Reasonable eCar Sharing IT-Infrastructure Features

Students of Class 3DK 2015/2016 - BHAK-Weiz

Dr. Karl Widdmannstraße 40, 8160 Weiz, Austria

Abstract

Within the scope of the Erasmus+ EU-project „Green Car“, different car-sharing concepts were identified, listed, and compared. This was done, taking into account the special case of pure electrically powered cars. After the comparison of available concepts, the most useful car-rental features were identified and integrated into a fully functional prototype, to be able to test and identify useful and helpful features.

Thus, this paper presents in section_1 a survey to list and evaluate essential features and potential "nice to haves-" gimmicks. In doing so, all offered features are rated and potential missing features are named.

Based on a list of identified new features, section_2 is about the implementation and integration of these features into a functioning prototype. Since these applications have to interact with a central service center, a communication scenario is shown and a solutions is presented, which is based on an ASP.Net - SOAP interface.

Finally, in section_3, a summery is given and a roadmap for the second project year is given.

Co-funded by the Erasmus+ Programme of the European Union

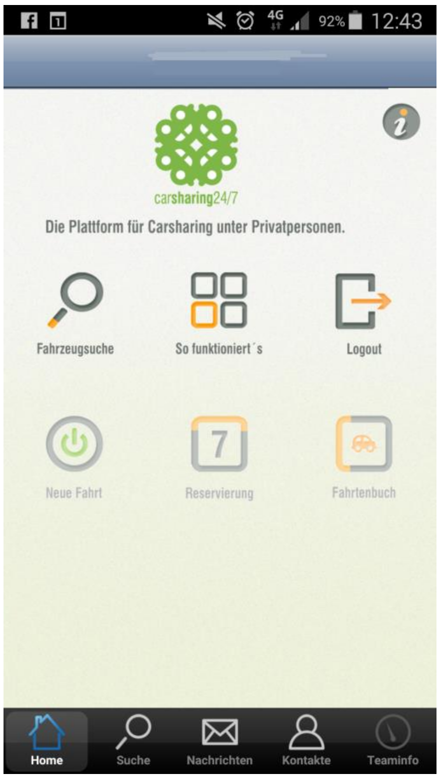
1. The Survey

A first survey was conducted to identify useful smartphone- and tablet-features on the one hand, but also web-portals on the other hand to identify essential features in the context of car-sharing. Unfortunately, not all features are given on the websites and we did not have time to install and test all available apps. Thus, only a short summary of car sharing companies and offered services is shown in the following table:

Feature	Car2Go	Cambio CarSharing	ZipCar	CarSharing24/7
Fixed car-stations	no	yes	yes	Depends to the renter
Flexible refundments	yes	yes	yes	Packet depending
Car-lending time	flexible	1h–30days	1h–7days	Packet depending
Making reservations via:		Key-safe with chip-card access	App or via card	Many solutions are possible - depending to the car-renter
Charging		Driven kilometer + period of occupation	80km for free, thereafter 0,2€ per km	

1.1. Carsharing24/7

A nice platform in the field of private- and commercial car sharing is provided by CarSharing24/7. The only problem with this service is that it is only available in three Austrian cities – Wien, Graz and Linz.



Similar to Caruso – a company located in Voralberg (Austria), Carsharing24/7 is a platform to rent the own car but also to organize the sharing of a car to go to work every morning. As an optional service, you can also add a vehicle-insurance, which can be balanced daily. This service is not very cheap, but necessary and will be an interesting business field in the near future.

As shown in figure 1a, the application offers four basic functions:

- The search for available cars
- The reservation of the favoured car
- The driver's logbook
- Data collection service

In addition, the app also gives assistance to the driver, if administrative or technical help is needed.

Figure 1a

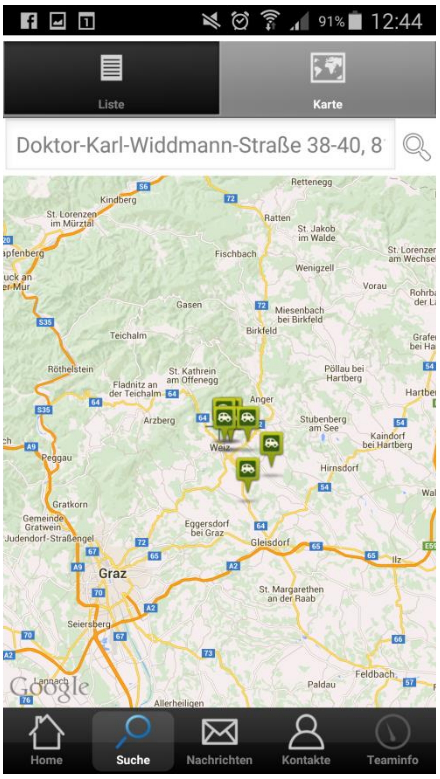
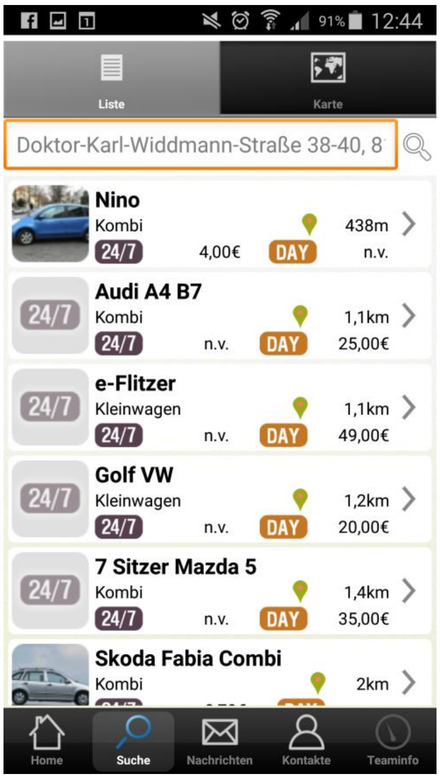


Figure 1b depicts all available cars on a map, if the own location is supplied in the address field. If a suitable car is found and if the marker is clicked, all car specific information is shown.

Figure 1b



Nino
Kombi 438m
24/7 4,00€ DAY n.v.

Audi A4 B7
Kombi 1,1km
24/7 n.v. DAY 25,00€

e-Flitzer
Kleinwagen 1,1km
24/7 n.v. DAY 49,00€

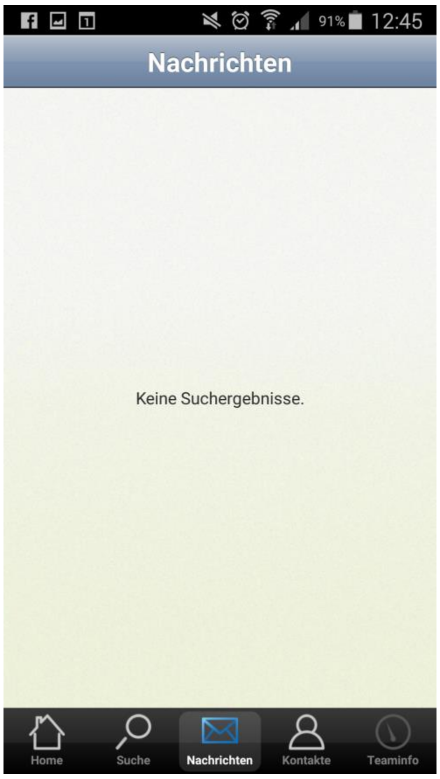
Golf VW
Kleinwagen 1,2km
24/7 n.v. DAY 20,00€

7 Sitzer Mazda 5
Kombi 1,4km
24/7 n.v. DAY 35,00€

Skoda Fabia Combi
Kombi 2km

If a list-view of available cars is preferred, the list of all nearby and available cars can be shown. By selecting a car, all relevant car-related information can be found. In this view, the search results are ordered by the distance to the driver’s current location.

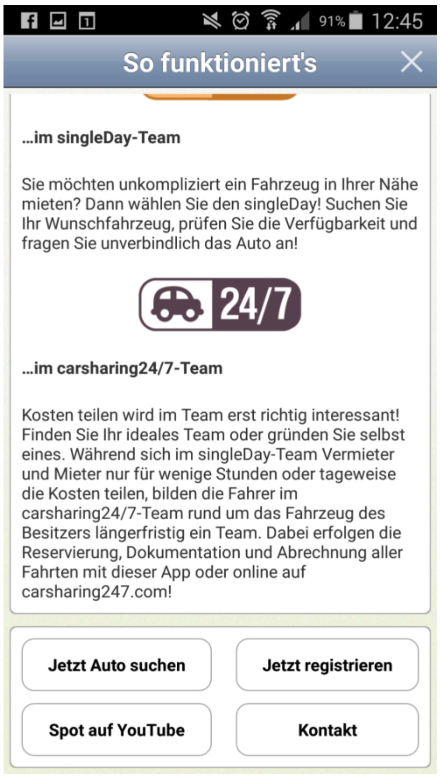
Figure 1c



Keine Suchergebnisse.

A very nice feature is the sending of short messages. Unfortunately, information can only be send from the service centre to the drivers. No mechanism is provided to send short messages to the service-center to inform about individual occurrences or if help is needed. Predefined messages, triggered by selecting one of a small set of listed messages, would make sense to save time in emergency situations or the breakdown of a car.

Figure 1d



…im singleDay-Team

Sie möchten unkompliziert ein Fahrzeug in Ihrer Nähe mieten? Dann wählen Sie den singleDay! Suchen Sie Ihr Wunschfahrzeug, prüfen Sie die Verfügbarkeit und fragen Sie unverbindlich das Auto an!

…im carsharing24/7-Team

Kosten teilen wird im Team erst richtig interessant! Finden Sie Ihr ideales Team oder gründen Sie selbst eines. Während sich im singleDay-Team Vermieter und Mieter nur für wenige Stunden oder tageweise die Kosten teilen, bilden die Fahrer im carsharing24/7-Team rund um das Fahrzeug des Besitzers längerfristig ein Team. Dabei erfolgen die Reservierung, Dokumentation und Abrechnung aller Fahrten mit dieser App oder online auf carsharing247.com!

The existence of an easy to use helpdesk service is absolutely essential. Simple assistance can be given by the help function of the application itself. In case of more complex questions, an automatic forwarding mechanism to the messaging service can be helpful and desireable.

Figure 1e

CarSharing24/7 does not offer the actual platform-SW - it is rather based on Ibola Mobility Solutions (IMS), who is a provider for customized and integral car sharing solutions. Provided functions ranges from automatic locking system (based on smartcards) to billing- and reporting services. Furthermore, the existence of an electronic driver's logbook is self-evident:

- Complete reservation- and booking system.
- Physical access control and locking mechanism based on chip cards.
- Complex billing system and clearing tool.
- Real-time reporting system (e.g. common state of the car or records of the amount of emitted carbon dioxide).
- Any combinations of car-technologies and property situations can be mapped by the system.
- Free-floating support (cross communal support).
- Multilingual and capable to support multi-clients.
- Interfaces to be able to support individual requirements.
- Electronic driver's log-books – for drivers and operators.
- Outsourcing – IMS can run the whole set of service if needed.

Software-Frontend

- A booking system is available for mobile- and desktop devices
- Parking positions of the cars, their battery-states and the states of fuel tanks can be retrieved automatically via network.
- Automatic calculation of remaining cruising radiuses (in the case of eCars).
- Extensive car-sharing services.
- Services can distinguish between business trips and private trips.
- Services can deal with private information (driver's logbook, costs, user profiles & users privacies).
- Individual branding and customizing (colour, logo and textual contents).

Software Backend

- Administration of user profiles, bank accounts, cars and fleets.
- Processing reservation data and treating of reservation changes.
- Electronic driver's log-book, which is accepted by the Austrian tax authority.
- Rich configuration settings (language, privacy, publicly accessible).
- Fleet settings are not limited to heterogeneous vehicles – motorcycles as well as eBikes and eScooter can also be integrated into the services.
- Complex billing-engine - supporting different scale of charges.
- Card management, card personalization and rollout of (branded) drivers ID-cards.
- Open interfaces to third party services (user management, accounting and statistics).
- Rich set of data export formats (PDF, Excel, Word, CSV, ...).
- API for White-Label customers.
- App to support fleet management tasks.

Hardware

- Automotive-certified onboard-box. DOPUX®1 compliant data formats.
- Rich data security services (to guarantee confidentiality and privacy by cryptographically secured data connections.
- Fast and simple integration into any type of car.
- Capturing car specific data like mileage, journey time and driven distances for the final accounting process.
- Free definable CAN-parameter for all capturing processes in realtime (e.g. state of onboard lighting system, etc. …).
- Supporting third party NFC-cards (e.g. discount cards, access cards, time recording cards etc. …).
- Theft protection.
- Automatic emergency calls via eCall.
- Synchronization of access credential (at least once the day).

1.2. Summary

In short, the list of the abovementioned features is comprehensive and seems to be complete. Very interesting is a ready to use hardware module, which can be directly connected to the CAN-bus of the car. Beyond that, the ability to read any card format and card systems can make this all-in-one device to a suitable tool for all purposes.

Given the fact that the integration of this device (or similar tools) is absolutely essential in the field of car-telemetry, car-access and car-billing services, the additional effort to integrate all aforementioned services would exceed the scope of our project. But since these features would be really nice to have, we think about low-cost solutions to get access to the needed data in the second year of the project.

Remarkable is the fact, that IMS does not support GPS and therefore an active fleet management – capable to track the movements of the cars or at least – is not possible, to determine each single care location. This is the point, where smartphones come into play, since all needed services are integral components of any kind of smart cell-phones.

Thus, the main idea of the proposed application is to use all these nice little features and make use of them to realize all in section 1.1 listed requirements enriched by the missing mechanisms:

- Detect the current car location, in emergency cases or to give support, but also in the case of car-deft
- Detect movements of the vehicle (to be able to generate user profiles)
- Use the above mentioned features to realize a real time fleet-tracking and fleet management system
- Determine the “safe-state” of an electric car in terms of the smallest distance to the next charging station

Since smartphones and tablets always support 3G- or LTE connection, all aforementioned information can be collected, calculated and sent to the service centre without any additional hardware component.

2. The App

One of the main Targets of the project is the design of a mobile application, which runs on a private cellphone or tablet, but will become an inseparable part of the vehicle, as soon as the driver is registered and the app is synchronized with the car. Since several security protocols are known and since designing new authentication protocols is not reasonable, we have to go without the integration of an adequate authentication mechanisms in the prototype. Thus, in a first step, we assume that the mobile device is already authenticated by the car and that the car was opened by an electronic locking mechanism (maybe based on certificates, etc. …).

The problem with authentication and locking mechanisms is, that some sort of hardware is needed in principle.

Based on results of the surveys, we identified the following essential requirements for the new application:

- To display the current position is obviously a vital requirement. On the one hand, we have to calculate the distances to all charging stations which are in reach and we have to support to find the shortest route. On the other hand, we need this feature anyway, since we need a real-time-service to identify the location of the car.
- Another obligatory feature is to display the location of registered charging stations. Since we always need the distances to the nearest stations - to be able to calculate the "safe state" of a car - we also need these locations to guide the drive to one of these stations by the shortest way. Calculation the "safe-state" in this context means to prevent the driver to have a breakdown due to low batteries. The application should be able to alert the driver and the service center when he is leaving the "safe-area".
- Besides assisting the driver in keeping the car in a "safe-state", the application can also assist in all communication processes between the car and the service center. In doing so, telemetric data can be sent together with the current position to the car service center, to be able to inform the driver about technical or other problems by using push services. In addition to this service-center-to-driver communication, already prepared emergency messages should be released by simple selecting and releasing one of a number of predefined emergency messages. Depending on these messages, the service-center will be able to react or to release rescue-missions.

Due to restricted time resources, we only start to think about transmitting telemetry-data, based on predefined hardware components. In doing so, we will present a first concept and a feasibility-study in the second year of the project.

2.1. Get Current Position

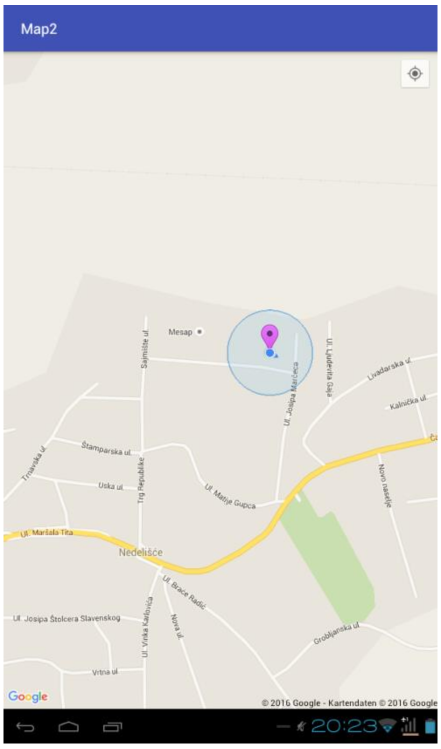
Starting with the standard google-maps Application - offered by Android-studio – we only had to add a mechanism to permanently check for location changes. Fortunately, the android developer group has got solutions for all possible problems – even so in the case of detecting location changes, based on a continuously determination of the current location. The core procedure is shown below. As soon as a connection to the GPS-System was found, the current location is stored as "myLastLocation". This measurement is done in an interval of 5 seconds. As soon as the measurement is finished, the current location is used to set a marker with predefined marker options (see figure 2).

```
@Override
public void onConnected(Bundle bundle) {
       Location mLastLocation =
       LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
    if (mLastLocation != null) {
        //place marker at current position
        mGoogleMap.clear();
        latLng = new LatLng(mLastLocation.getLatitude(),
                               mLastLocation.getLongitude());
        MarkerOptions markerOptions = new MarkerOptions();
        markerOptions.position(latLng);
        markerOptions.title("Current Position");

markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HU
        E_MAGENTA));
        mCurrLocation = mGoogleMap.addMarker(markerOptions);
    }

    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(5000); //5 seconds
    mLocationRequest.setFastestInterval(3000); //3 seconds
    mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);
    //mLocationRequest.setSmallestDisplacement(0.1F); //1/10 meter

    LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
    mLocationRequest, this);
}
```



This screenshot of the current car-loaction was take at ATON-Nacionalni gimnastički centar (Croatia)

Figure 2: Marker to show the current car-position.

2.2. Show Position of available Cars

To access the Google Maps servers with the Maps-API, we have to add a Maps-API key to our application. The key is free, and can be used for all applications that call the Maps-API. Furthermore, it supports an unlimited number of users and can be used for many application installations. The Maps-API key was received from the Google APIs Console by providing our application's signing certificate and its package name. Thereafter, the key was added to our application by adding an element to the application's AndroidManifest.xml file:

```
<meta-data android:name="com.google.android.maps.v2.API_KEY"
android:value="api-key-here"/>
```

To use Google Maps, some permission have to be set first to determine the map resolution:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<!-- The following two permissions are not required to use  Google Maps Android API v2, but
are recommended. -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

In your activity, we have to create the method that will initialize the map. Thus, we have to create a method name `setUpMap()`, this will configure our map object and set its listeners.

We also have to create a markerClickListener() inside this map, this will respond when we click a marker on the map, in this case, it will show an InfoWindow with the marker's details.

Finally, our activity looked like this:

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;

public class MainActivity extends Activity
{
    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        setUpMap();
    }

    private void setUpMap()
    {
        // Do a null check to confirm that we have not already instantiated the map.
        if (mMap == null)
        {
            // Try to obtain the map from the SupportMapFragment.
            mMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.map)).getMap();

            // Check if we were successful in obtaining the map.

            if (mMap != null)
            {
                mMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener()
                {
                    @Override
                    public boolean onMarkerClick(com.google.android.gms.maps.model.Marker
                    marker)
                    {
                        marker.showInfoWindow();
                        return true;
                    }
                });
            }
            else
                Toast.makeText(getApplicationContext(), "Unable to create Maps",
                Toast.LENGTH_SHORT).show();
        }
    }
}
```

Now the map is working correctly and we can add markers for some cares, showing its name and some additional information. The first step to do this is to prepare a marker class, containing all information about our custom specific marker:

```
public class MyMarker
{
    private String mLabel;
    private String mIcon;
    private Double mLatitude;
    private Double mLongitude;

    public MyMarker(String label, String icon, Double latitude, Double longitude)
    {
        this.mLabel = label;
        this.mLatitude = latitude;
        this.mLongitude = longitude;
        this.mIcon = icon;
    }

    public String getmLabel()
    {
        return mLabel;
    }

    public void setmLabel(String mLabel)
    {
        this.mLabel = mLabel;
    }

    public String getmIcon()
    {
        return mIcon;
    }

    public void setmIcon(String icon)
    {
        this.mIcon = icon;
    }

    public Double getmLatitude()
    {
        return mLatitude;
    }

    public void setmLatitude(Double mLatitude)
    {
        this.mLatitude = mLatitude;
    }

    public Double getmLongitude()
    {
        return mLongitude;
    }

    public void setmLongitude(Double mLongitude)
    {
        this.mLongitude = mLongitude;
    }
}
```

Now we are almost finished and we can create our `MyMarker` objects, depending on the needed information.

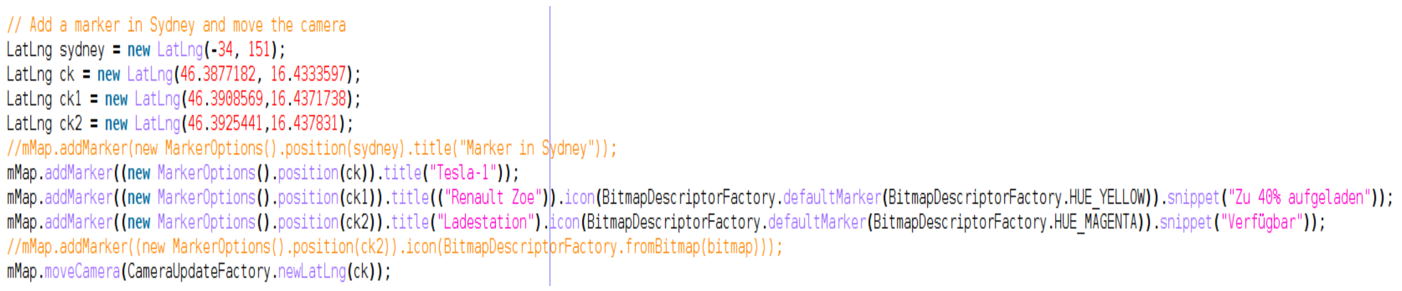
This is done by adding our markers to the the `ArrayList` of `MyMarker` in our `onCreate` method

```
// Initialize the HashMap for Markers and MyMarker object
    mMarkersHashMap = new HashMap<Marker, MyMarker>();

    mMyMarkersArray.add(new MyMarker("Brasil", "icon1", Double.parseDouble("-28.5971788"),
                        Double.parseDouble("-52.7309824")));
    mMyMarkersArray.add(new MyMarker("United States", "icon2",
                        Double.parseDouble("33.7266622"), Double.parseDouble("-87.1469829")));
    mMyMarkersArray.add(new MyMarker("Canada", "icon3", Double.parseDouble("51.8917773"),
                        Double.parseDouble("-86.0922954")));
```

Example taken from: http://www.rogcg.com/blog/2014/04/20/android-working-with-google-maps-v2-and-custom-markers#

Adding some static positions and marker content is shown in the picture bellow (Figure 3).



```
// Add a marker in Sydney and move the camera
LatLng sydney = new LatLng(-34, 151);
LatLng ck = new LatLng(46.3877182, 16.4333597);
LatLng ck1 = new LatLng(46.3908569,16.4371738);
LatLng ck2 = new LatLng(46.3925441,16.437831);
//mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
mMap.addMarker((new MarkerOptions().position(ck)).title("Tesla-1"));
mMap.addMarker((new MarkerOptions().position(ck1)).title(("Renault Zoe")).icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_YELLOW)).snippet("Zu 40% aufgeladen"));
mMap.addMarker((new MarkerOptions().position(ck2)).title("Ladestation").icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_MAGENTA)).snippet("Verfügbar"));
//mMap.addMarker((new MarkerOptions().position(ck2)).icon(BitmapDescriptorFactory.fromBitmap(bitmap)));
mMap.moveCamera(CameraUpdateFactory.newLatLng(ck));
```

Figure 3. Setting markers to a specific position and providing additional information

When applying these markers to our map, we get the following result: (Figure 4)

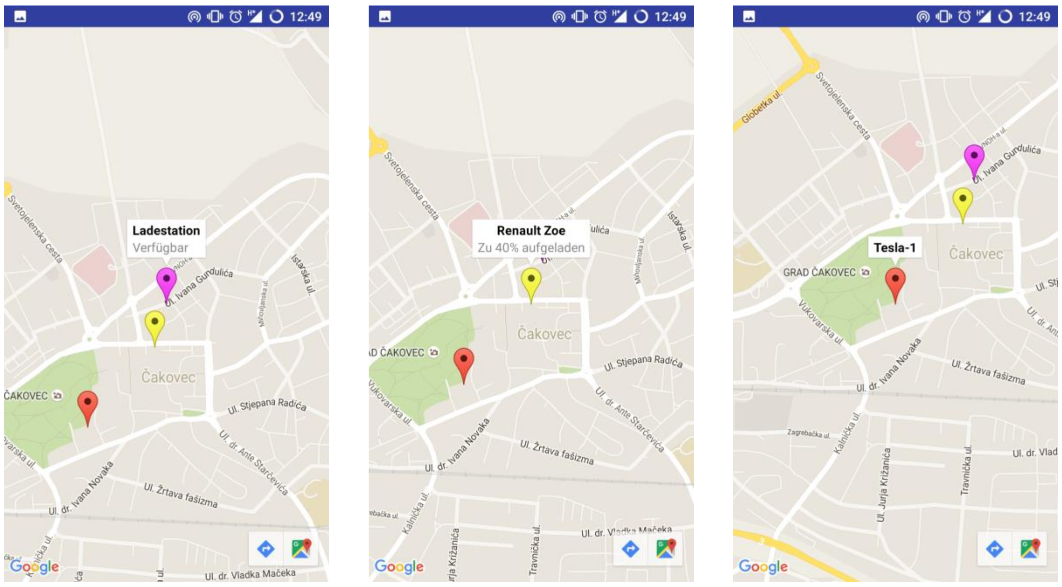


Figure 4: Showing car-information when different markers (cars) are clicked.

Since a lot of information needs to be exchanged between the service center and each single car-application, a reliable connection is needed to run the following services:

- Get information about available cars (position, state)
- Send information if the car-state has changed
- Send car position information in short intervals (to inform the service center about the current location and to run car tracking services).
- Send predefined emergency calls to the service center.
- Poll for messages form the service center (driver information services).

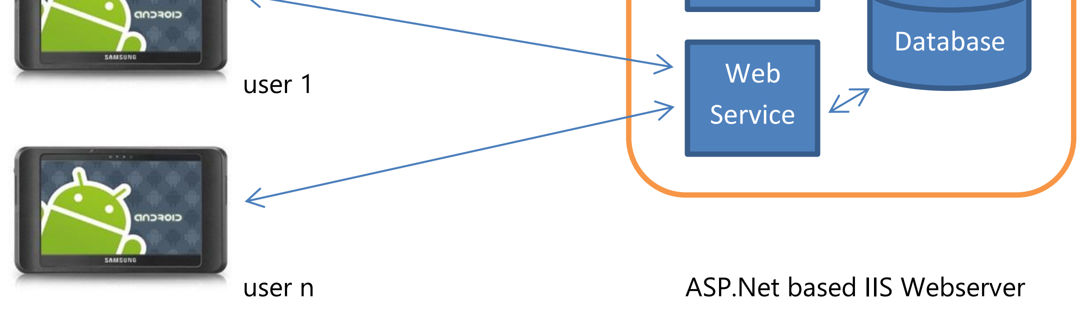
Since some experience exists in the field of web services, SOAP will be used to run all these services. Unfortunately, this interface do not exist so far, but a first idea will be given in the next section

2.3. SOAP-Webservice

A Web service is a remote function, which is offered by an electronic device to another electronic device. Web services use SOAP over HTTP protocol, so it is possible to use the Web services from any location in the world if an internet access is available.

SOAP is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It relies on the well-known Extensible Mark-up Language (XML) for its message format.

In our solution, the server side of the web-service will be based on ASP.Net and will run on an IIS web-server located at our school. When using Visual Studio, the implementation of web-services is very simpler and requires only a few mouse clicks. The same webserver will also be used to run the web application to run the car fleet management service and to run all service centre related tasks. Furthermore, this server can also run the public website as an interface between the customer and the service centre.



When starting a new web-service project, there is just a single file containing the web-service-class and all needed web service functions:

```
public class WebService : System.Web.Services.WebService {

    public WebService () {
        //InitializeComponent();
    }

    [WebMethod]
    public string WebMethod1() {
        return "WebMethod1";
    }

    [WebMethod]
    public string WebMethod2() {
        return "WebMethod2";
    }

}
```

On the client side, we only have to modify the main java file. In the following example, the **WebServiceDemoActivity.java file** file was taken from c-charpcorner.com and was adapted to meet our needs:

```
public class WebServiceDemoActivity extends Activity
{
  /** Called when the activity is first created. */
    private static String SOAP_ACTION1 = "http://tempuri.org/WebMethod1";
    private static String SOAP_ACTION2 = "http://tempuri.org/ WebMethod1";
    private static String NAMESPACE = "http://tempuri.org/";
    private static String METHOD_NAME1 = " WebMethod1";
    private static String METHOD_NAME2 = " WebMethod2";
    private static String URL = "http://www.w3schools.com/webservices/tempconvert.asmx?WSDL";

    Button btnFar,btnCel,btnClear;
    EditText txtFar,txtCel;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      btnFar = (Button)findViewById(R.id.btnFar);
      btnCel = (Button)findViewById(R.id.btnCel);
      btnClear = (Button)findViewById(R.id.btnClear);
      txtFar = (EditText)findViewById(R.id.txtFar);
      txtCel = (EditText)findViewById(R.id.txtCel);

      btnFar.setOnClickListener(new View.OnClickListener()
      {
            @Override
            public void onClick(View v)
            {
                //Initialize soap request + add parameters
            SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME1);

            //Use this to add parameters
            request.addProperty("WebMethod1",txtFar.getText().toString());
```

```
        //Declare the version of the SOAP request
        SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);

        envelope.setOutputSoapObject(request);
        envelope.dotNet = true;

        try {
            HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);

            //this is the actual part that will call the webservice
            androidHttpTransport.call(SOAP_ACTION1, envelope);

            // Get the SoapResult from the envelope body.
            SoapObject result = (SoapObject)envelope.bodyIn;

            if(result != null)
            {
                //Get the first property and change the label text
                txtCel.setText(result.getProperty(0).toString());
            }
            else
            {
                Toast.makeText(getApplicationContext(), "No Response",Toast.LENGTH_LONG).show();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        }
    });
```

The Java code above shows the initializing of a sop request and how parameters can be added to the function call. The try-section shows, how SOAP calls are sent to the web-service and how call-results can be treated.

3. Summary

As we have seen in the survey, many car-sharing IT-solutions suffer from an essential service – *the car tracking mechanism.* The reason therefor is, that some special hardware devices are needed, which either need to get access to the on-board GPS-system or need a separate GPS-device, equipped with an extern antenna.

The main idea of the proposed solution is to use the customers tablet or smartphone and if it is possible to bind the user's identity and services to the car. In doing so, it would be possible to make use of the tablet's GPS-functions to realize a car tracking mechanism for free.

But not only GPS is for free – if G3 or LTE is available on the tablet, the internet connection can be used for any kind of communication (driver-center, center-driver, emergency calls, etc. …). Finally, the user's identity can be connected to the tablet and authentication mechanisms can be implemented easily by a simple endpoint- or device authentication.

Even for the car locking system, NFC-technologies can be used for the pairing mechanism (car-app) or simple Bluetooth-connections can be used to open the care by using certificates and confidential protocols. Fortunately, protocols based on these technologies are well known and public available, thus, the presented work do not investigate into that direction. This would take too much time with to less essential output.

For that reason, the presented work was limited to the following two topics: car tracing and car communication.

All presented work started from the assumption that the tablet was authenticated by the car by using any appropriate mechanism.

In section 2, the current car-location and detection mechanism was presented, based on a simple prototype implementation. Since the app is developed by using Android Studio, all necessary settings are shown with the aid of short code snippets.

Furthermore, the placing of markers –*to show the current location and state of a car*– is shown as well.

The communication between the app and the service center was intended to be based on a SOAP-web service. This part was developed in the second year of the project.

During the second year, we built a web-platform, to be able to perform all administrative tasks, which were needed by the service center. This web-application is based on ASP.Net and is written in C# for reasons of simplification, since C# is one of the programming languages, which are used in our programming lessons.