



**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
2.	Charakterystyka programu.....	3
2.1.	Przeznaczenie programu	3
2.2.	Przegląd zastosowań programu	3
3.	Struktura programu – serwer.....	3
3.1.	Dane wykorzystywane przez program	3
3.2.	Baza danych.....	3
3.3.	Obsługa bazy danych	4
3.4.	Struktura serwera.....	4
3.5.	Konfiguracja połączenia z bazą danych	5
3.6.	Komunikacja klient – serwer	6
4.	Struktura programu – klient.....	6
4.1.	Informacje ogólne	6
4.2.	Podział na moduły	6
4.3.	Moduł aktywności	7
4.4.	Moduł modeli	8
4.5.	Przekazywanie obiektów między modułami	8
4.6.	Moduł narzędzi.....	9
4.7.	Asynchroniczne wykonywanie zadań.....	9
4.8.	Połączenie klient – serwer.....	10
4.9.	Tworzenie żądania HTTP	12
4.10.	Moduł zasobów	13
5.	Podręcznik użytkownika	14
5.1.	Rejestracja	14
5.2.	Logowanie do systemu.....	14
5.3.	Główne okno użytkownika	15
5.4.	Składanie wypożyczenia	15
5.5.	Historia wypożyczeń.....	16
6.	Literatura (źródła internetowe).....	17
7.	Spis ilustracji.....	17

1. 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.

2. Charakterystyka programu

2.1. 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.

2.2. 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 udostępnić odpłatnie swoje pojazdy na określonych warunkach.

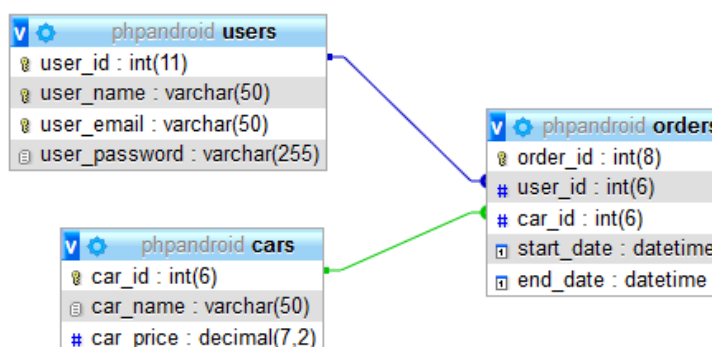
3. Struktura programu – serwer

3.1. 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.

3.2. Baza danych

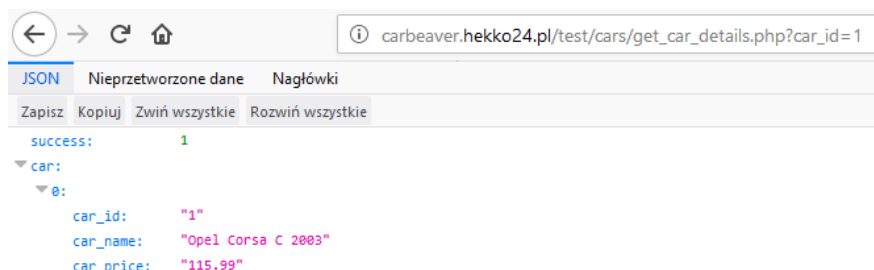
Do przechowywania informacji użyta została baza danych MySQL. Jej struktura to relacyjny model tabel powiązanych ze sobą. Gromadzi wszystkie dane, niezbędne do prawidłowego funkcjonowania aplikacji. Udostępniona w Internecie pozwala na spójne działanie wszystkich zainstalowanych aplikacji, które mają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



Rys. 1 Schemat ERD bazy danych

3.3. 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

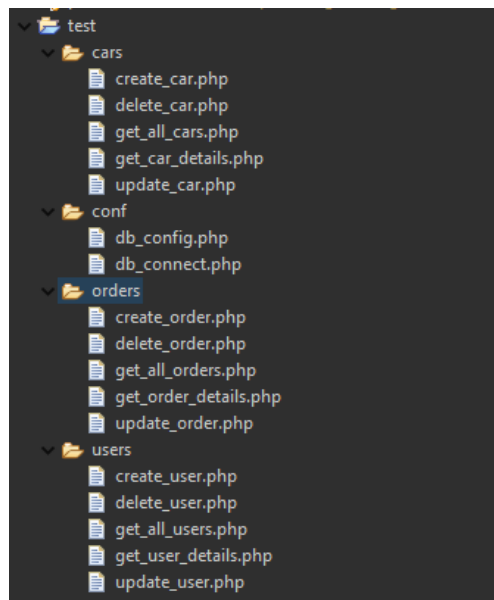
3.4. Struktura serwera

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 tabeli,
- 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 pożądaną sposób całą strukturę katalogów i plików należy umieścić w katalogu o nazwie „test”, w głównym folderze serwera PHP.

3.5. 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.

```

1 <?php
2 define('DB_USER', "root"); // db user
3 define('DB_PASSWORD', ""); // db password (mention your db password here)
4 define('DB_DATABASE', "phpandroid"); // database name
5 define('DB_SERVER', "localhost"); // db server
6 ?>

```

Rys. 4 Plik z danymi do połączenia z bazą danych (db_conf.php)

```

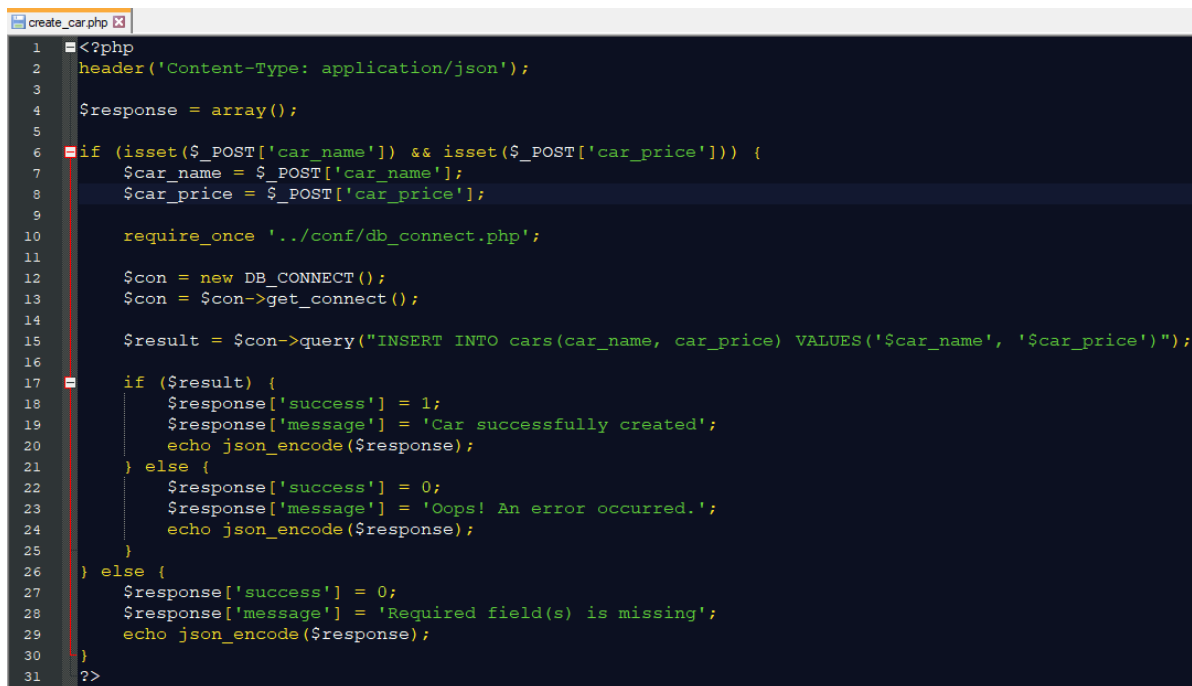
1 <?php
2 class DB_CONNECT {
3     private $con;
4
5     function __construct()
6     {
7         require_once 'db_config.php';
8
9         $this->con = new mysqli(DB_SERVER, DB_USER, DB_PASSWORD, DB_DATABASE);
10
11         if($this->con->connect_errno > 0) {
12             die('Unable to connect to database [' . $this->con->connect_error . ']');
13         }
14     }
15     public function get_connect() {
16         return $this->con;
17     }
18     public function close() {
19         mysqli_close($this->con);
20     }
21 }
22 ?>

```

Rys. 5 Klasa otwierająca połączenie bazodanowe (db_connect.php)

3.6. 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).



```
1 <?php
2 header('Content-Type: application/json');
3
4 $response = array();
5
6 if (isset($_POST['car_name']) && isset($_POST['car_price'])) {
7     $car_name = $_POST['car_name'];
8     $car_price = $_POST['car_price'];
9
10    require_once '../conf/db_connect.php';
11
12    $con = new DB_CONNECT();
13    $con = $con->get_connect();
14
15    $result = $con->query("INSERT INTO cars(car_name, car_price) VALUES('$car_name', '$car_price')");
16
17    if ($result) {
18        $response['success'] = 1;
19        $response['message'] = 'Car successfully created';
20        echo json_encode($response);
21    } else {
22        $response['success'] = 0;
23        $response['message'] = 'Oops! An error occurred.';
24        echo json_encode($response);
25    }
26 } else {
27     $response['success'] = 0;
28     $response['message'] = 'Required field(s) is missing';
29     echo json_encode($response);
30 }
31 ?>
```

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.

4. Struktura programu – klient

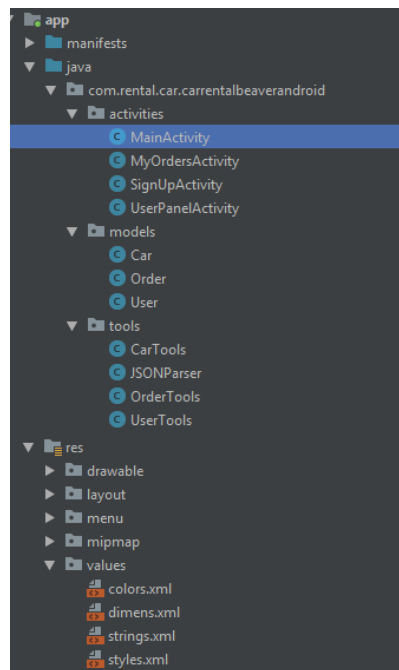
4.1. Informacje ogólne

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.

4.2. Podział na moduły

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

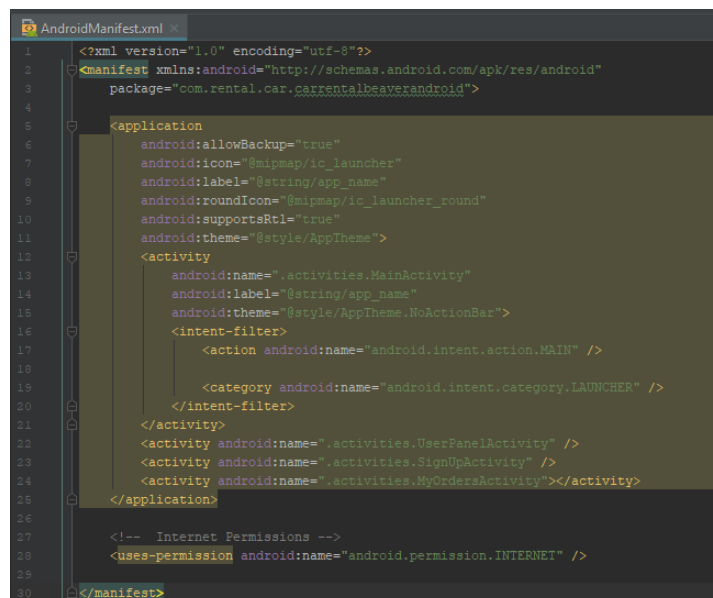
- Aktywności – znajdują się tutaj klasy Javy, które obsługują widoki wyświetlane w aplikacji (GUI),
- Modele – zbiór klas, które są odpowiedzialne za reprezentację konkretnych zestawu danych i obiektów w aplikacji,
- Narzędzia – klasy wykorzystywane do połączeń z serwerem,
- Zasoby – zbiór plików definiujących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



Rys. 7 Moduły aplikacji Android „CarBeaver”

4.3. 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

4.4. 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.

4.5. Przekazywanie obiektów między modułami

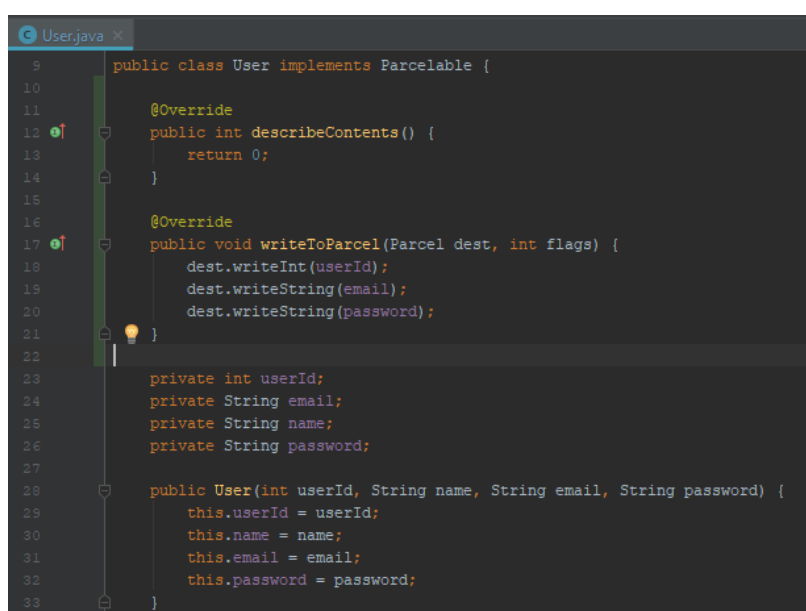
Pojedyncze dane, kolekcje oraz obiekty mogą być przekazywane do innej aktywności w momencie jej przygotowania, a następnie uruchomienia. Nową aktywność uruchamia się, budując nową intencję. Do niej dołączane są dodatkowe dane (metoda: `putExtra()`), które kolejno mogą zostać wykorzystane po jej uruchomieniu.



```
79 private void login() {
80     if (loggedUser != null) {
81         Log.d("Logon", "Found logged user");
82         Intent intent = new Intent(MainActivity.this, UserProfileActivity.class);
83         intent.putExtra("logged_user", loggedUser);
84         startActivity(intent);
85         finish();
86     } else {
87         Snackbar.make(findViewById(R.id.logonButton), "Błędny login lub hasło.", Snackbar.LENGTH_LONG)
88             .show();
89         List<User> allUsers = userTools.getAllUsers();
90         for (User user : allUsers) {
91             Log.d("DATA", user.getUserId() + " " + user.getEmail());
92         }
93     }
94 }
```

Rys. 9 Przekazywania dodatkowych danych do innej aktywności

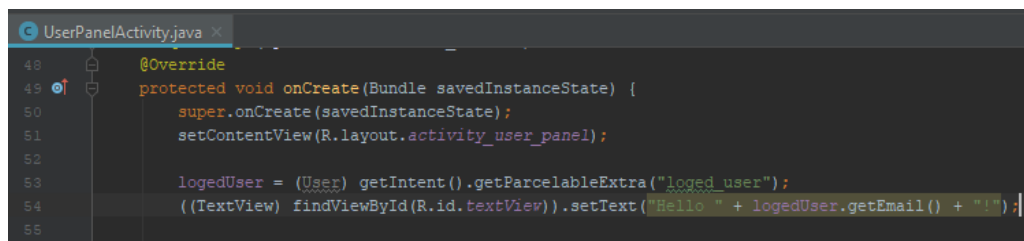
W prosty sposób można przekazywać pojedyncze dane standardowego typu danych. Aby przekazać obiekt do innej aktywności musi on implementować wbudowany interfejs `Parcelable` lub `Serializable`. W pierwszym przypadku konieczna jest implementacja jego abstrakcyjnych metod `describeContents()` oraz `writeToParcel()`.



```
9 public class User implements Parcelable {
10
11     @Override
12     public int describeContents() {
13         return 0;
14     }
15
16     @Override
17     public void writeToParcel(Parcel dest, int flags) {
18         dest.writeInt(userId);
19         dest.writeString(email);
20         dest.writeString(password);
21     }
22
23     private int userId;
24     private String email;
25     private String name;
26     private String password;
27
28     public User(int userId, String name, String email, String password) {
29         this.userId = userId;
30         this.name = name;
31         this.email = email;
32         this.password = password;
33     }
34 }
```

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.



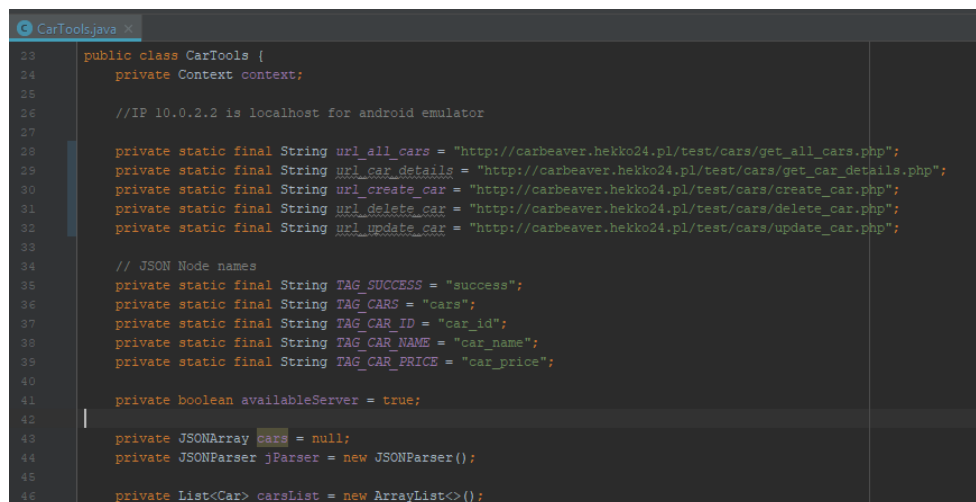
```
48      @Override
49      protected void onCreate(Bundle savedInstanceState) {
50          super.onCreate(savedInstanceState);
51          setContentView(R.layout.activity_user_panel);
52
53          loggedUser = (User) getIntent().getParcelableExtra("logged_user");
54          ((TextView) findViewById(R.id.text_view)).setText("Hello " + loggedUser.getEmail() + "!");
55      }
```

Rys. 11 Odbieranie przekazanych danych z intencji

4.6. 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.



```
23      public class CarTools {
24          private Context context;
25
26          //IP 10.0.2.2 is localhost for android emulator
27
28          private static final String url_all_cars = "http://carbeaver.hekko24.pl/test/cars/get_all_cars.php";
29          private static final String url_car_details = "http://carbeaver.hekko24.pl/test/cars/get_car_details.php";
30          private static final String url_create_car = "http://carbeaver.hekko24.pl/test/cars/create_car.php";
31          private static final String url_delete_car = "http://carbeaver.hekko24.pl/test/cars/delete_car.php";
32          private static final String url_update_car = "http://carbeaver.hekko24.pl/test/cars/update_car.php";
33
34          // JSON Node names
35          private static final String TAG_SUCCESS = "success";
36          private static final String TAG_CARS = "cars";
37          private static final String TAG_CAR_ID = "car_id";
38          private static final String TAG_CAR_NAME = "car_name";
39          private static final String TAG_CAR_PRICE = "car_price";
40
41          private boolean availableServer = true;
42
43          private JSONArray cars = null;
44          private JSONParser jParser = new JSONParser();
45
46          private List<Car> carsList = new ArrayList<>();
47      }
```

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.

4.7. 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.

```

180 class CreateCars extends AsyncTask<String, String, String> {
181     private ProgressDialog pDialog;
182
183     protected String doInBackground(String... args) {
184         if (args.length < 2) {
185             return UserTools.Result.FAILED.getValue();
186         }
187
188         String name = args[0];
189         String price = args[1];
190
191         // Building Parameters
192         List<NameValuePair> params = new ArrayList<>();
193         params.add(new BasicNameValuePair(TAG_CAR_NAME, name));
194         params.add(new BasicNameValuePair(TAG_CAR_PRICE, price));
195
196         // getting JSON Object
197         // Note that create product url accepts POST method
198         JSONObject json = jParser.makeHttpRequest(url_create_car,
199             "POST", params);
200
201         if (json == null) {
202             availableServer = false;
203             return UserTools.Result.FAILED.getValue();
204         }
205         availableServer = true;
206
207         // check log cat fro response
208         Log.d("Create Response", json.toString());
209
210         // check for success tag
211         try {
212             int success = json.getInt(TAG_SUCCESS);
213
214             if (success == 1) {
215                 return UserTools.Result.SUCCESS.getValue();
216             } else {
217                 return UserTools.Result.FAILED.getValue();
218             }
219         } catch (JSONException e) {
220             e.printStackTrace();
221         }
222
223         return UserTools.Result.FAILED.getValue();
224     }

```

Rys. 13 Metoda wykonywana asynchronicznie

```

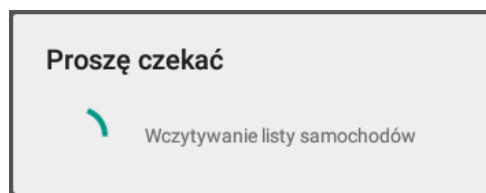
225 @Override
226 protected void onPreExecute() {
227     super.onPreExecute();
228     pDialog = ProgressDialog.show(context, "Proszę czekać", "Tworzenie nowego samochodu.");
229 }
230
231 protected void onPostExecute(String file_url) {
232     // dismiss the dialog once done
233     pDialog.dismiss();
234 }
235
236 }

```

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.

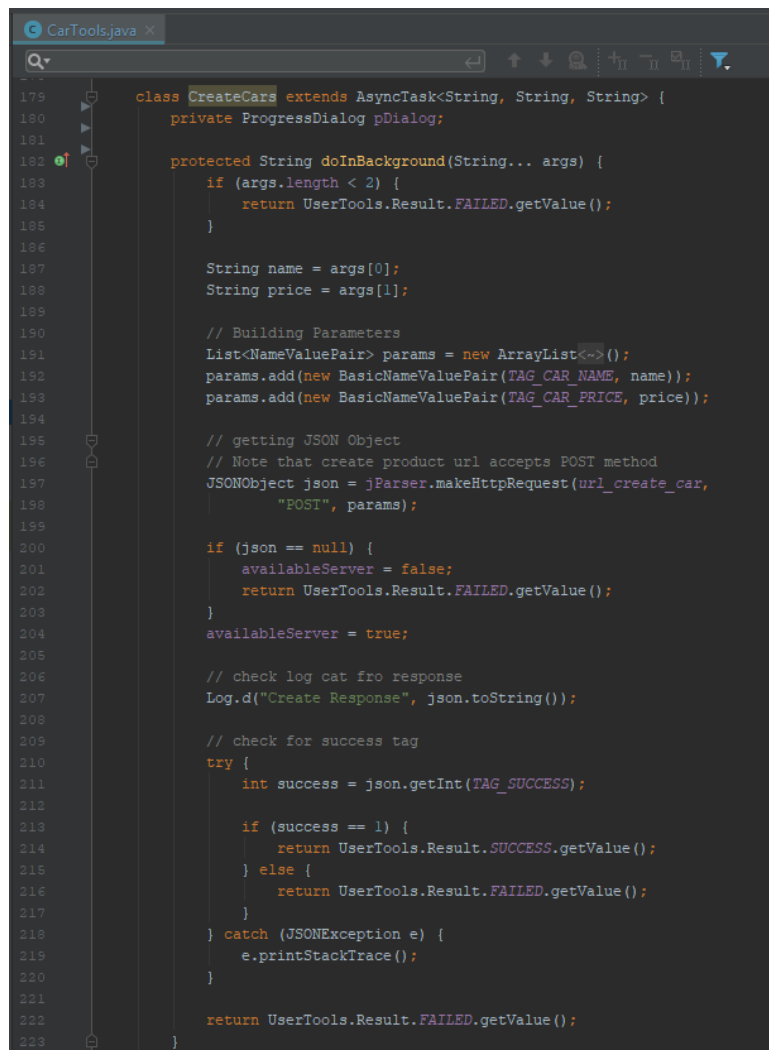


Rys. 15 Preloader uruchamiany przez zadania asynchroniczne

4.8. 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.



```

179 class CreateCars extends AsyncTask<String, String, String> {
180     private ProgressDialog pDialog;
181
182     protected String doInBackground(String... args) {
183         if (args.length < 2) {
184             return UserTools.Result.FAILED.getValue();
185         }
186
187         String name = args[0];
188         String price = args[1];
189
190         // Building Parameters
191         List<NameValuePair> params = new ArrayList<>();
192         params.add(new BasicNameValuePair(TAG_CAR_NAME, name));
193         params.add(new BasicNameValuePair(TAG_CAR_PRICE, price));
194
195         // getting JSON Object
196         // Note that create product url accepts POST method
197         JSONObject json = jParser.makeHttpRequest(url_create_car,
198             "POST", params);
199
200         if (json == null) {
201             availableServer = false;
202             return UserTools.Result.FAILED.getValue();
203         }
204         availableServer = true;
205
206         // check log cat fro response
207         Log.d("Create Response", json.toString());
208
209         // check for success tag
210         try {
211             int success = json.getInt(TAG_SUCCESS);
212
213             if (success == 1) {
214                 return UserTools.Result.SUCCESS.getValue();
215             } else {
216                 return UserTools.Result.FAILED.getValue();
217             }
218         } catch (JSONException e) {
219             e.printStackTrace();
220         }
221
222         return UserTools.Result.FAILED.getValue();
223     }

```

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 metodą 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.

```

113 class LoadAllCars extends AsyncTask<String, String, String> {
114     // Progress Dialog
115     private ProgressDialog pDialog;
116
117     protected String doInBackground(String... args) {
118         List<NameValuePair> params = new ArrayList<>();
119         JSONObject json = jParser.makeHttpRequest(url_all_cars, "GET", params);
120
121         if (json == null) {
122             availableServer = false;
123             return UserTools.Result.FAILED.getValue();
124         }
125         availableServer = true;
126
127         // Check your log cat for JSON response
128         Log.d("All cars: ", json.toString());
129
130         try {
131             // Checking for SUCCESS TAG
132             int success = json.getInt(TAG_SUCCESS);
133
134             if (success == 1) {
135                 // users found
136                 // Getting Array of Products
137                 cars = json.getJSONArray(TAG_CARS);
138                 carsList = new ArrayList<>();
139
140                 // looping through All Products
141                 for (int i = 0; i < cars.length(); i++) {
142                     JSONObject c = cars.getJSONObject(i);
143
144                     // Storing each json item in variable
145                     int id = c.getInt(TAG_CAR_ID);
146                     String name = c.getString(TAG_CAR_NAME);
147                     String price = c.getString(TAG_CAR_PRICE);
148
149                     Car car = new Car(id, name, new BigDecimal(price));
150
151                     carsList.add(car);
152                 }
153
154                 return UserTools.Result.SUCCESS.getValue();
155             } else {
156                 return UserTools.Result.FAILED.getValue();
157             }
158         } catch (JSONException e) {
159             e.printStackTrace();
160         }
161
162         return UserTools.Result.FAILED.getValue();
163     }

```

Rys. 17 Metoda wykonująca żądanie do serwera metodą GET

4.9. 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.

```

37 public JSONObject makeHttpRequest(String url, String method, List<NameValuePair> params) {
38     try {
39         // check for request method
40         if(method == "POST"){
41             // request method is POST
42             DefaultHttpClient httpClient = new DefaultHttpClient();
43             HttpPost httpPost = new HttpPost(url);
44             httpPost.setEntity(new UrlEncodedFormEntity(params));
45
46             HttpResponse httpResponse = httpClient.execute(httpPost);
47             HttpEntity httpEntity = httpResponse.getEntity();
48             is = httpEntity.getContent();
49         } else if(method == "GET"){
50             // request method is GET
51             DefaultHttpClient httpClient = new DefaultHttpClient();
52             String paramString = URLEncodedUtils.format(params, "utf-8");
53             url += "?" + paramString;
54             HttpGet httpGet = new HttpGet(url);
55
56             HttpResponse httpResponse = httpClient.execute(httpGet);
57             HttpEntity httpEntity = httpResponse.getEntity();
58             is = httpEntity.getContent();
59         }
60     } catch (UnsupportedEncodingException e) {
61         e.printStackTrace();
62     } catch (ClientProtocolException e) {
63         e.printStackTrace();
64     } catch (IOException e) {
65         e.printStackTrace();
66     }
67
68     try {
69         BufferedReader reader = new BufferedReader(new InputStreamReader(is, "iso-8859-1"), 8);
70         StringBuilder sb = new StringBuilder();
71         String line = null;
72         while ((line = reader.readLine()) != null) {
73             sb.append(line + "\n");
74         }
75         is.close();
76         json = sb.toString();
77     } catch (Exception e) {
78         Log.e("Buffer Error", "Error converting result " + e.toString());
79     }
80     // try parse the string to a JSON object
81     try {
82         jsonObj = new JSONObject(json);
83     } catch (JSONException e) {
84         Log.e("JSON Parser", "Error parsing data " + e.toString());
85     }
86     // return JSON String
87     return jsonObj;

```

Rys. 18 Wykonywanie żądania HTTP i odpowiedź JSON

4.10. 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.

```

1 <resources>
2   <string name="app_name">CarBeaver</string>
3   <string name="action_settings">Ustawienia</string>
4   <string name="login">LOGOWANIE</string>
5   <string name="signup">REJESTRACJA</string>
6   <string name="email">E-mail</string>
7   <string name="password">Hasło</string>
8   <string name="password_repeat">Powtórz hasło</string>
9   <string name="login">Zaloguj</string>
10  <string name="newOrder">Nowe zamówienie</string>
11  <string name="myOrders">Moje zamówienia</string>
12  <string name="register">Zarejestruj</string>
13 </resources>

```

Rys. 19 Zasób etykiet GUI – strings.xml

5. Podręcznik użytkownika

5.1. 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.



REJESTRACJA

E-mail

Hasło

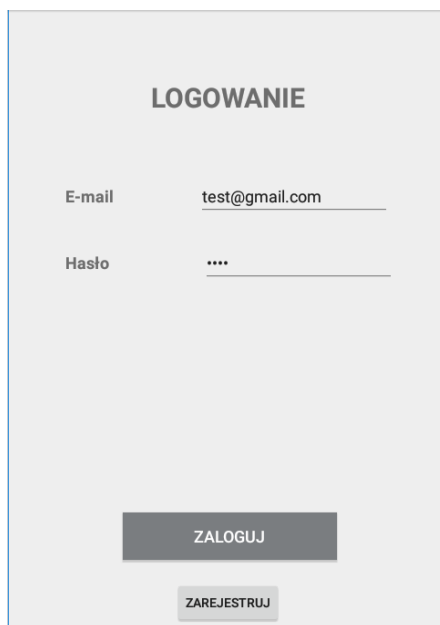
Powtórz hasło

ZAREJESTRUJ

Rys. 20 Formularz rejestracji konta

5.2. 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.



LOGOWANIE

E-mail

Hasło

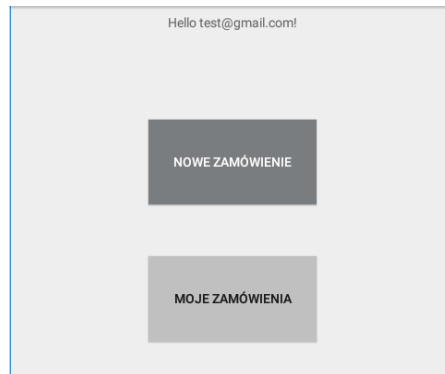
ZALOGUJ

ZAREJESTRUJ

Rys. 21 Ekran logowania

5.3. Główne okno użytkownika

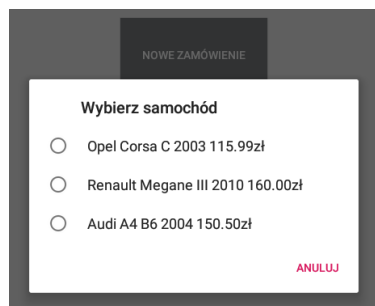
Po zalogowaniu użytkownik ma do wyboru wykonanie dwóch operacji: złożenia nowego zamówienia oraz podgląd aktualnych wypożyczeń. Akcje te dostępne są pod dwoma widocznymi na Rys. 22 przyciskami.



Rys. 22 Ekran użytkownika

5.4. Składanie wypożyczenia

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

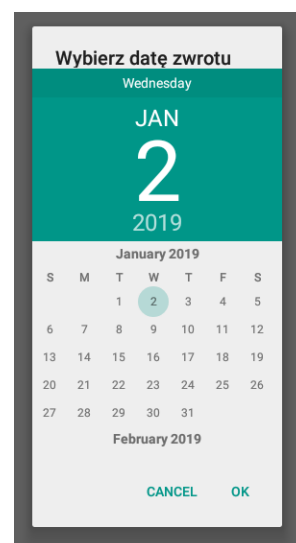


Rys. 23 Wybór pojazdu

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



Rys. 24 Wybór daty wypożyczenia



Rys. 25 Wybór daty zwrotu

5.5. 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.

CarBeaver
Samochód: Renault Megane III 2010
Data wypożyczenia: 2018-12-30
Data zwrotu: 2019-01-02
Cena za dobę: 160.0zł

Rys. 26 Historia wypożyczeń

6. 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]

7. Spis ilustracji

Rys. 1 Schemat ERD bazy danych	3
Rys. 2 Odpowiedź serwera w formacie JSON	4
Rys. 3 Struktura katalogów serwera PHP	5
Rys. 4 Plik z danymi do połączenia z bazą danych (db_conf.php)	5
Rys. 5 Klasa otwierająca połączenie bazodanowe (db_connect.php)	5
Rys. 6 Przykładowy model CRUD serwera	6
Rys. 7 Moduły aplikacji Android „CarBeaver”	7
Rys. 8 AndroidManifest.xml – cechy i komponenty aplikacji	7
Rys. 9 Przekazywania dodatkowych danych do innej aktywności	8
Rys. 10 Klasa implementująca interfejs Parcelable	8
Rys. 11 Odbieranie przekazanych danych z intencji	9
Rys. 12 Definicja adresów URL serwera	9
Rys. 13 Metoda wykonywana asynchronicznie	10
Rys. 14 Metody pre i post wykonawcze	10
Rys. 15 Preloader uruchamiany przez zadania asynchroniczne	10
Rys. 16 Metoda wykonująca żądanie do serwera metodą POST	11
Rys. 17 Metoda wykonująca żądanie do serwera metodą GET	12
Rys. 18 Wykonywanie żądania HTTP i odpowiedź JSON	13
Rys. 19 Zasób etykiet GUI – strings.xml	13
Rys. 20 Formularz rejestracji konta	14
Rys. 21 Ekran logowania	14
Rys. 22 Ekran użytkownika	15
Rys. 23 Wybór pojazdu	15
Rys. 24 Wybór daty wypożyczenia	15
Rys. 25 Wybór daty zwrotu	15
Rys. 26 Historia wypożyczeń	16