**Wydział Matematyczno – Przyrodniczy  
Uniwersytet Rzeszowski**

**Przedmiot:**

**Programowanie aplikacji na urządzenia mobilne**

**Dokumentacja techniczna projektu:**

***Wypożyczalnia samochodów***

***„CarBeaver”***

**Wykonanie:**

**Joanna Zubel**

**Paweł Skocz**

**Dominik Gołąb**

**Sylwester Bontur**

**Prowadzący: mgr inż. Marcin Chyła**

**Rzeszów 2018**

**Spis treści**

[1. Opis programu 3](#_Toc533851387)

[2. Charakterystyka programu 3](#_Toc533851388)

[2.1. Przeznaczenie programu 3](#_Toc533851389)

[2.2. Przegląd zastosowań programu 3](#_Toc533851390)

[3. Struktura programu – serwer 3](#_Toc533851391)

[3.1. Dane wykorzystywane przez program 3](#_Toc533851392)

[3.2. Baza danych 3](#_Toc533851393)

[3.3. Obsługa bazy danych 4](#_Toc533851394)

[3.4. Struktura serwera 4](#_Toc533851395)

[3.5. Konfiguracja połączenia z bazą danych 5](#_Toc533851396)

[3.6. Komunikacja klient – serwer 6](#_Toc533851397)

[4. Struktura programu – klient 6](#_Toc533851398)

[4.1. Informacje ogólne 6](#_Toc533851399)

[4.2. Podział na moduły 6](#_Toc533851400)

[4.3. Moduł aktywności 7](#_Toc533851401)

[4.4. Moduł modeli 8](#_Toc533851402)

[4.5. Przekazywanie obiektów między modułami 8](#_Toc533851403)

[4.6. Moduł narzędzi 9](#_Toc533851404)

[4.7. Asynchroniczne wykonywanie zadań 9](#_Toc533851405)

[4.8. Połączenie klient – serwer 10](#_Toc533851406)

[4.9. Tworzenie żądania HTTP 12](#_Toc533851407)

[4.10. Moduł zasobów 13](#_Toc533851408)

[5. Podręcznik użytkownika 14](#_Toc533851409)

[5.1. Rejestracja 14](#_Toc533851410)

[5.2. Logowanie do systemu 14](#_Toc533851411)

[5.3. Główne okno użytkownika 15](#_Toc533851412)

[5.4. Składanie wypożyczenia 15](#_Toc533851413)

[5.5. Historia wypożyczeń 16](#_Toc533851414)

[6. Literatura (źródła internetowe) 17](#_Toc533851415)

[7. Spis ilustracji 17](#_Toc533851416)

# 

# Opis programu

„CarBeaver” to prototyp systemu z aplikacją na urządzenia mobilne działającej na platformie Android. Umożliwia obsługę zewnętrznej bazy danych i synchronizowanie informacji na wszystkich urządzeniach z zainstalowaną aplikacją. W podstawowej konfiguracji umożliwia tworzenie i weryfikowanie użytkowników, istniejących wypożyczeń oraz składanie nowych.

# Charakterystyka programu

## Przeznaczenie programu

Aplikacja przeznaczona jest dla wszystkich użytkowników platformy Android (głównie smartfonów i tabletów), którzy szukają rozwiązania mogącego pomóc wypożyczyć samochód w prosty i łatwo dostępny sposób.

## Przegląd zastosowań programu

Aplikacja może zostać wykorzystana w przedsiębiorstwach świadczących usługi wypożyczania samochodów, do celów zwiększenia zasięgu prowadzenia ich działalności oraz zautomatyzowania procesu wypożyczania. Kolejnym zastosowaniem może być stworzenie miejsca dla społeczności ludzi chcących udostępniać odpłatnie swoje pojazdy na określonych warunkach.

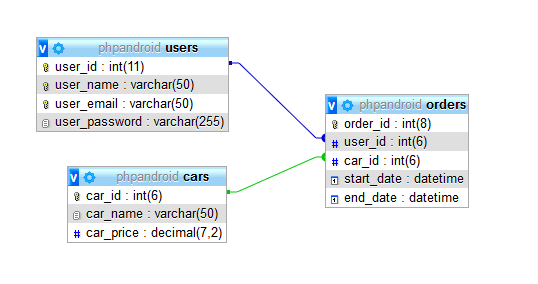
# Struktura programu – serwer

## Dane wykorzystywane przez program

Program wykorzystuje podstawowe dane weryfikacyjne użytkowników, tj. nazwa, email, hasło. Generowane są także dane składanych przez nich zamówień skojarzone z zapisanymi samochodami. Struktura wykorzystywanych i przechowywanych danych może ulec zmianie w zależności od stopnia oraz kierunku dalszego rozwoju aplikacji.

## Baza danych

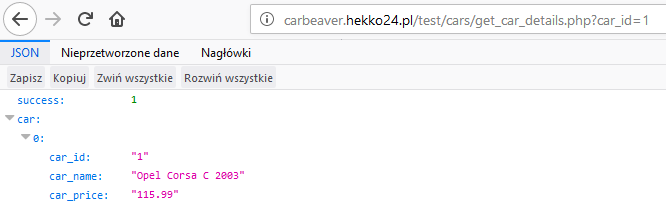
Do przechowywania informacji użyta została baza danych MySQL. Jej struktura to relacyjny model tabel powiązanych ze sobą. Gromadzi wszystkie dane, niezbędne do prawidłowego funkcjonowania aplikacji. Udostępniona w Internecie pozwala na spójne działanie wszystkich zainstalowanych aplikacji, które mają dostęp do sieci.



Rys. 1 Schemat ERD bazy danych

## Obsługa bazy danych

„Po stronie serwera” systemu znajduje się obsługa bazy danych. Realizowana jest przez model CRUD (ang. Create, Read, Update and Delete) zaimplementowanego w języku PHP. Serwer przyjmuje, obsługuje żądania od „klienta” (aplikacji Android), a następnie zwraca wyniki i komunikaty. Sama komunikacja pomiędzy serwerem a klientem odbywa się za pomocą formatu JSON (ang. JavaScript Object Notation), natomiast w odwrotnym kierunku (klient – serwer) za pomocą metod POST i GET protokołu HTTP.



Rys. 2 Odpowiedź serwera w formacie JSON

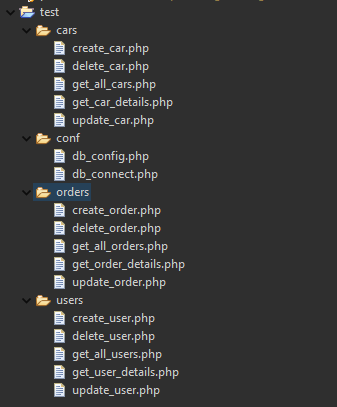
## Struktura serwera

Serwer aplikacji „CarBeaver” składa się z prostej struktury katalogów (cars, orders, users i conf). Katalog „conf” przechowuje skrypty niezbędne do nawiązania połączenia z bazą danych. Pozostałe foldery mają domyślnie tworzyć intuicyjny adres URL odpowiadający podmiotom, wobec których serwer będzie wykonywał konkretne zadania.

Zadania wykonywane przez serwer w zależności od nazwy pliku:

* create\_\*.php – tworzenie nowego rekordu w tabeli,
* delete\_\*.php – usunięcie wskazanego rekordu z tabeli,
* get\_all\_\*.php – zwraca wszystkie rekordy z tabel,
* get\_\*\_details.php – zwraca rekord o wskazanym id z tabeli,
* update\_\*.php – modyfikuje wybrany rekord w tabeli.

W miejsce „\*” należy rozumieć nazwę, która odpowiada nazwie katalogu, a także tabeli w bazie danych (user, car lub oder).

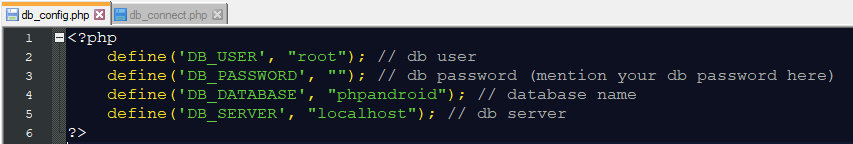


Rys. 3 Struktura katalogów serwera PHP

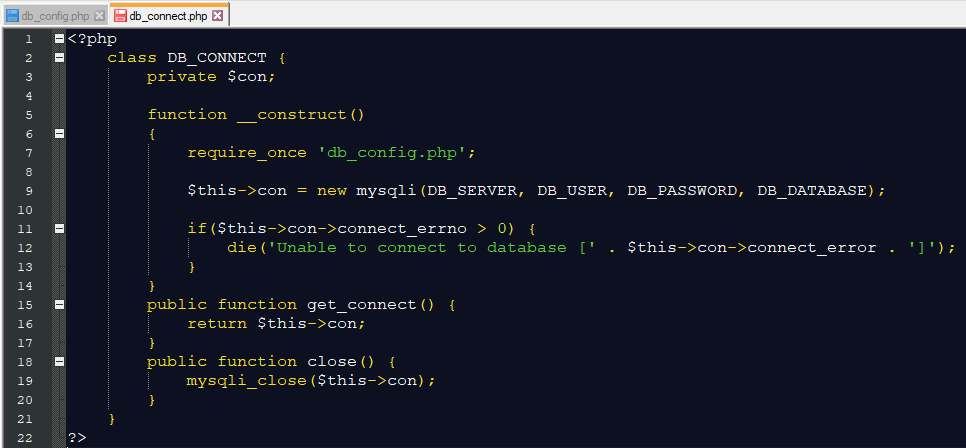
Aby serwer działał w pożądany sposób całą strukturę katalogów i plików należy umieścić w katalogu o nazwie „test”, w głównym folderze serwera PHP.

## Konfiguracja połączenia z bazą danych

Wykorzystywany skrypt PHP łączy się z bazą danych za pomocą dwóch plików konfiguracyjnych: db\_config.php oraz db\_connect.php. Pierwszy odpowiada za definiowanie stałych informacji niezbędnych do połączenia z bazą danych, natomiast drugi stanowi klasę do bezpiecznego połączenia się z bazą.



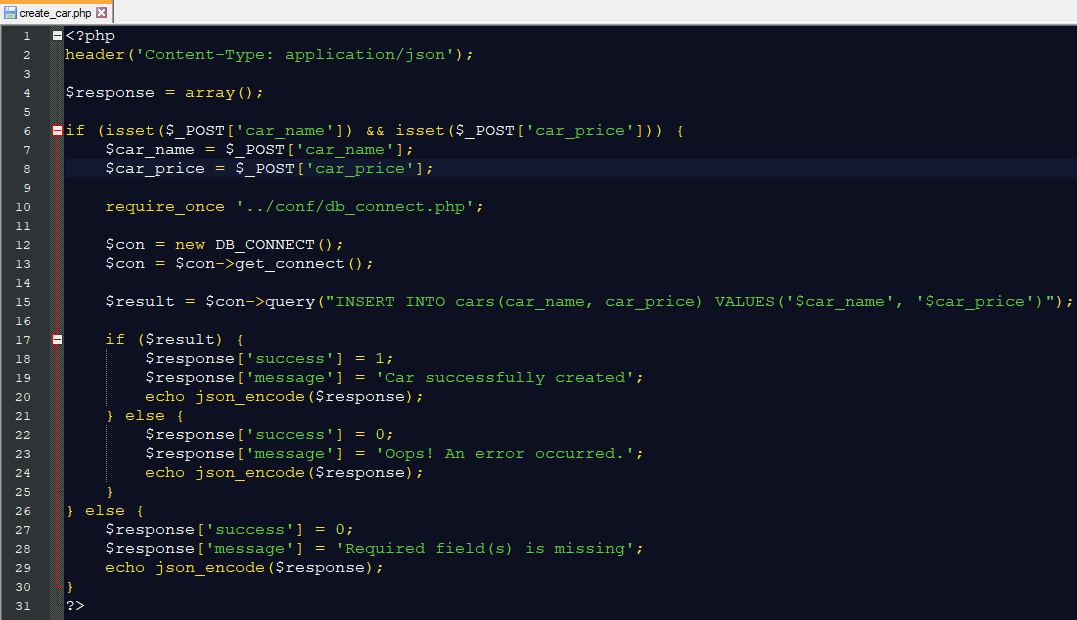
Rys. 4 Plik z danymi do połączenia z bazą danych (db\_conf.php)



Rys. 5 Klasa otwierająca połączenie bazodanowe (db\_connect.php)

## Komunikacja klient – serwer

Dane do serwera mogą być przesyłane przez protokół HTTP, za pośrednictwem jego metod – POST oraz GET. Język PHP interpretuje je w postaci tablic tzw. zmiennych super globalnych $\_POST oraz $\_GET. Serwer przyjmuje dane, wykonuje zapytanie oraz przygotowuje i zwraca odpowiedź. Do odpowiedzi zawsze dołączany jest komunikat o powodzeniu operacji (1 – sukces, 0 – porażka).



Rys. 6 Przykładowy model CRUD serwera

PHP umożliwia poprawne zwracanie wyniku w formacie JSON dzięki dołączeniu nagłówka   
Content-Type oraz użyciu funkcji json\_encode(). Nagłówek informuje przeglądarkę jakiego typu treści ma się spodziewać, funkcja natomiast zamienia tablice na poprawny łańcuch znaków w formacie JSON. Spreparowany przez tę metodę tekst, wystarczy tylko wyświetlić.

Wynik przykładowej odpowiedzi w formacie JSON przedstawia Rys. 2.

# Struktura programu – klient

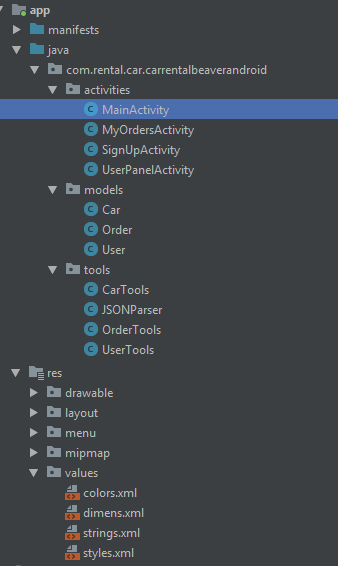
## Informacje ogólne

Aplikacja została stworzona przy użyciu narzędzia Android Studio 3.2.1, które zawiera zestaw niezbędnych komponentów do tworzenia, budowania oraz testowania aplikacji. Kod źródłowy opiera się na języku Java oraz systemie Gradle, służącym do budowy projektu oraz dostarczaniu niezbędnych bibliotek.

## Podział na moduły

Aplikacja na androida systemu „CarBeaver”, zbudowana jest z następujących modułów:

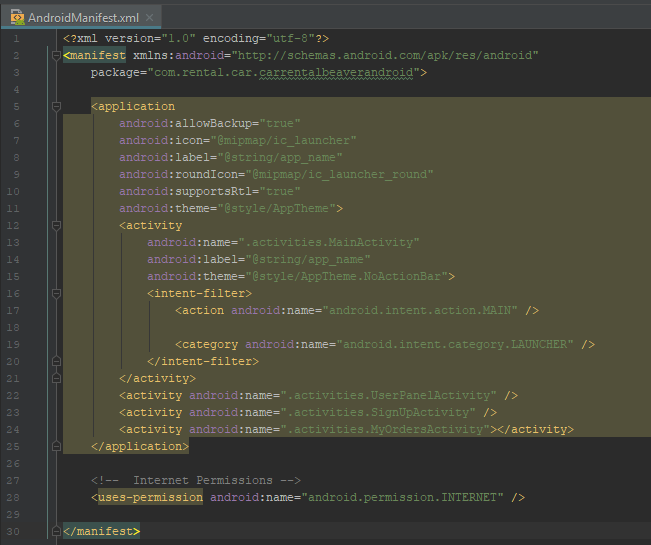
* Aktywności – znajdują się tutaj klasy Javy, które obsługują widoki wyświetlane w aplikacji (GUI),
* Modele – zbiór klas, które są odpowiedzialne za reprezentację konkretnych zestawu danych i obiektów w aplikacji,
* Narzędzia – klasy wykorzystywane do połączeń z serwerem,
* Zasoby – zbiór plików definiujących wygląd, wartości, kolory itp. w dokumentach XML.



Rys. 7 Moduły aplikacji Android „CarBeaver"

## Moduł aktywności

Moduły te to klasy, które dziedziczą po jednej z wbudowanych w Androidzie aktywności, np. „AppCompatActivity”. Jest to specjalna klasa, która umożliwia korzystanie z biblioteki aktywności specyficznych dla paska funkcji. Każda aktywność jest powiązana z konkretnym układem zapisanym w XML. Moduły aktywności umożliwiają zatem obsługę zdarzeń jakie zachodzą na konkretnym ekranie aplikacji. Budowane są w niej metody, które reagują na takie akcje jak kliknięcia w przyciski czy samo załadowanie się aktywności. Każda aktywność musi być dodatkowo zapisana w pliku AdnroidManifest.xml



Rys. 8 AndroidManifest.xml – cechy i komponenty aplikacji

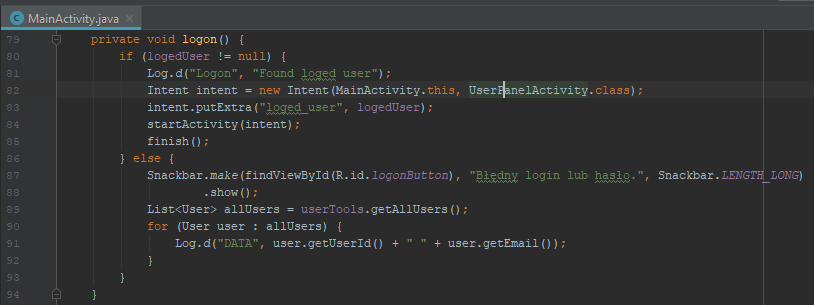
## Moduł modeli

Modele to klasy, które reprezentują pojedyncze zasoby danych, wykorzystywane w aplikacji. Swoją budową odzwierciedlają strukturę bazy danych, tak aby reprezentowane przez nie obiekty pokrywały się z zasobami bazodanowymi.

W działaniu aplikacji konieczne jest, aby część danych mogła być dostępna z różnych miejsc, bez ponownego pobierania ich z bazy danych. Przekazywane są one pomiędzy aktywnościami jako pojedyncza informacja lub jako cały obiekt.

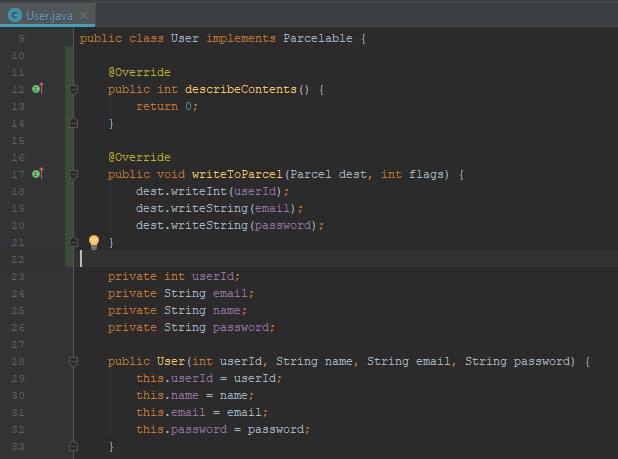
## Przekazywanie obiektów między modułami

Pojedyncze dane, kolekcje oraz obiekty mogą być przekazywane do innej aktywności w momencie jej przygotowania, a następnie uruchomienia. Nową aktywność uruchamia się, budując nową intencję. Do niej dołączane są dodatkowe dane (metoda: putExtra()), które kolejno mogą zostać wykorzystane po jej uruchomieniu.

****

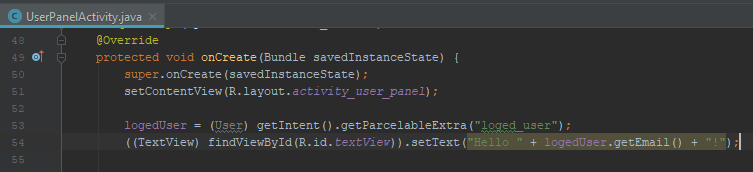
Rys. 9 Przekazywania dodatkowych danych do innej aktywności

W prosty sposób można przekazywać pojedyncze dane standardowego typu danych. Aby przekazać obiekt do innej aktywności musi on implementować wbudowany interfejs Parcel lub Serializable. W pierwszym przypadku konieczna jest implementacja jego abstrakcyjnych metod describeContents() oraz writeToParcel().



Rys. 10 Klasa implementująca interfejs Parcelable

W momencie uruchomienia i budowania się nowej aktywności, z intencji pobierane są dodatkowe dane. Zwraca je metoda getParcelableExtra() po podaniu nazwy z jaką zostały one dołączone do intencji.

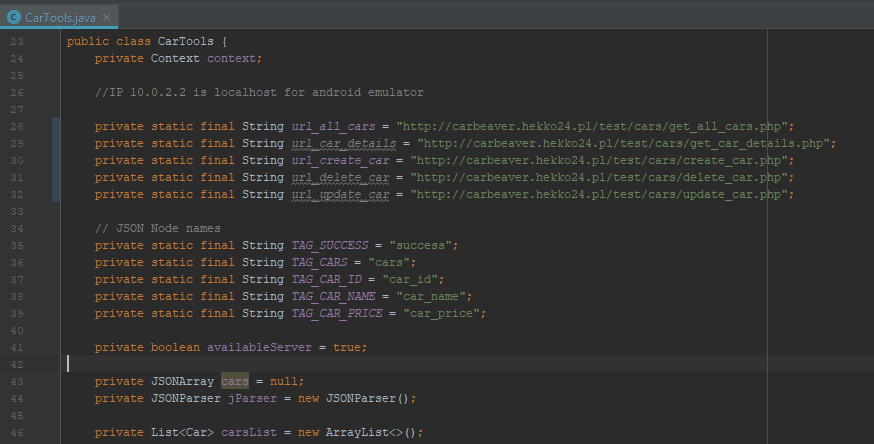


Rys. 11 Odbieranie przekazanych danych z intencji

## Moduł narzędzi

Narzędzia w aplikacji „CarBeaver” odpowiedzialne są za operacje modelu CRUD, czyli tworzenie, pobieranie, edycję oraz usuwanie danych wykorzystywanych w systemie. Dodatkowo odpowiedzialne są za hashowanie haseł dla kont użytkowników, obsługę protokołu HTTP, konwersję z formatu JSON oraz asynchroniczne łączenie z serwerem aplikacji.

Każde narzędzie dla modelu CRUD, posiada pola definiujące adres URL, pod którym znajduje się punkt końcowy serwera przyjmujący żądania HTTP i zwracający dane w formacie JSON.

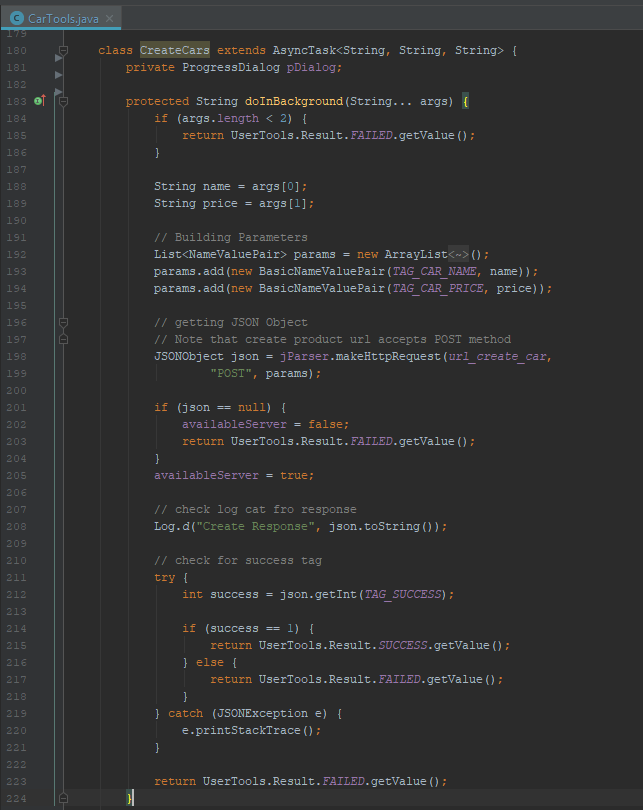


Rys. 12 Definicja adresów URL serwera

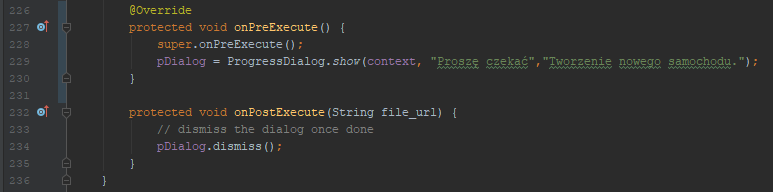
Na Rys. 12 podane adresy URL należą do domeny *carbeaver.hekko24.pl*. Jest to tymczasowy hosting online zarezerwowany na potrzeby prezentacji systemu „CarBeaver” z dostępem do Internetu. Jeśli serwer zostanie uruchomiony na lokalnej maszynie (np. w celu przeprowadzenia lokalnie testów), domenę tę można zmienić na adres *10.0.2.2*, który jest adresem IP dla localhost wirtualnych emulatorów Androida.

## Asynchroniczne wykonywanie zadań

Sposób ten został wykorzystany do wszystkich metod połączeń między klientem a serwerem. W narzędziach utworzone zostały wewnętrzne klasy dziedziczące abstrakcyjną klasę „AsyncTask”. Klasa ta daje możliwość łatwego i prawidłowego korzystania z wątków interfejsu użytkownika. Umożliwia wykonywanie operacji w tle i publikowanie wyników w wątku interfejsu użytkownika bez konieczności manualnego manipulowania wątkami.



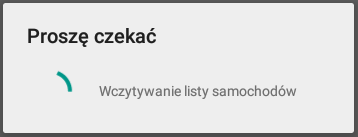
Rys. 13 Metoda wykonywana asynchroniczne



Rys. 14 Metody pre i post wykonawcze

Przed wykonywaniem głównej metody realizującej zadanie asynchronicznie, uruchamiana się metoda onPreExecute(). Ma ona za zadanie wywołanie preloadera informującego o wykonywanych czynnościach.

Po wykonaniu głównej metody asynchronicznej, uruchamiana jest metoda onPostExecute(), odpowiedzialna za odwołanie preloadera.

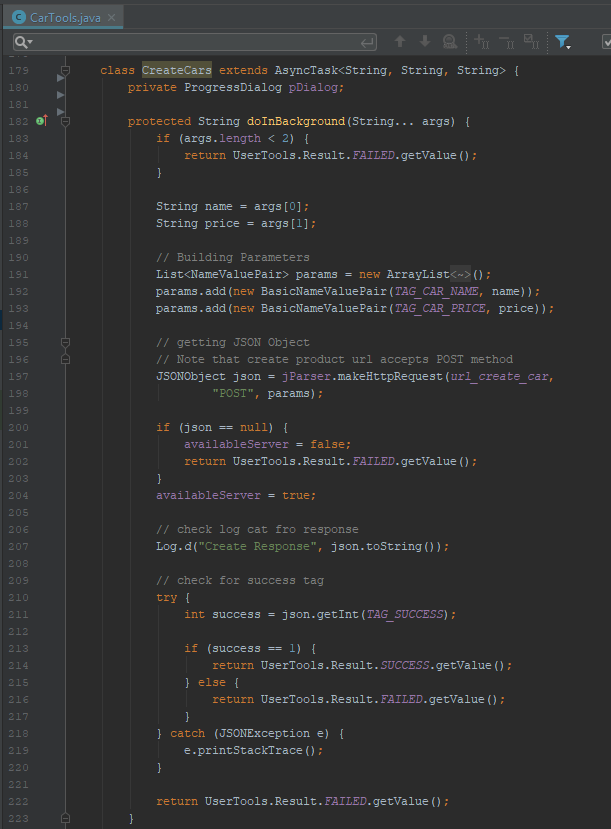


Rys. 15 Preloader uruchamiany przez zadania asynchroniczne

## Połączenie klient – serwer

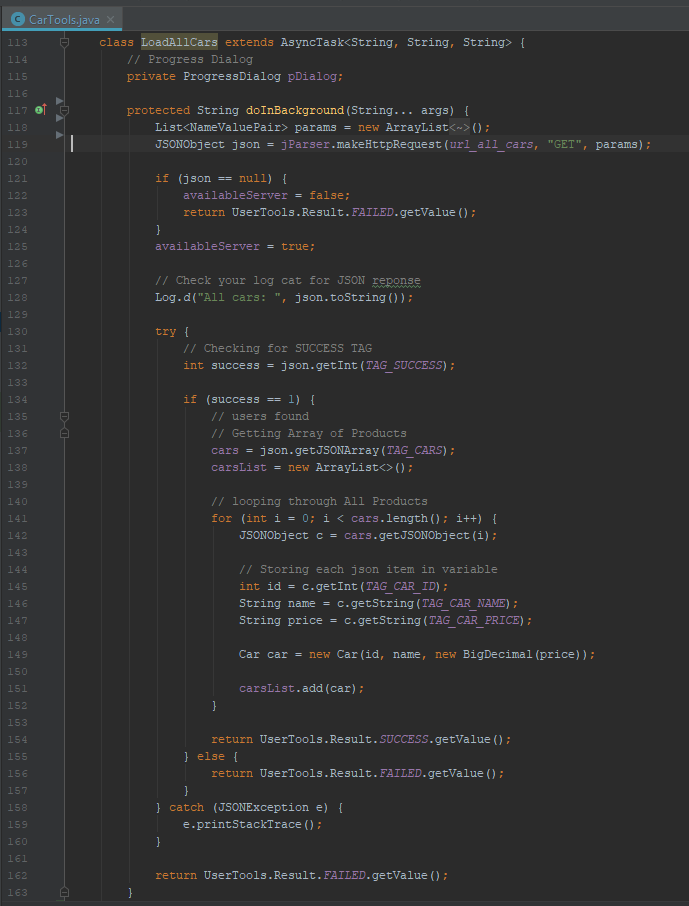
Połączenie klienta z serwerem odbywa się za pomocą protokołu HTTP. Tak jak opisane wcześniej, wykorzystywane są adresy URL, które są końcowym punktem serwera odpowiedzialnym za wykonywanie operacji modelu CRUD (Rys. 12). Połączenie to odbywa się za pomocą zadań asynchronicznych, a w zasadzie klas asynchronicznych, dzięki czemu dane przesyłane są w „tle” działania aplikacji.

Dane z klienta do serwera przesyłane są za pomocą metod protokołu HTTP, czyli GET oraz POST.



Rys. 16 Metoda wykonująca żądanie do serwera metodą POST

Metoda buduje listę parametrów, następnie przez obiekt jParser wykonuje żądanie HTTP na konkretny adres URL, wybraną metodę POST lub GET dołączając listę zbudowanych parametrów. Dostępny serwer zawsze odpowiada w formacie JSON z komunikatem czy żądanie zakończyło się powodzeniem. Jeśli żądaniem było pobranie jakiejś struktury danych np. listy dostępnej floty pojazdów, są one pobierane iteracyjnie, zamieniane na obiekty i dołączane do kolekcji.

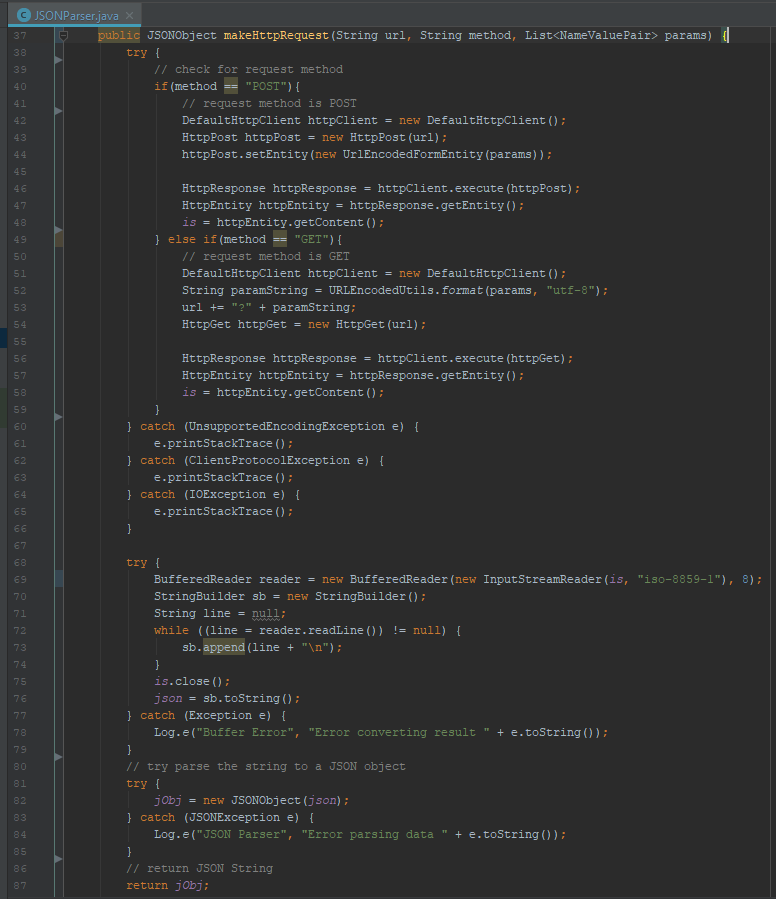


Rys. 17 Metoda wykonująca żądanie do serwera metodą GET

## Tworzenie żądania HTTP

Za tworzenie żądania odpowiada obiekt klasy JSONParser i jego metoda makeHttpRequest(). Przyjmuje ona adres URL na jaki zostanie wykonane żądanie, metodę żądania oraz listę dołączonych parametrów. W zależności od metody budowany jest strumień wejściowy (ang. Input Stream), który jest wynikiem wykonanego żądania oraz, z którego odczytywany jest tekst liniami. Całkowicie odczytany strumień daje kompletny tekst typu String w formacie JSON. Jest to wartość zwracana przez tę metodę i reprezentuje odpowiedź serwera na zadane żądanie.

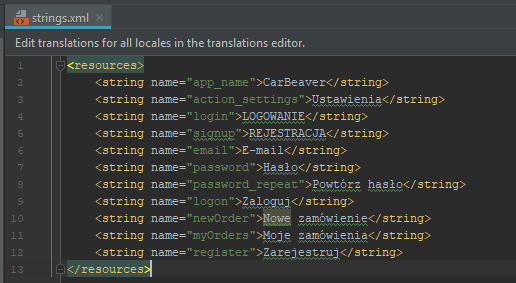
# 



Rys. 18 Wykonywanie żądania HTTP i odpowiedź JSON

## Moduł zasobów

Zasoby dostarczają konkretne wartości, oraz ustawienia aplikacji mobilnej. Głównie definiuje układy interfejs GUI, teksty etykiet, wykorzystywane kolory czy grafiki. Przeniesienie tych informacji do zewnętrznych plików XML stwarza, że aplikacja staje się bardzo uniwersalna i prosta pod kątem zmian. Bez ingerowania w kod źródłowy aplikacji, w zależności od sytuacji można definiować całkowicie inny zestaw zasobów. Jest to dużą zaletą w sytuacji, gdy np. chcemy aby aplikacja była wielojęzyczna.



Rys. 19 Zasób etykiet GUI – strings.xml

# Podręcznik użytkownika

## Rejestracja

Rejestracja nowego użytkownika dostępna jest pod przyciskiem „Zarejestruj” w głównym oknie aplikacji. Aby utworzyć nowe konto użytkownik musi podać adres e-mail oraz dwukrotnie podać hasło. Jeśli nie istnieje użytkownik o podanym adresie e-mail, rejestracja przebiega pomyślnie. W przeciwnym razie system prosi o poprawienie danych.



Rys. 20 Formularz rejestracji konta

## Logowanie do systemu

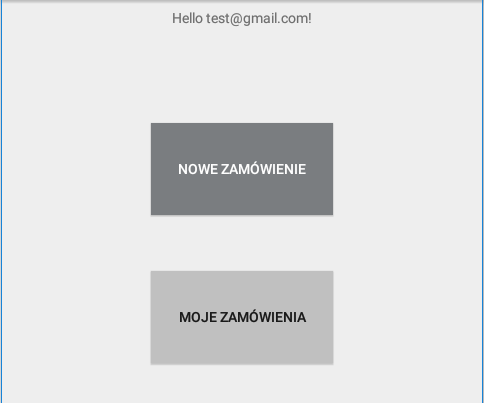
Pierwszym oknem aplikacji jest formularz umożliwiający zalogowanie się do swojego konta. W tym formularzu system prosi użytkownika o podanie swojego adresu e-mail oraz hasła. Jeśli urządzenie posiada połączenie z Internetem, a użytkownik zostanie pomyślnie zweryfikowany, zostanie wyświetlony ekran bieżącego użytkownika.



Rys. 21 Ekran logowania

## Główne okno użytkownika

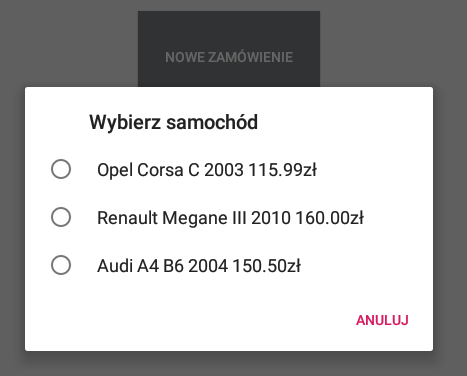
Po zalogowaniu użytkownik ma do wyboru wykonanie dwóch operacji: złożenia nowego zamówienia oraz podgląd aktualnych wypożyczeń. Akcje te dostępne są pod dwoma widocznymi na Rys. 22 przyciskami.



Rys. 22 Ekran użytkownika

## Składanie wypożyczenia

Po wybraniu przycisku „Nowe zamówienie”, użytkownik proszony jest o wybór jednego z dostępnych samochodów.

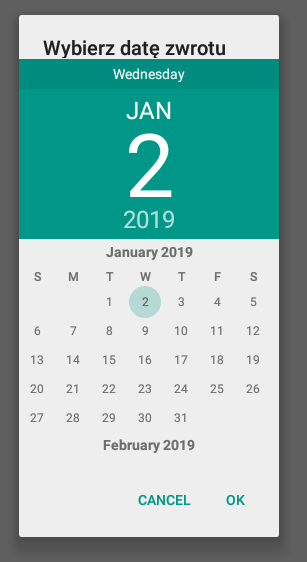


Rys. 23 Wybór pojazdu

Następnie proszony jest o wybór daty wypożyczenia, oraz w kolejnym oknie daty zwrotu.



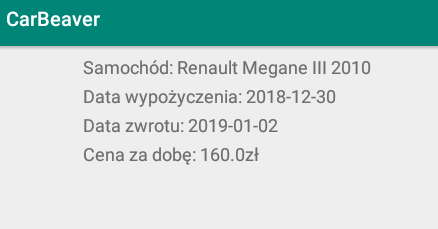
Rys. 24 Wybór daty wypożyczenia



Rys. 25 Wybór daty zwrotu

## Historia wypożyczeń

Po wybraniu przycisku „Moje zamówienia” dostępna jest lista złożonych wypożyczeń. Zawiera ona dane o wypożyczonym pojeździe, jego cenę za dobę oraz okres wypożyczenia pojazdu.



Rys. 26 Historia wypożyczeń

# 

# Literatura (źródła internetowe)

*http://math.uni.lodz.pl/~kubinski/android/w1.pdf* , [dostęp: 12. 2018]

*https://developer.android.com/guide /*, [dostęp: 12. 2018]

*http://php.net/manual/pl/function.json-encode.php* , [dostęp: 12. 2018]

*https://www.simplifiedcoding.net/android-mysql-tutorial-to-perform-basic-crud-operation/* , [dostęp: 12. 2018]

*https://javastart.pl/baza-wiedzy/darmowy-kurs-android/narzedzia/asynctask* , [dostęp: 12. 2018]

*https://guides.codepath.com/android/Creating-and-Executing-Async-Tasks* , [dostęp: 12. 2018]

# Spis ilustracji

[Rys. 1 Schemat ERD bazy danych 3](#_Toc533851253)

[Rys. 2 Odpowiedź serwera w formacie JSON 4](#_Toc533851254)

[Rys. 3 Struktura katalogów serwera PHP 5](#_Toc533851255)

[Rys. 4 Plik z danymi do połączenia z bazą danych (db\_conf.php) 5](#_Toc533851256)

[Rys. 5 Klasa otwierająca połączenie bazodanowe (db\_connect.php) 5](#_Toc533851257)

[Rys. 6 Przykładowy model CRUD serwera 6](#_Toc533851258)

[Rys. 7 Moduły aplikacji Android „CarBeaver" 7](#_Toc533851259)

[Rys. 8 AndroidManifest.xml – cechy i komponenty aplikacji 7](#_Toc533851260)

[Rys. 9 Przekazywania dodatkowych danych do innej aktywności 8](#_Toc533851261)

[Rys. 10 Klasa implementująca interfejs Parcelable 8](#_Toc533851262)

[Rys. 11 Odbieranie przekazanych danych z intencji 9](#_Toc533851263)

[Rys. 12 Definicja adresów URL serwera 9](#_Toc533851264)

[Rys. 13 Metoda wykonywana asynchroniczne 10](#_Toc533851265)

[Rys. 14 Metody pre i post wykonawcze 10](#_Toc533851266)

[Rys. 15 Preloader uruchamiany przez zadania asynchroniczne 10](#_Toc533851267)

[Rys. 16 Metoda wykonująca żądanie do serwera metodą POST 11](#_Toc533851268)

[Rys. 17 Metoda wykonująca żądanie do serwera metodą GET 12](#_Toc533851269)

[Rys. 18 Wykonywanie żądania HTTP i odpowiedź JSON 13](#_Toc533851270)

[Rys. 19 Zasób etykiet GUI – strings.xml 13](#_Toc533851271)

[Rys. 20 Formularz rejestracji konta 14](#_Toc533851272)

[Rys. 21 Ekran logowania 14](#_Toc533851273)

[Rys. 22 Ekran użytkownika 15](#_Toc533851274)

[Rys. 23 Wybór pojazdu 15](#_Toc533851275)

[Rys. 24 Wybór daty wypożyczenia 15](#_Toc533851276)

[Rys. 25 Wybór daty zwrotu 15](#_Toc533851277)

[Rys. 26 Historia wypożyczeń 16](#_Toc533851278)