podatek

'value': Integer, RO, cena brutto (w groszach)

'sale\_date': DateTime, data sprzedaży

'paid\_type': Integer, rodzaj płatności (0-brak, 1-na miejscu, 2-online)

'paid\_date': DateTime, data płatności

'paid': Boolean, RO

'subscription\_id': Integer,

'uri': Url,RO

---

/api/user [dane użytkownika](https://github.com/Delor4/CarceresBE/blob/master/resources/manage.py)

Dozwolone metody: GET, PUT

---

/api/client [dane klienta](https://github.com/Delor4/CarceresBE/blob/master/resources/manage.py)

Dozwolone metody: GET, PUT

---

/api/client/subscriptions [dane rezerwacji klienta](https://github.com/Delor4/CarceresBE/blob/master/resources/subscriptions.py)

Dozwolone metody: GET, POST

---

/api/client/payments [płatności klienta](https://github.com/Delor4/CarceresBE/blob/payments/resources/payments.py)

Dozwolone metody: GET, GET /<id>, PUT /<id>

(po wywołaniu PUT /<id> status zostaje ustawiony na zapłacono online z aktualną datą)

---

/api/client/cars/ samochody klienta

Dozwolone metody: GET, GET /<id>

---

/api/zones/<id>/info statystyki strefy parkowania

Dozwolone metody: GET /<id>

Pola:

'all': Integer, RO, całkowita ilość miejsc parkingowych w strefie

'free': Integer, RO, wolne miejsca

'occupied': Integer, RO, zajęte miejsca

'zone\_id': Integer, RO

RO - Read Only

WO - Write Only

UQ - Unique

NN - Not Null

Zainicjowana baza danych posiada jednego użytkownika z prawami administratora: admin hasło: carceres.

W trybie debug (nie zadziała na serwerze produkcyjnym) wysłanie żądania GET pod /api/seed doda do bazy danych przykładowe dane.

Backend

Backend przyjmuje połączenia do "/api/login" z nazwą użytkownika i hasłem (basic auth). Jeżeli dane się zgadzają z zapisanymi to zwraca dynamicznie wygenerowane "access\_token" i "refresh\_token" (tokeny nie są zapisywane w bazie danych).

W przypadku gdy użytkownik 5 razy z rzędu poda niewłaściwe hasło możliwość uzyskania tokenów zostaje zablokowana na 10 minut. (wartości do ustawienia w pliku konfiguracyjnym: "AUTOBLOCKADE\_ATTEMPTS", "AUTOBLOCKADE\_TIME")

Przy błędach uwierzytelniania serwer zwraca response code "401 UNAUTHORIZED" (sam nagłówek z pustą zawartością).

Do uzyskania nowych tokenów (access i refresh) można użyć "POST /api/refresh" wysyłając w treści "refresh\_token", np:

W tokenach są zapisane (m. in.) nazwa użytkownika i 'czas życia' tokenu. (Czas do ustawienia globalnie w pliku konfiguracyjnym, domyślnie w "settings.py". "SECRET\_ACCESS\_KEY\_EXPIRATION" oraz "SECRET\_REFRESH\_KEY\_EXPIRATION". Teraz 1h ale docelowo 10 min.)

Przy połączeniach wymagających autoryzacji trzeba dodawać access\_token w specjalnym nagłówku ("x-access-tokens").

Serwer sprawdza wtedy:

- czy token jest obecny

- czy nie ma błędów w tokenie

- czy token nie jest przedawniony

Przy powyższych błędach zwraca response code „401 UNAUTHORIZED” z komunikatem błędu.

Czasami serwer sprawdza też prawa dostępu do danego połączenia (prawa są zależne od typu użytkownika).

Przy tym błędzie zwraca response code „403 FORBIDDEN”.

STRONICOWANIE WYNIKÓW

Do zapytań zwracających listy (endpointy GET /api/<model>s ) można dodawać argumenty ograniczające ilość wyników (parametry zapytania GET).

- start => wyznacza startowy element listy (domyślnie 1)

- limit => wyznacza maksymalną ilość zwróconych elementów (domyślnie wartość config[DEFAULT\_PAGE\_LIMIT] lub 25)

Przykład:

/api/clients?start=21&limit=10

Zwracane wartości to:

- start: startowy element zwróconej listy

- limit: określony limit

- count: całkowitą ilość wyników

- results: właściwa lista wyników

- previous: url do poprzedniej strony z wynikami (puste jeżeli brak poprzedniej)

- next: url do następnej strony z wynikami (puste jeżeli brak następnej)

Przykład:

/api/places?start=109&limit=20

{

"count": 110,

"limit": 20,

"next": "",

"previous": "http://localhost:43343/api/places?start=89&limit=20",

"results": [

{

"id": 109,

"name": null,

"nr": 51,

"pos\_x": 38.0,

"pos\_y": 82.0,

"uri": "http://localhost:43343/api/places/109",

"zone\_id": 2

},

{

"id": 110,

"name": null,

"nr": 52,

"pos\_x": 41.0,

"pos\_y": 86.0,

"uri": "http://localhost:43343/api/places/110",

"zone\_id": 2

}

],

"start": 109

}

Gdy parametry wyjdą poza dozwolony zakres zostanie zwrócony kod 404 NOT FOUND i odpowiedni komunikat.

Np:

{

"message": "Pagination start outside allowed values. Expected: 1 - 110. Provided: 116"

}

{

"message": "Pagination limit outside allowed values. Expected more than 0. Provided: -1"

}

---

[Sortowanie wyników](#sortowanie):

---

Dodając parametr sort\_by można sortować po polach poszczególnych tabel.

Określanie kierunku sortowania: desc(pole), asc(pole).

Sortując po wielu polach nazwy oddzielamy przecinkiem.

Przykład:

/api/places?sort\_by=asc(zone),desc(pos\_y),nr

HEADERY

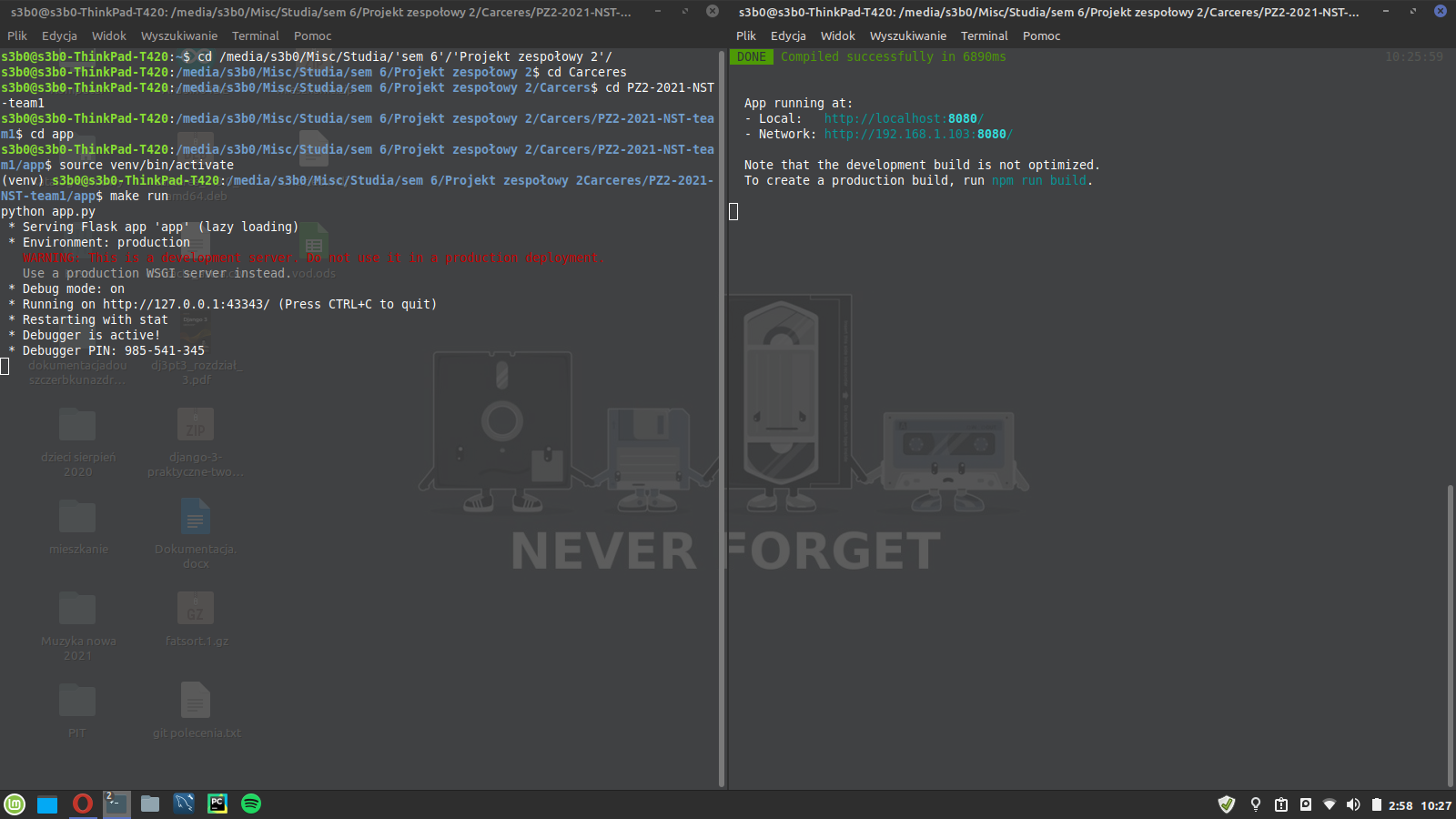
Przy endpointach które zmieniają stan bazy (metody POST, PUT, DELETE) w response ustawiam headery wyłączające cache w przeglądarce.

Przy endpointach wyświetlających pojedyńczy model ustawiam Last-Modified. Można go użyć do sprawdzenia czy dane się zmieniły i czy trzeba ściągać je ponownie (np. wysyłając zapytanie HEAD przed właściwym GET). Przeglądarki robią to automatycznie do sprawdzania czy zwracać dane z cache (ale można to ich zachowanie wyłączyć).

Użytkowanie

Instrukcja wdrożeniowa

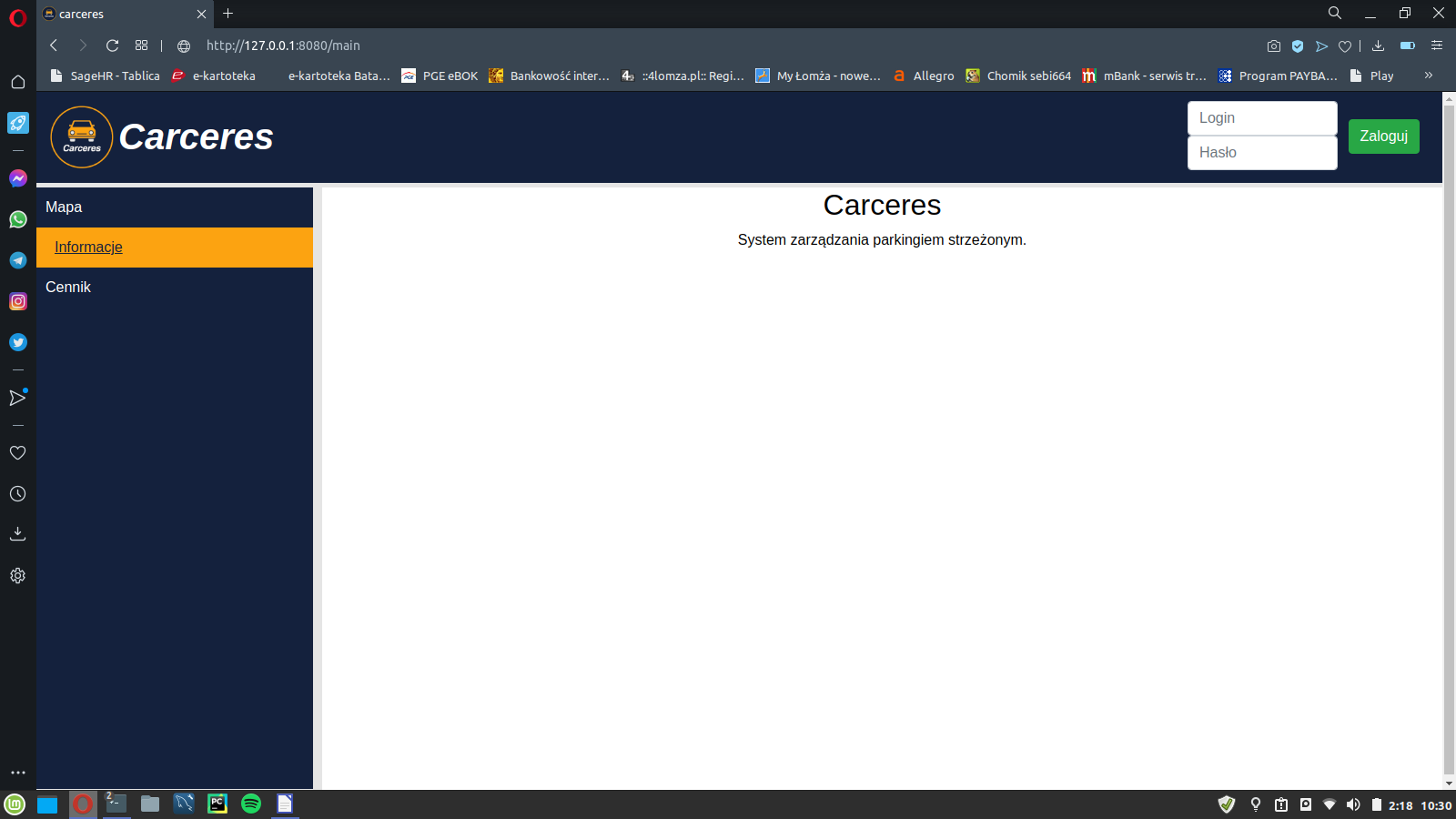
Wdrożenie aplikacji jest stosunkowo proste i ogranicza się jedynie do uruchomienia Dockera na serwerze i zaimplementowaniu do niego gotowego obrazu. Zastosowanie konteneryzacji przyspiesza proces, ponieważ całość odbywa się automatycznie i po uruchomieniu aplikacja jest w zasadzie gotowa do działania. Pozostaje jedynie kwestia konfiguracji domen (jeżeli jest taka potrzeba, a w zasadzie jest to wygodniejsze ponieważ w przypadku zastosowania domen unikamy problemu zmiany adresu IP serwera w przypadku zmiany) jednak jest to kwestia do indywidualnego ustalenia przez administratora.

Następnie należy uruchomić backend flaskowy i node.js w celu uruchomienia usług.

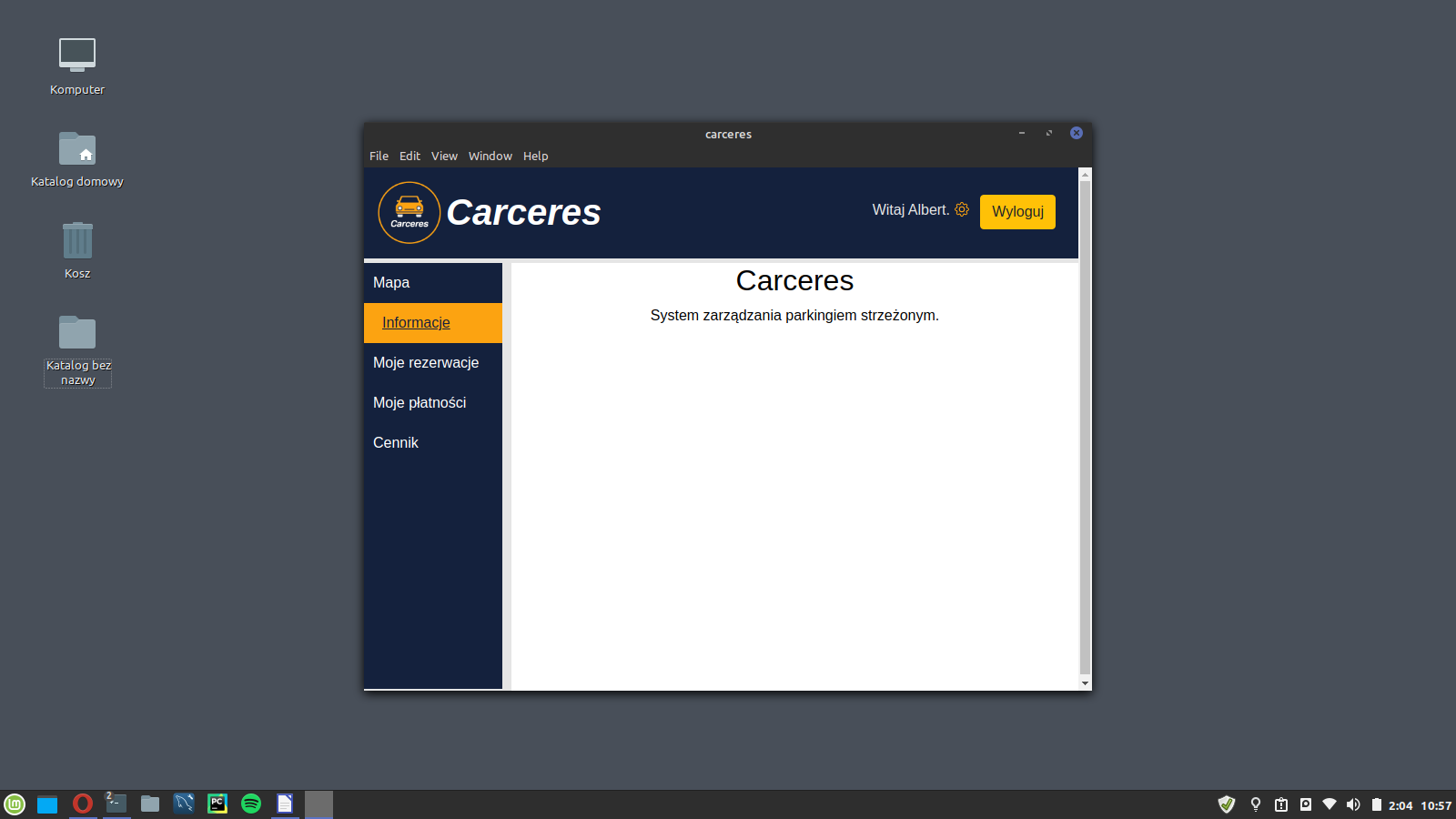
Rys. Backend. Po lewej terminal API (Flask), po prawej Node.JS

Instrukcja użytkowania

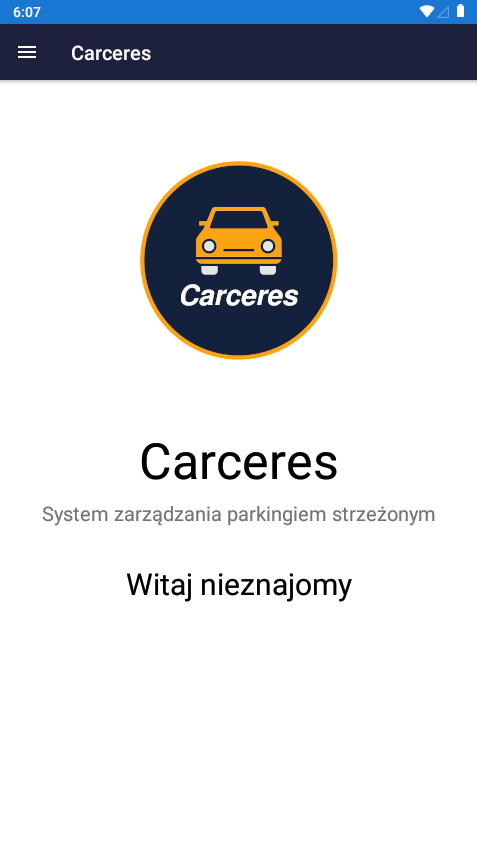
Aby móc korzystać użytkownik musi uruchomić albo wersję WEB (wpisując adres internetowy) albo uruchomić wersję desktopową.



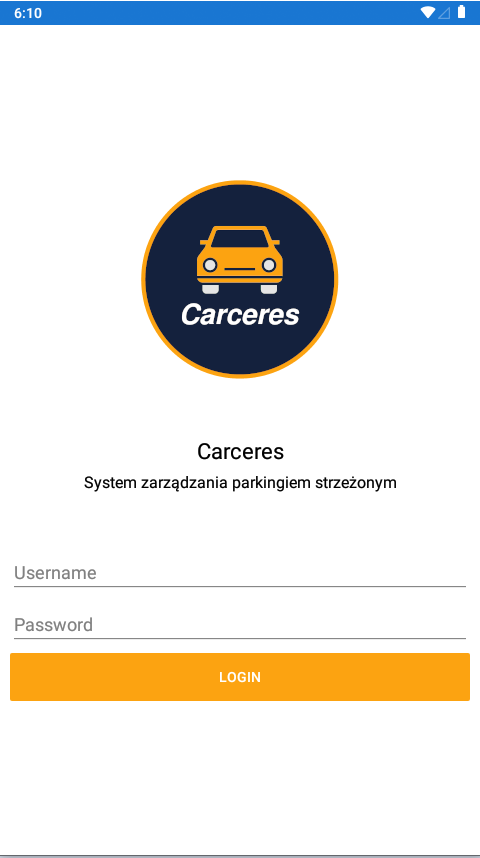
Rys. Widok po uruchomieniu wersji WEB.



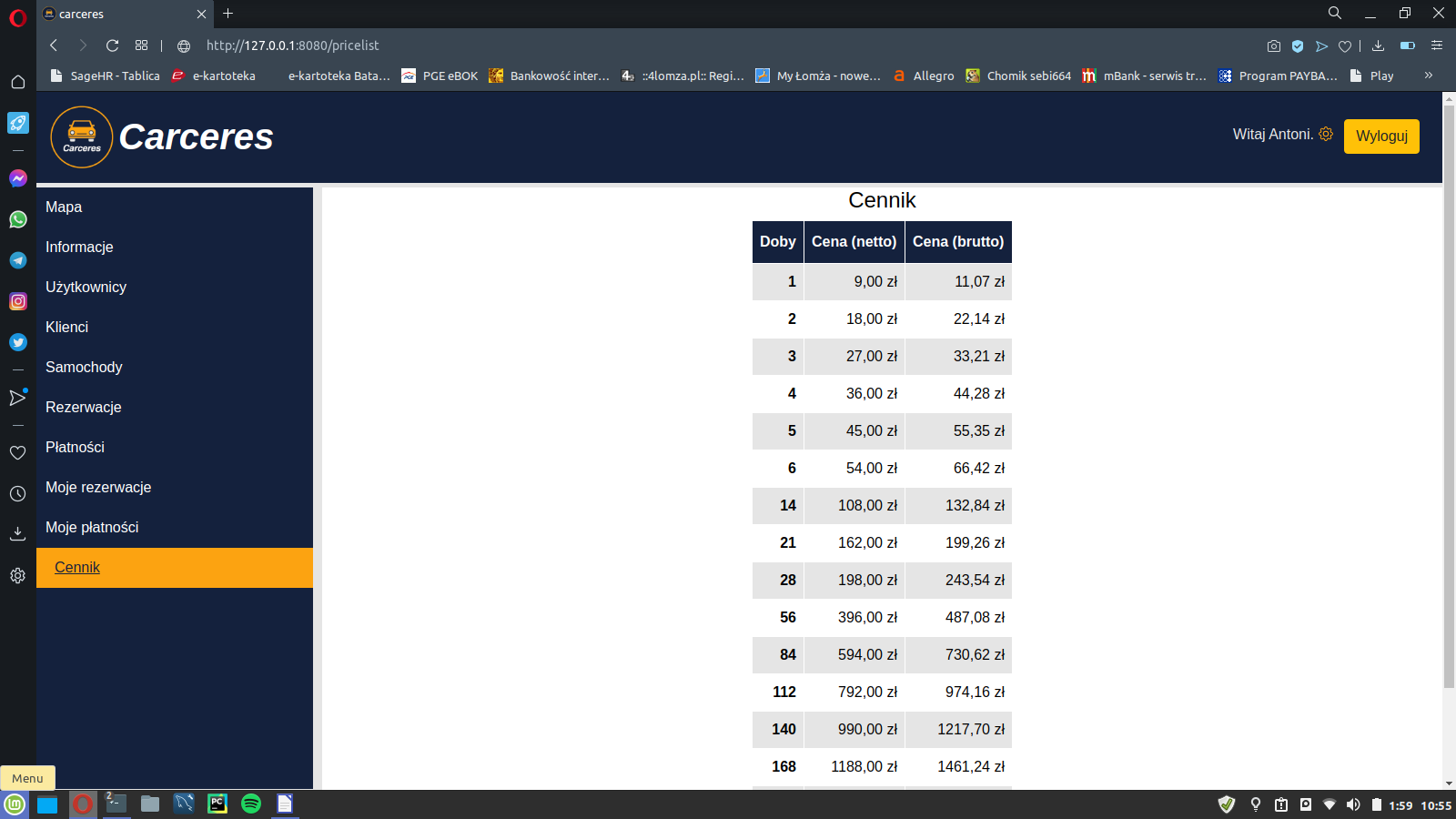
Rys. Widok po uruchomieniu wersji desktop.



Rys. Widok po uruchomieniu wersji mobile.

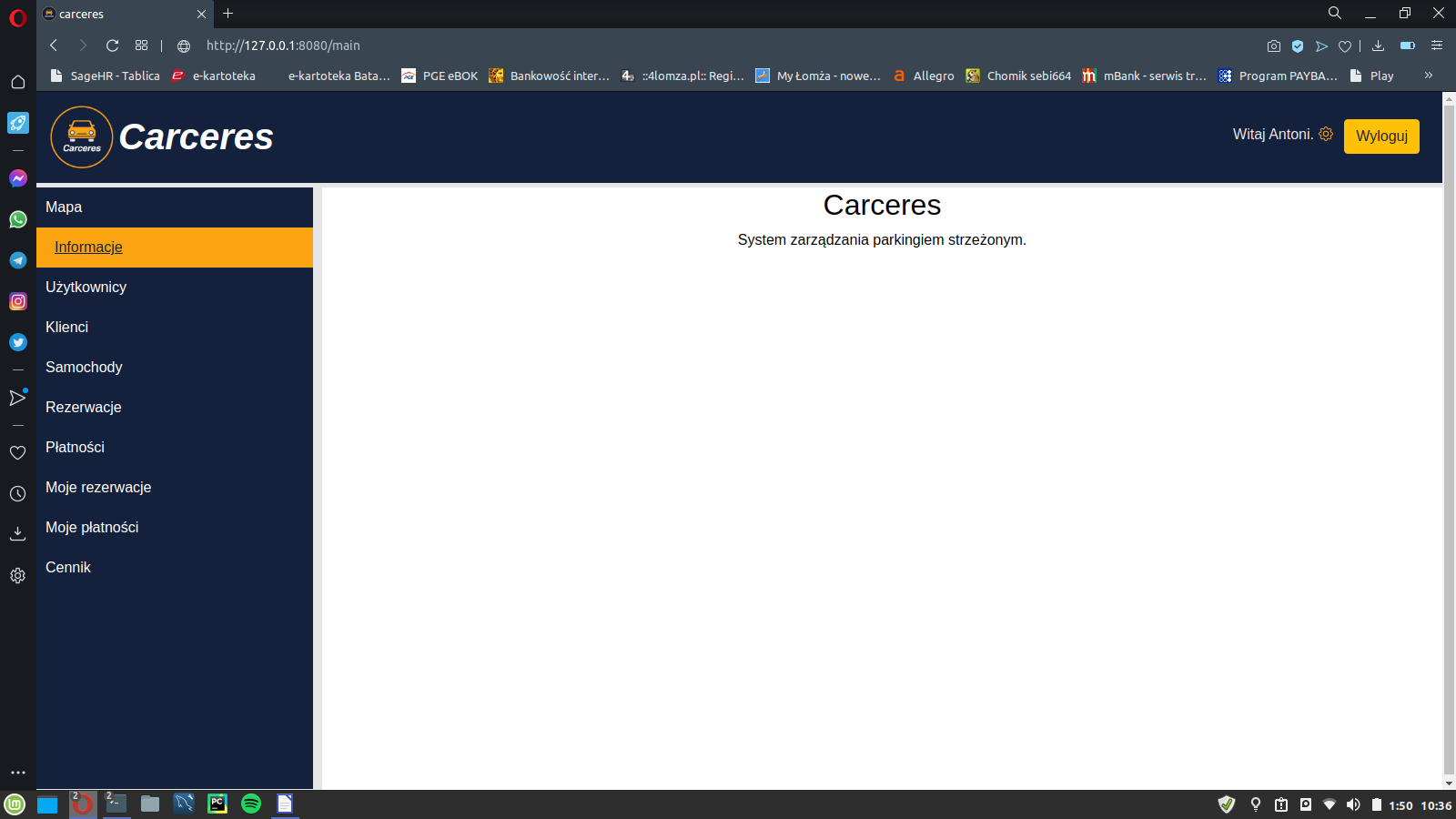


Rys. Ekran logowania wersji mobile.

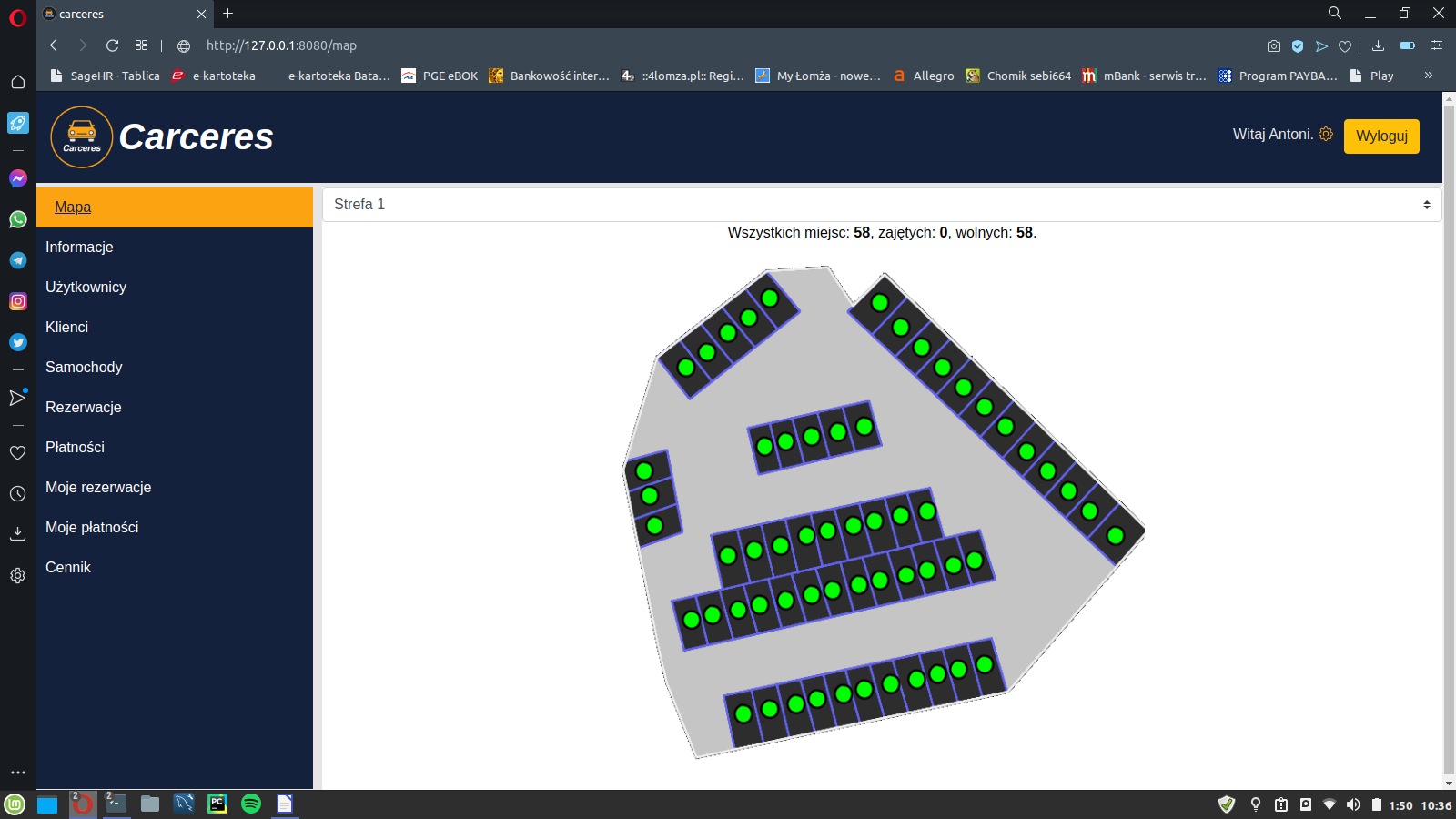


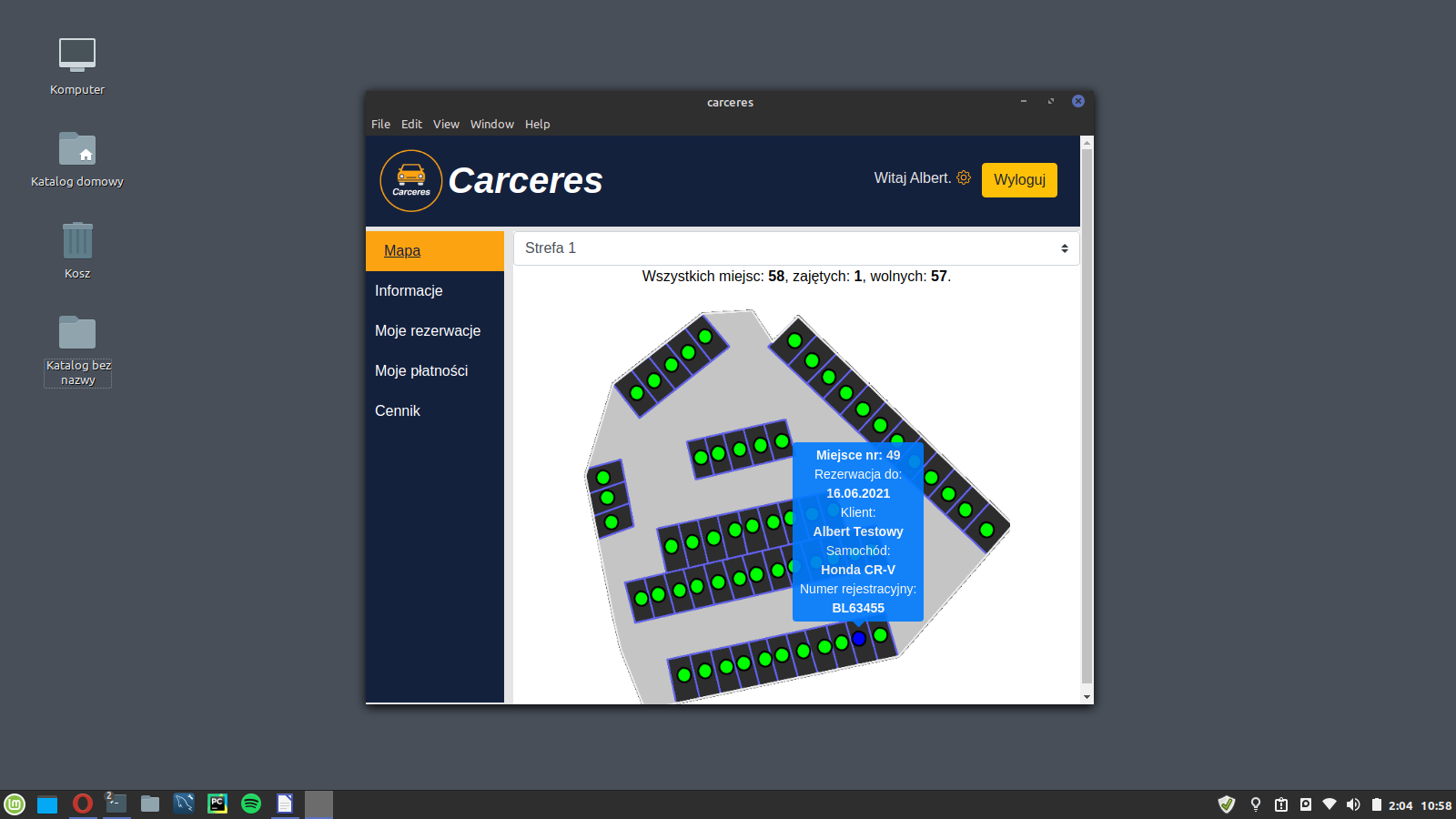
Rys. Widok cennika – wersja WEB.

Użytkownik bez zalogowania ma dostęp jedynie do cennika, informacji ogólnych oraz mapy parkingu. Aby móc np. zarezerwować miejsce musi się zalogować (jeżeli nie ma konta, musi poprosić o nie np. stróża).

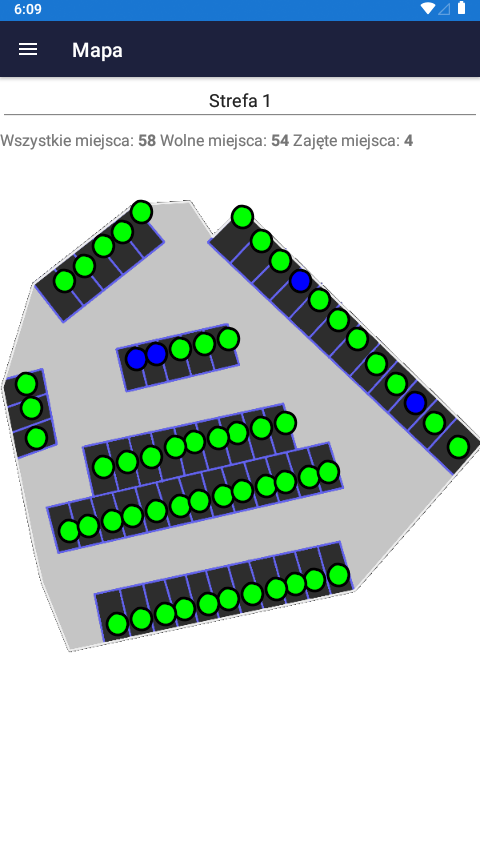


Rys. Widok po zalogowaniu administratora (stróża) wersji WEB.

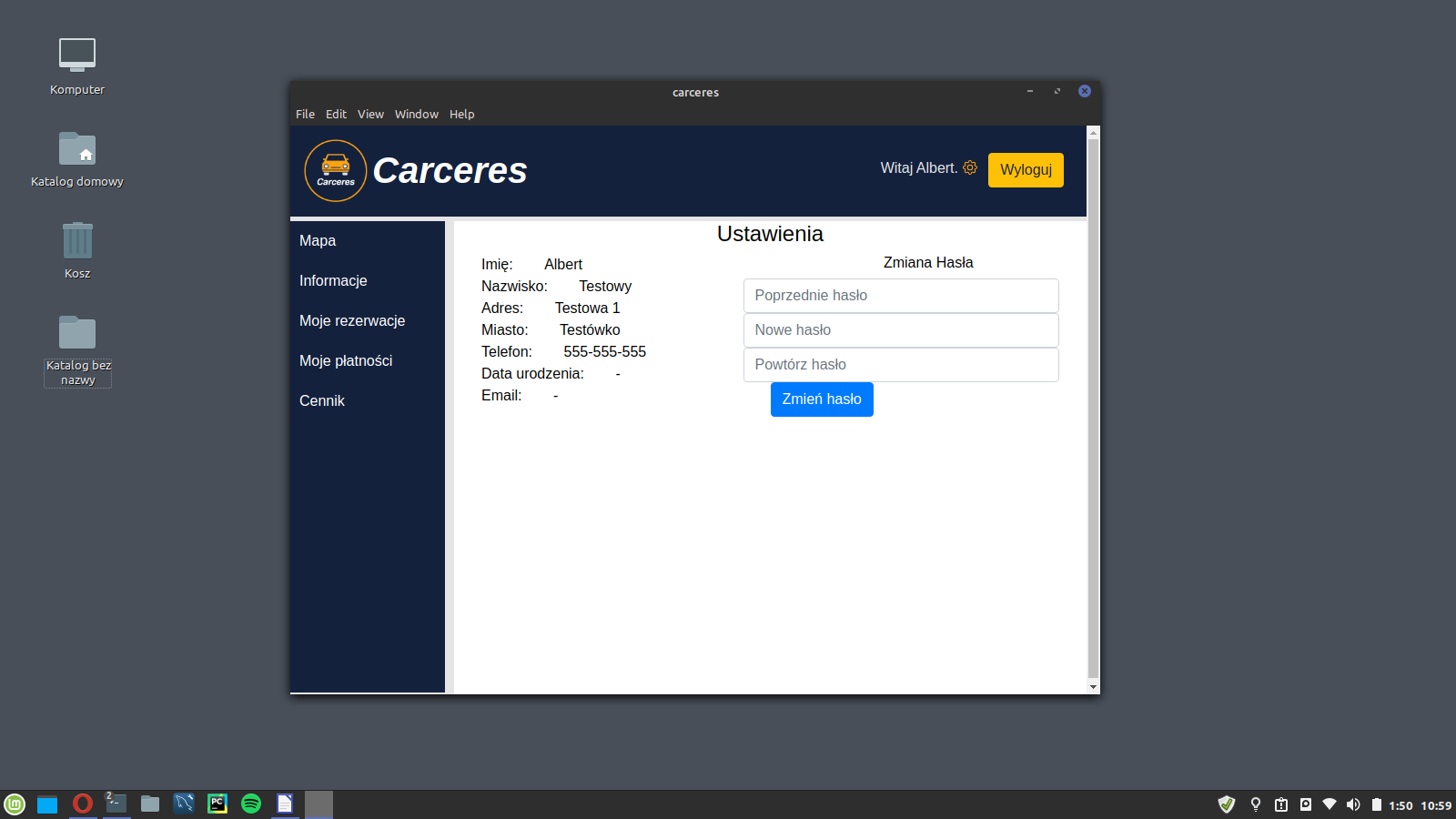
Rys. Widok na mapę parkingu – wersja WEB.



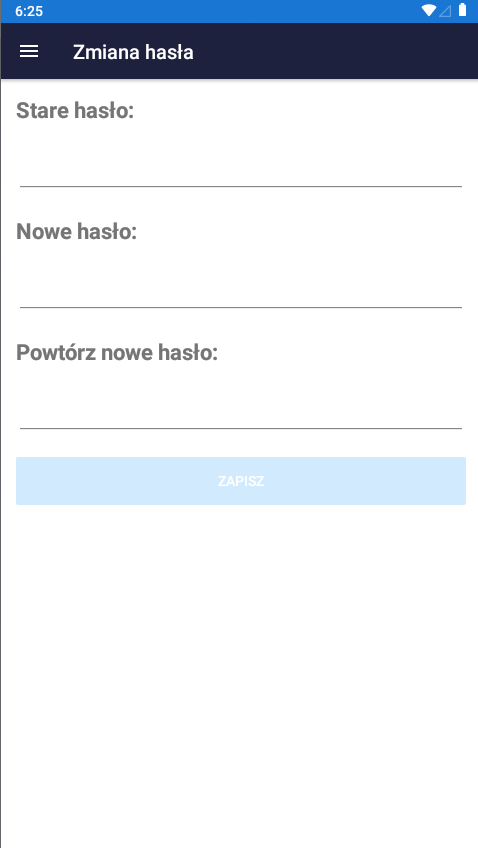
Rys. Widok na mapę parkingu – wersja desktop.



Rys. Widok na mapę parkingu – wersja mobile.

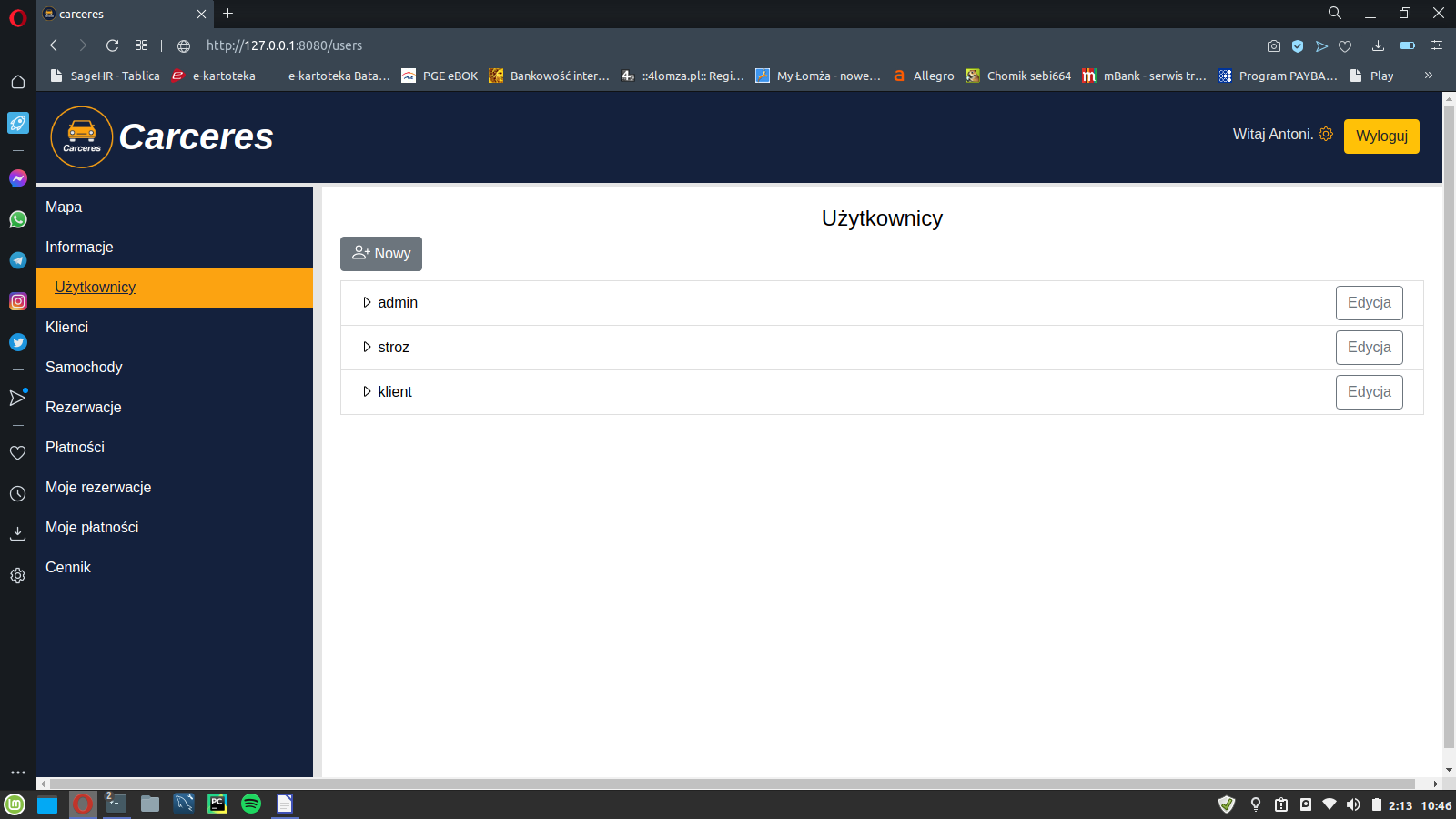


Rys. Ustawienia użytkownika – wersja desktop.

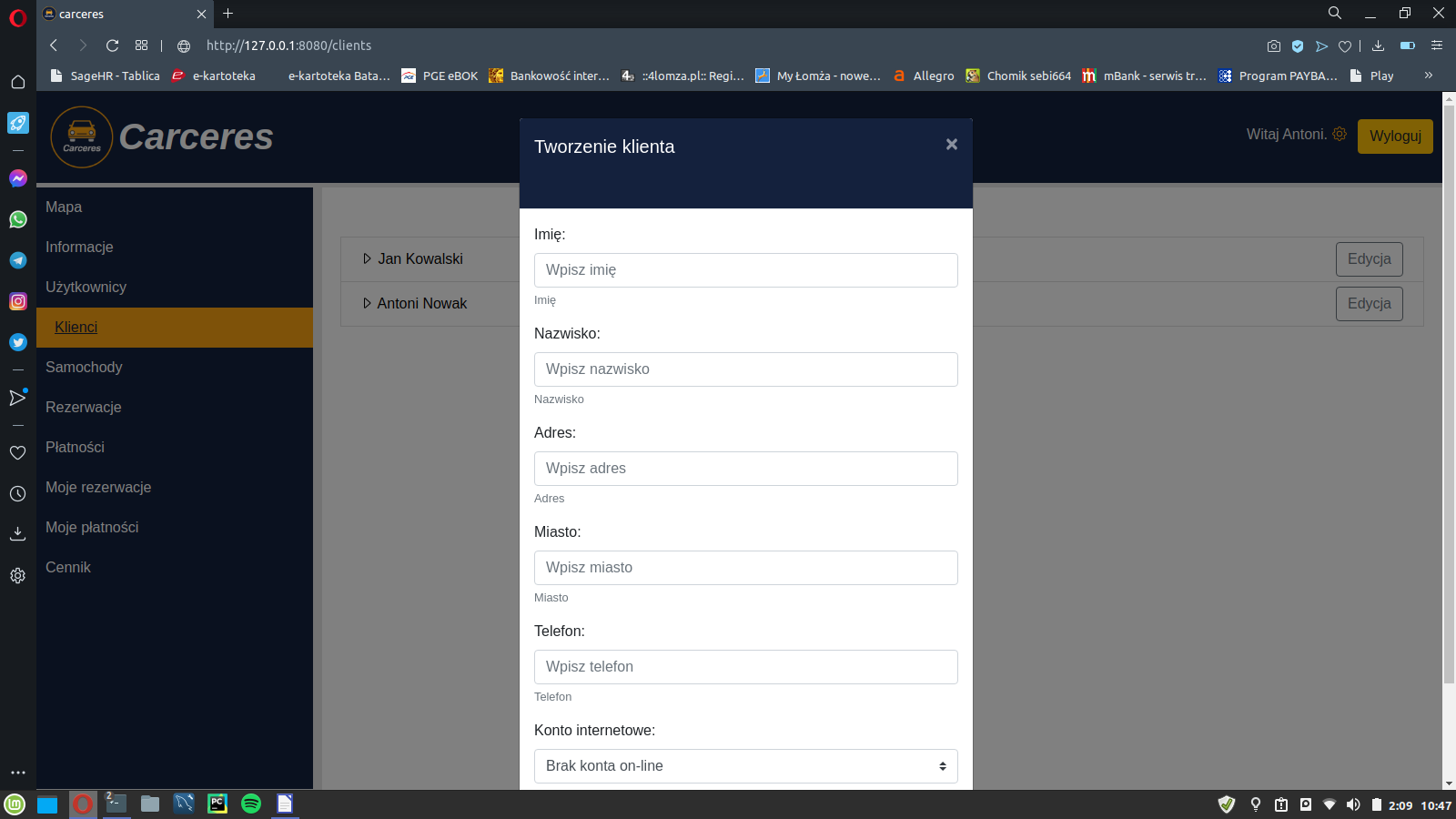


Rys. Ustawienia użytkownika – wersja mobile.

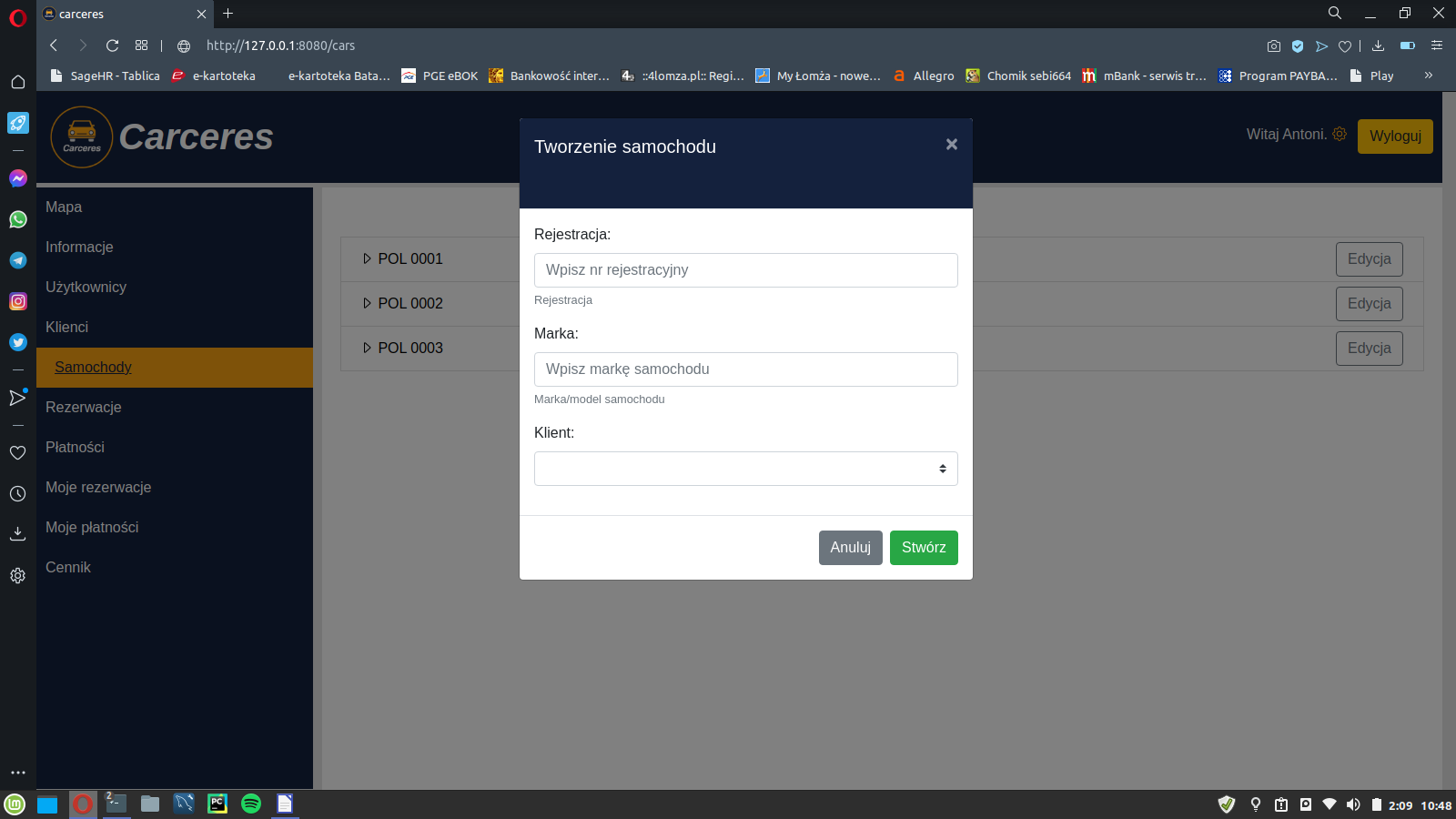
Po zalogowaniu (w zależności od rodzaju konta) użytkownik otrzymuje dostęp do funkcjonalności systemu: możliwość rezerwacji, sprawdzenia rezerwacji, sprawdzenia stanu płatności. Administrator dodatkowo może tworzyć konta użytkowników.



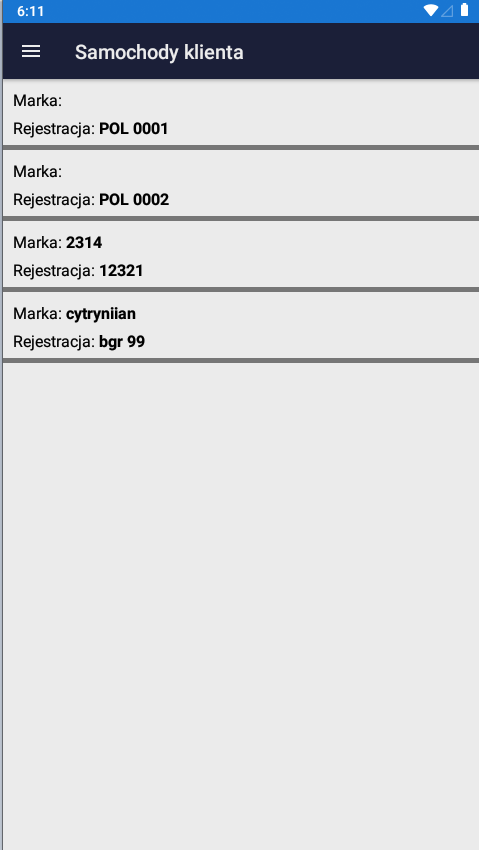
Rys. Lista użytkowników – wersja WEB.



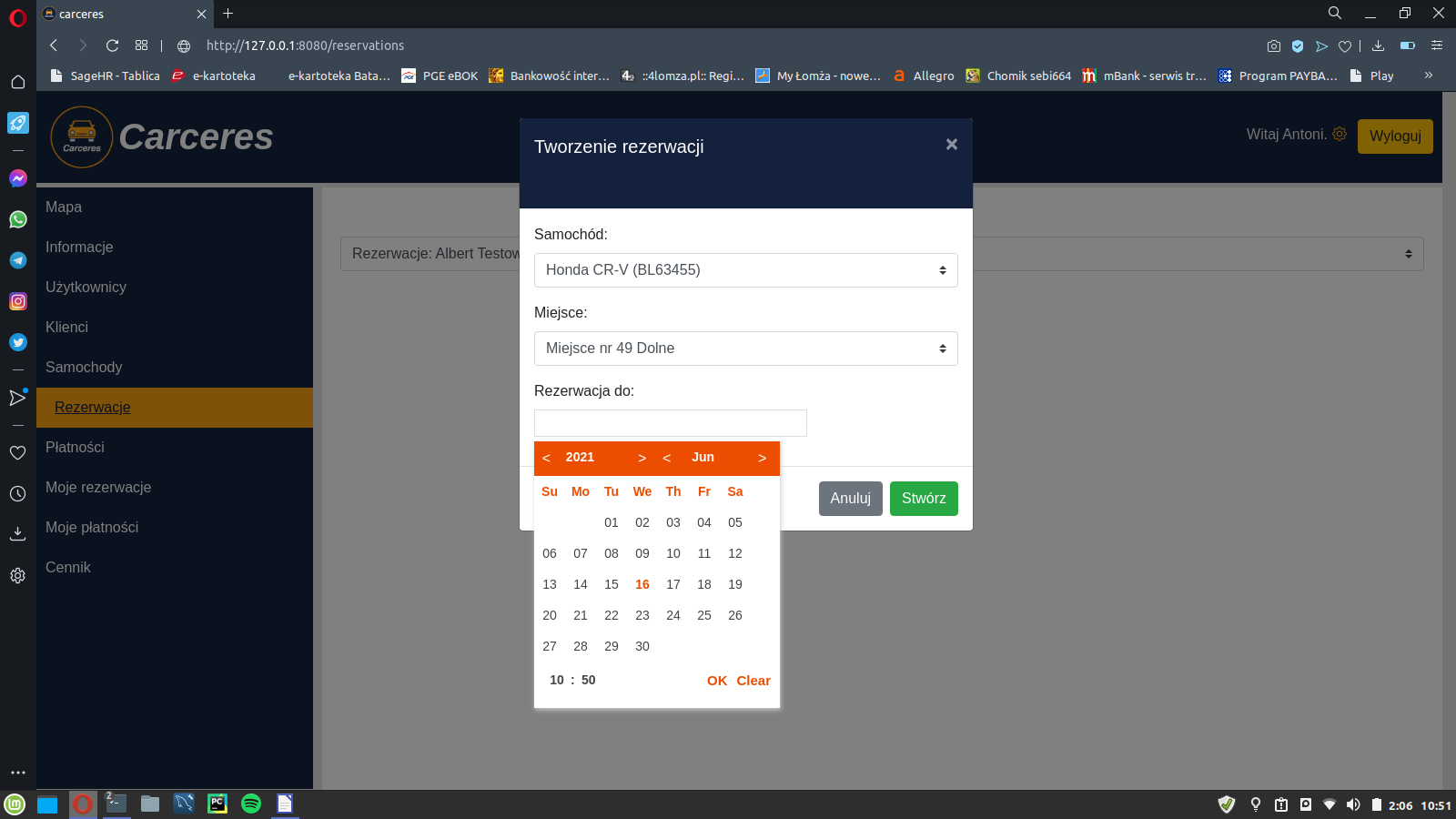
Rys. Tworzenie klienta – wersja WEB.



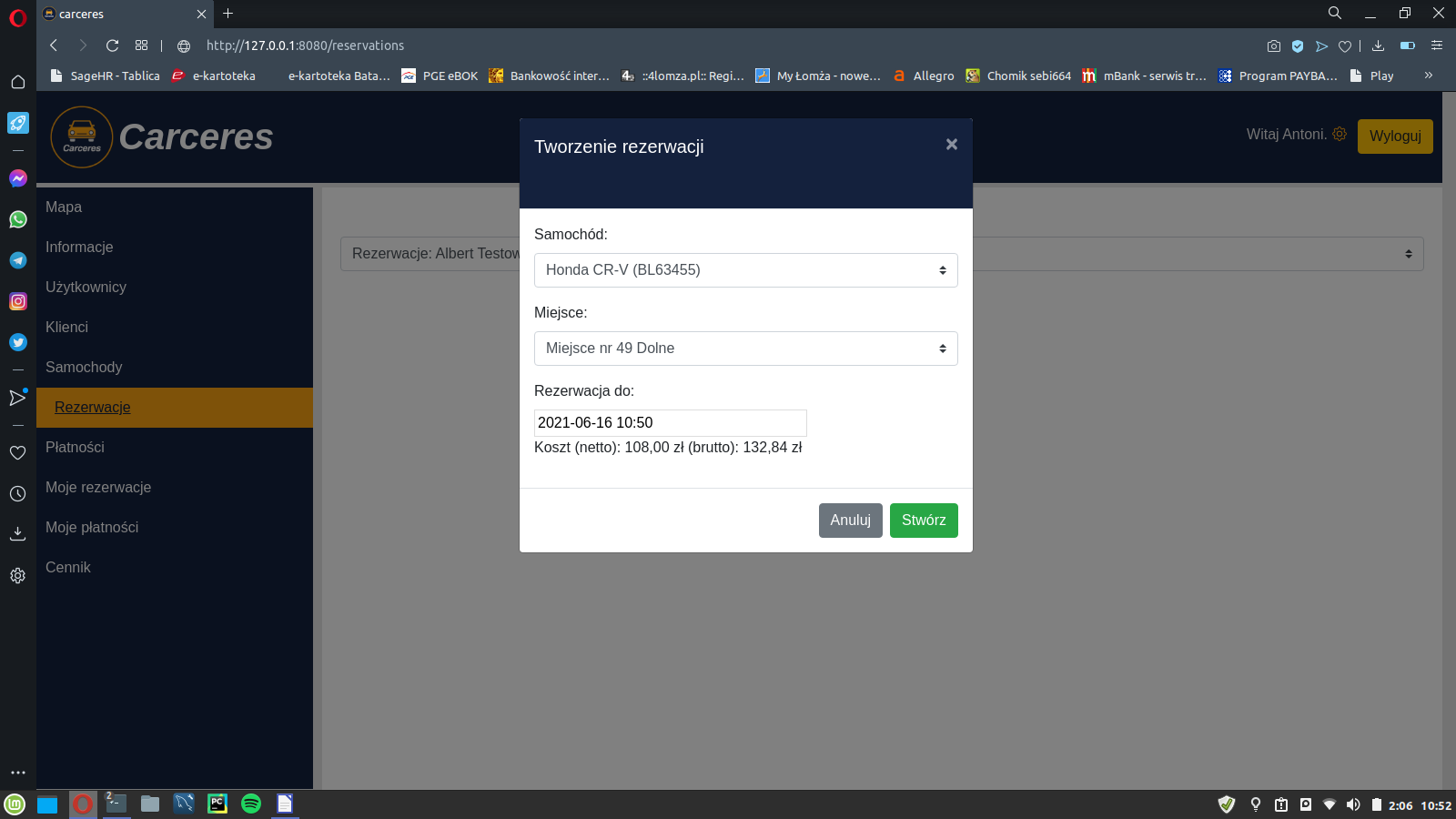
Rys. Tworzenie samochodu – wersja WEB.



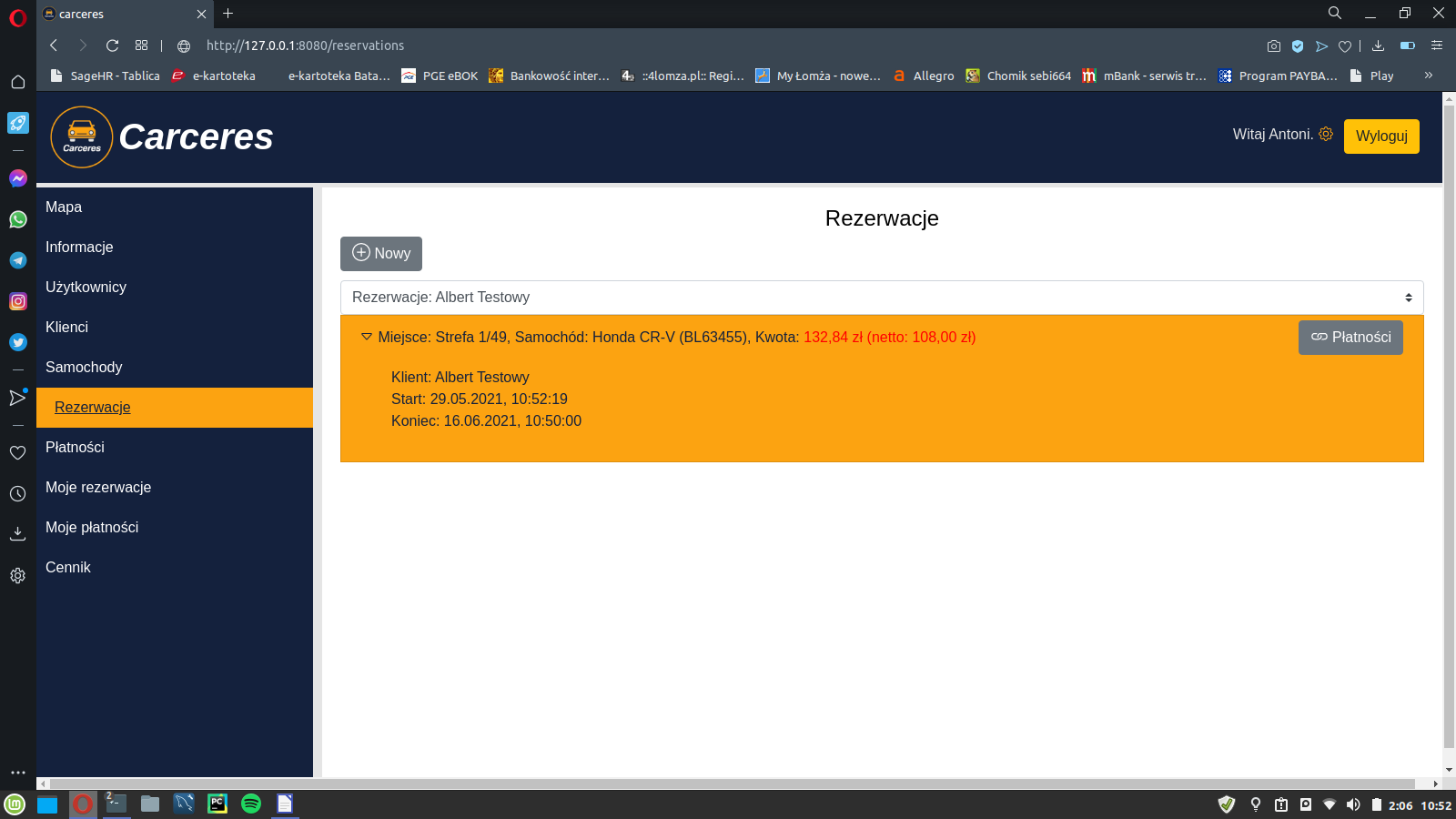
Rys. Lista samochodów klienta – wersja mobile.



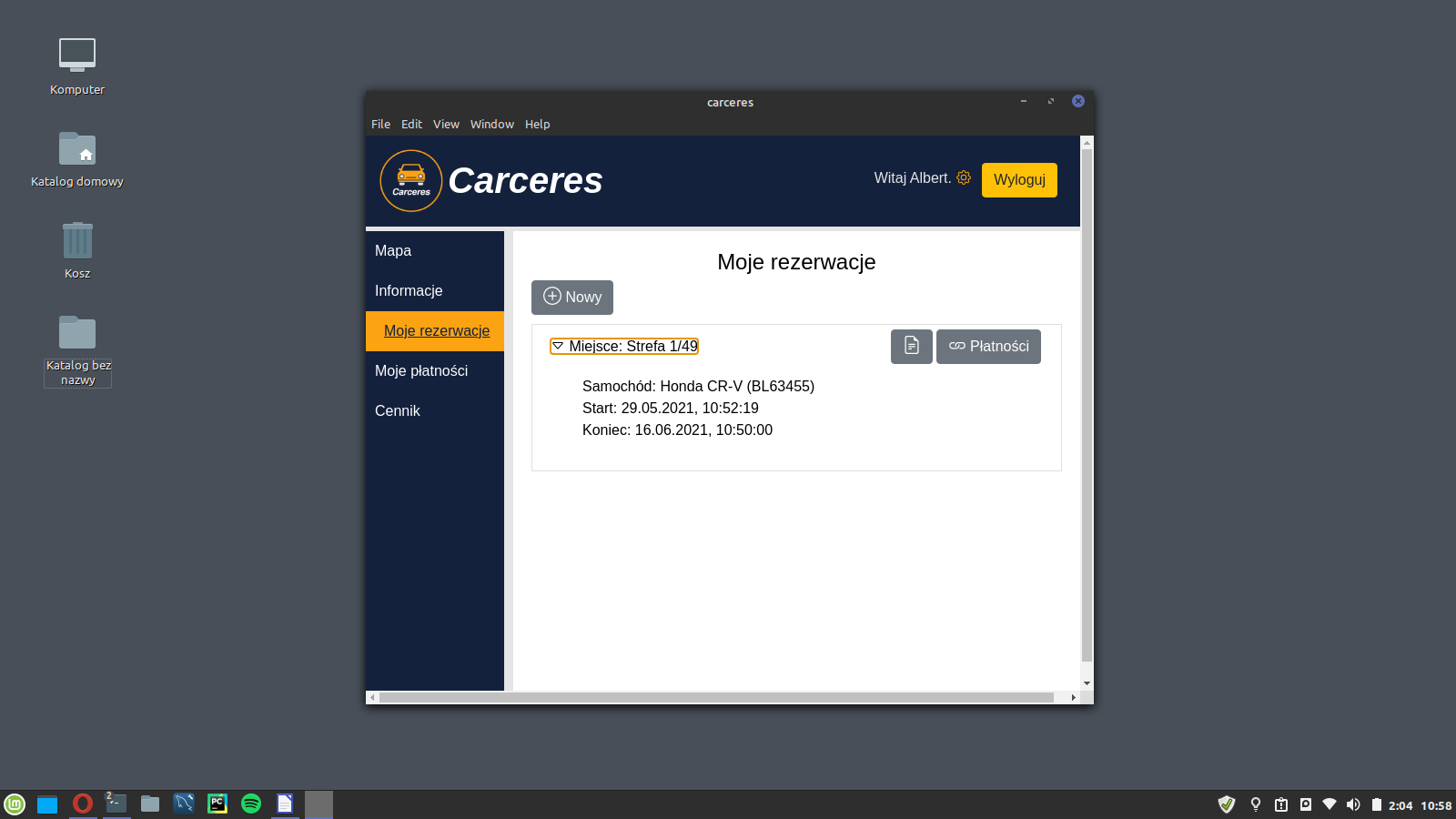
Rys. Tworzenie nowej rezerwacji – wersja WEB.



Rys. Tworzenie rezerwacji – wersja WEB.



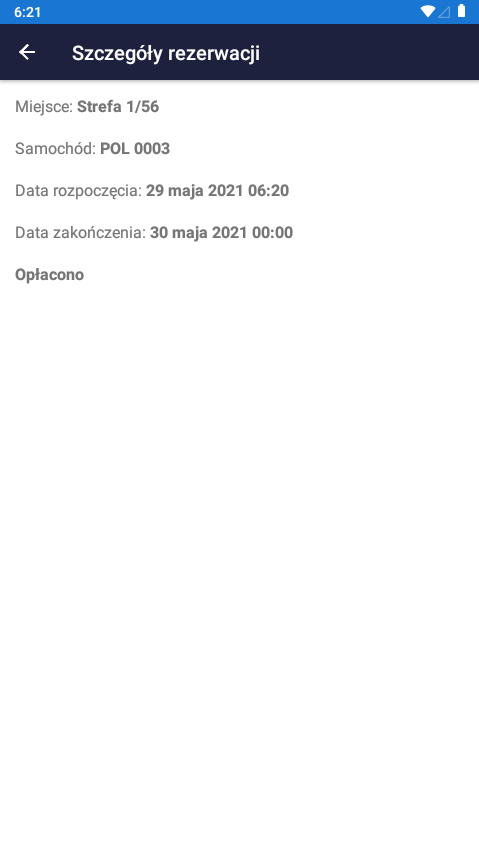
Rys. Lista rezerwacji – wersja WEB.



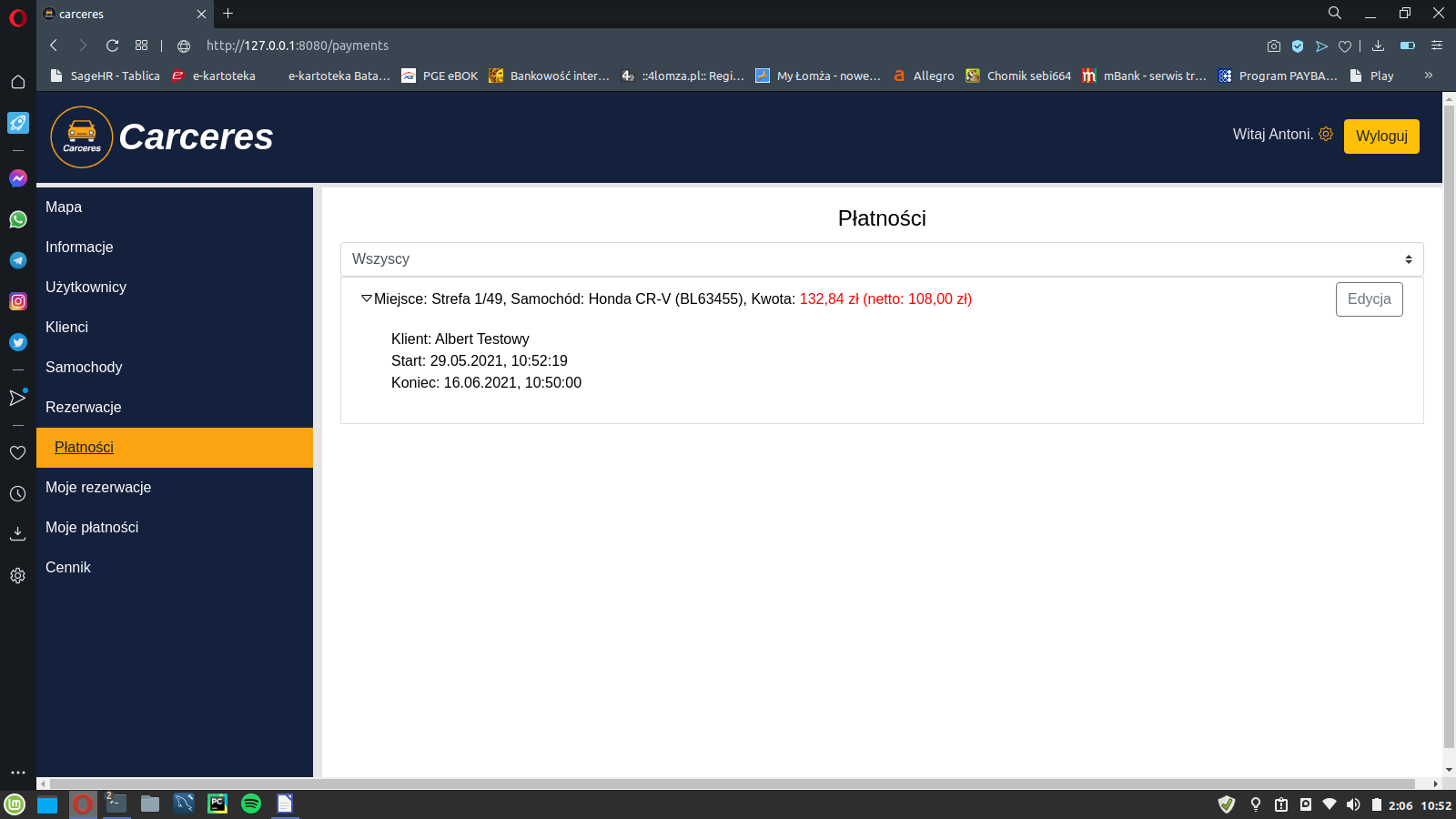
Rys. Lista rezerwacji – wersja desktop.

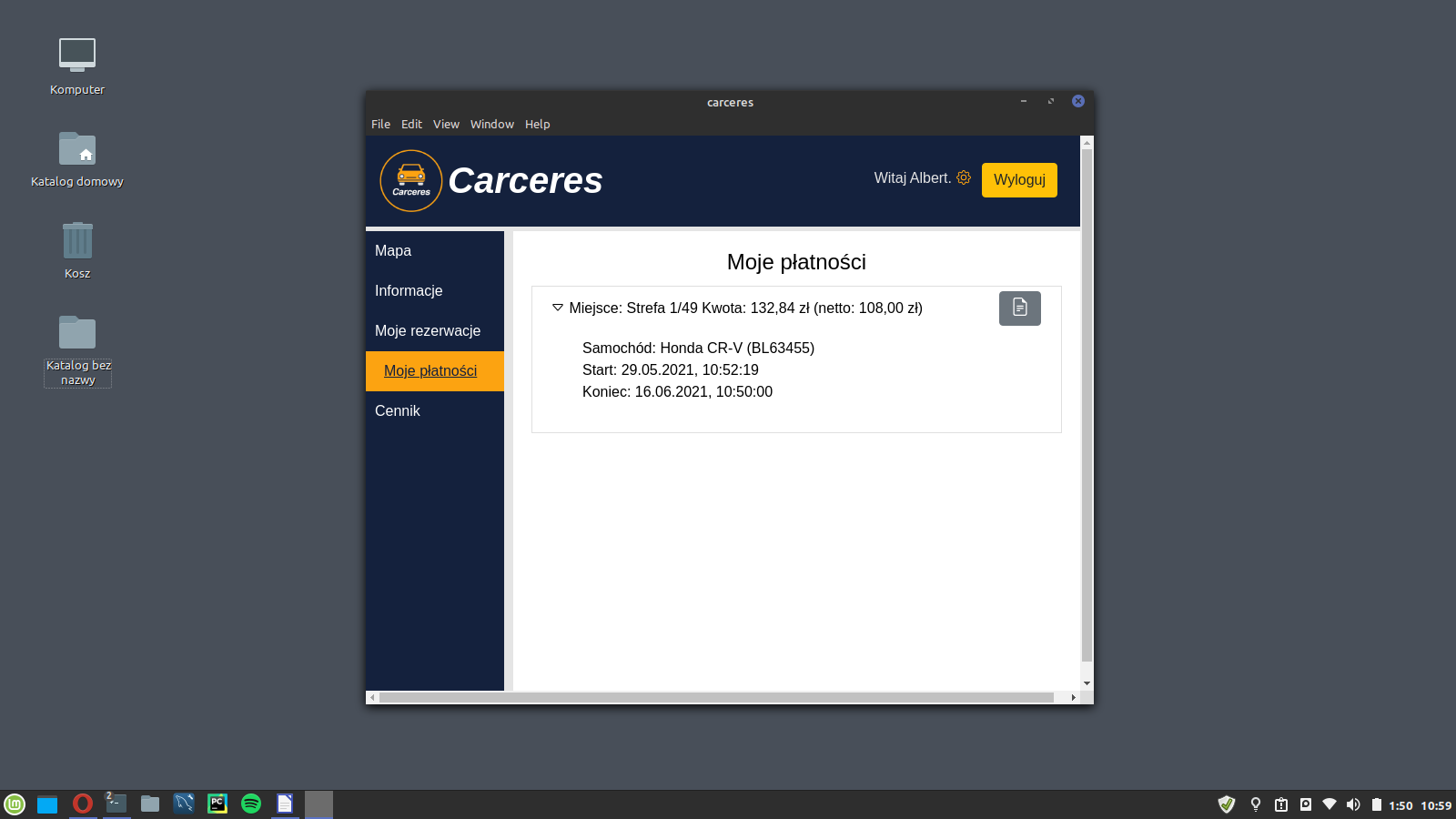


Rys. Lista rezerwacji – wersja mobile.



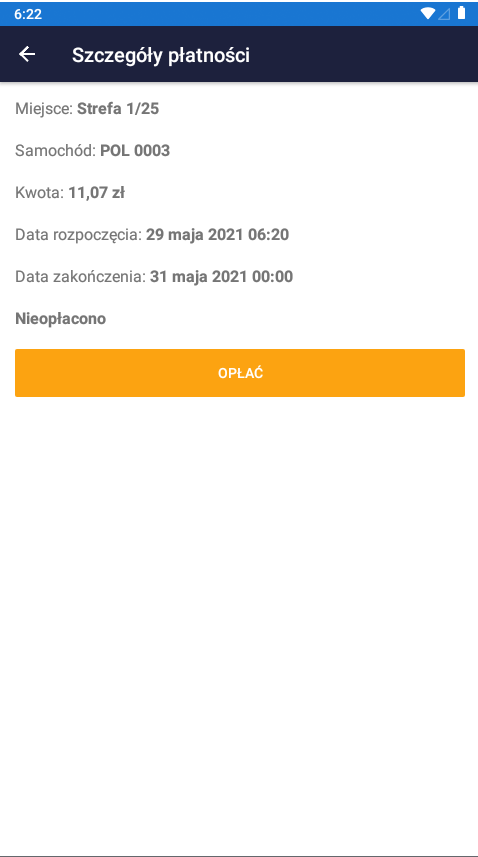
Rys. Szczegóły rezerwacji – wersja mobile.

****Rys. Lista płatności – wersja WEB.

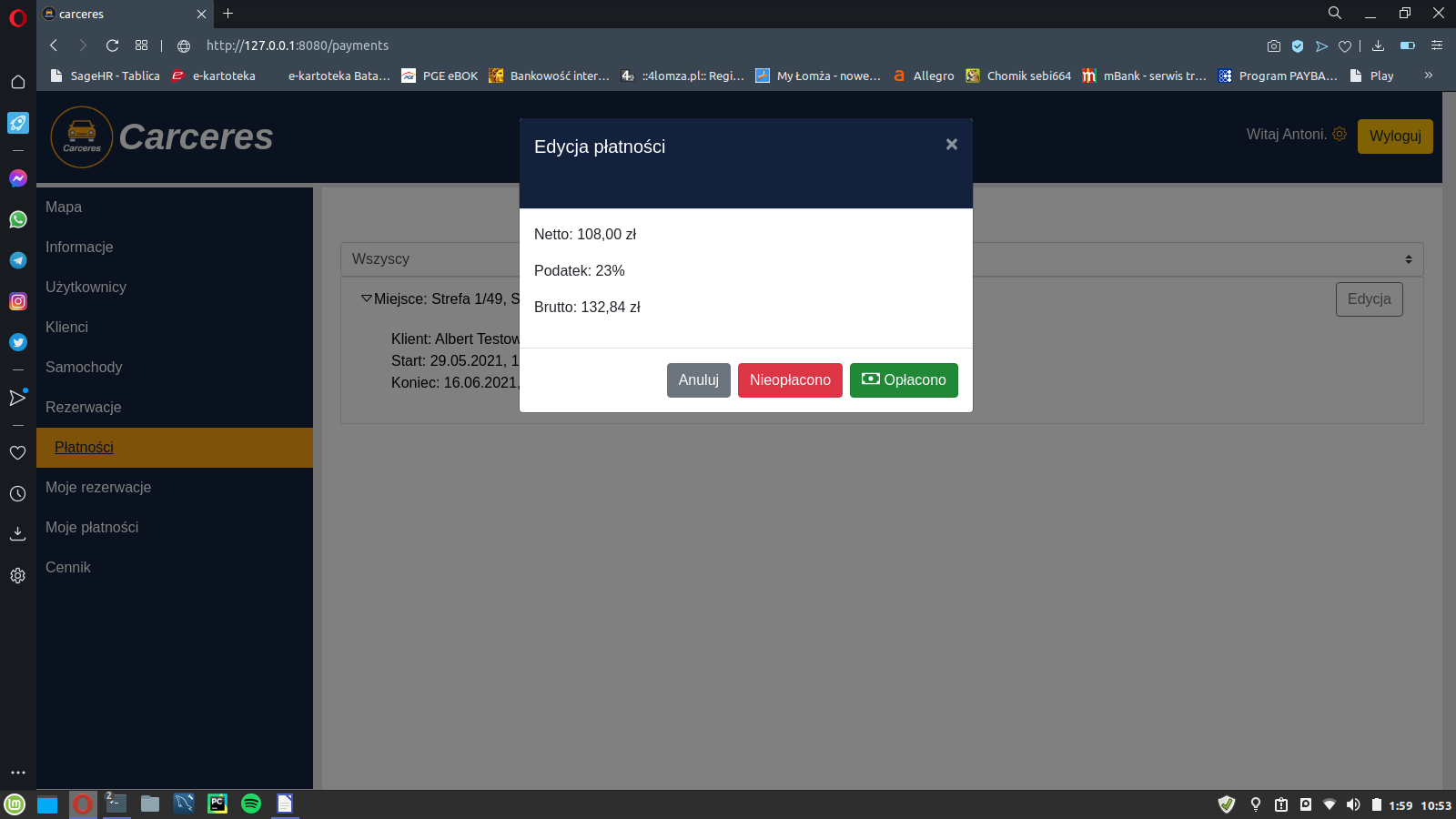
Rys. Lista płatności – wersja desktop.

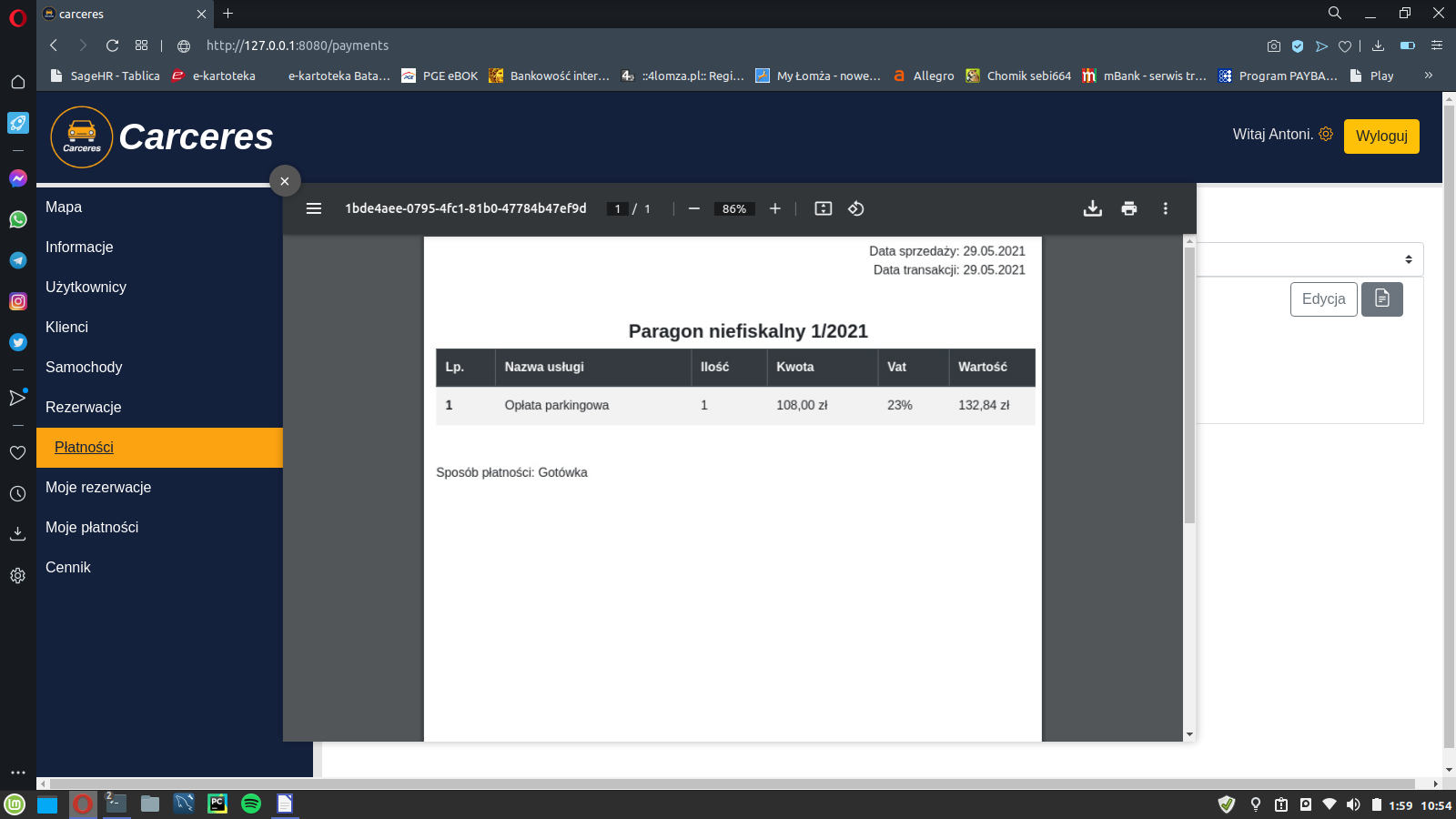


Rys. Lista płatności – wersja mobile.



Rys. Szczegóły płatności – wersja mobile.

****Rys. Widok na okienko płatności – wersja WEB.

Rys. Podgląd paragonu – wersja WEB.

Podsumowanie

Główne założenia systemu zostały spełnione. Aplikacja działa wieloplatformowo co jest istotnym atutem systemu. Nie jest istotny ani rodzaj platformy sprzętowej ani system operacyjny. Do dodatkowej realizacji pozostało zintegrowanie aplikacji z systemem płatności internetowych, przygotowanie pod obsługę urządzeń typu bramy wjazdowe itp.

Literatura

<https://python101.readthedocs.io/pl/latest/webflask/>

<https://flask.palletsprojects.com/en/2.0.x/>

<https://nodejs.org/en/docs/>

<https://getbootstrap.com/docs/4.1/getting-started/introduction/>

<https://docs.microsoft.com/pl-pl/xamarin/xamarin-forms/>

Słownik

**Klient –** osoba korzystająca z parkingu,

**Strefa parkingowa –** teren wykorzystany i podzielony na miejsca parkingowe,

**Miejsce parkingowe –** obszar strefy parkingowej na którym może znajdować się pojazd klienta

**Abonament –** czas przez który klient może korzystać w parkingu,

**Karta parkingowa –** dokument umożliwiający wjazd oraz wyjazd z parkingu, wydawany klientowi po dokonaniu rezerwacji i jej opłaceniu,

**Stróż –** pracownik parkingu obsługujący system,

**Zarządca parkingu –** właściciel parkingu