Developing a framework-concept to evaluate and present car-telemetries to support users in fuel- and time-saving driving

Reichart Robert

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2014

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, June 25, 2014

Reichart Robert

Contents

Abstract

In this thesis a framework and a client are implemented which support users in fuel- and time-saving driving. To do so users have to create an account using a valid email address to which a activation code gets sent. When the account is activated users are able to add cars to their cars list by using the cars manufacturer, model, engine, and build date range. While driving, the cars telemetry which contains the current fuel consumption, speed, and location are sent to the server using a self-implemented JSON based protocol, compared against other users car telemetries and responded to the client which shows the current users location, speed and the speed the user would may drive to reach the destination as fast as possible while not having to handle the circumstance of too much fuel consumption. The implemented framework should be modular, high scaleable and easy to be customized and configured.

Kurzfassung

In dieser Masterarbeit werden ein Framework und eine Benutzeranwendung entwickelt, um Anwender beim sprit- und zeitsparenden Fahren zu unterstützen. Um dies zu erreichen müssen Anwender mit einer gültigen E-Mail Adresse (an welche später ein Aktivierungscode geschickt wird) einen Account erstellen. Sobald der Account aktiviert worden ist, können Anwender Autos zu ihrer Autoliste hinzufügen in dem sie den Hersteller, das Model, die Motorleistung sowie die Datumsspanne der Produktion angeben. Während der Fahrt werden die Telemetriedaten des gefahrenen Autos (welche den momentanten Spritverbrauch, Geschwindigkeit und die Position des Anwenders beinhalten) mittels eigenentwickelten und JSON basierten Protokoll an den Server gesendet, welcher diese mit den Telemetriedaten der anderen Anwender vergleicht und zur Benutzeranwendung (welche dem Anwender seine aktuelle Position und seine Geschwindigkeit sowie die maximale Geschwindigkeit die er fahren kann ohne gravierierende Spriteinbußen hinnehmen zu müssen) zurückgibt. Das entwickelte Framework soll modular und hoch skalierbar sein sowie einfach angepasst und konfiguriert werden können.

Chapter 1

Motivation

Since the last decade the fuel price in Germany and Austria is more and more increasing. While one liter of super fuel in Germany has cost in average 1.05 € in 2002, the average price in 2012 was 1.60 € (see Figure 1.1).

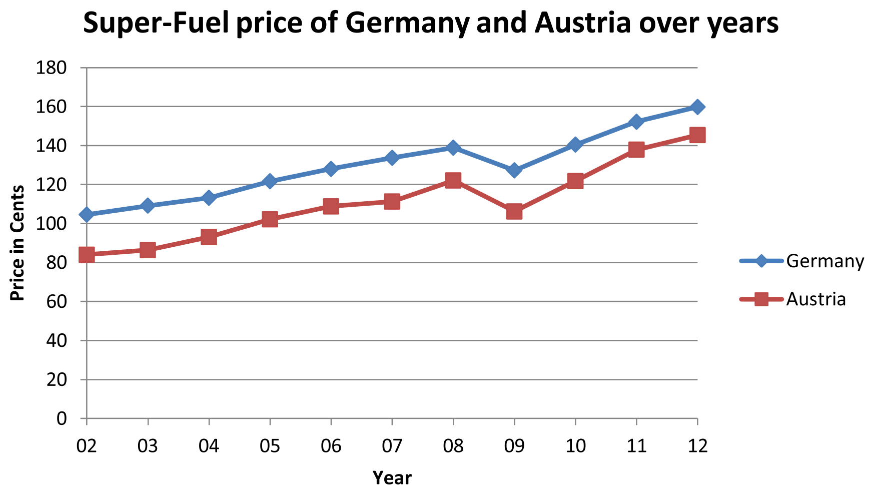


Figure 1.1: Development of the super-fuel price per liter in Germany and Austria. Data taken from [31] and [32].

A way to prevent car drivers from spending too much money on fuel is to change their driving habit. This can be achieved by giving car drivers direct feedback to their actual driving behavior. To do so the cars dashboard can be used, which is already done by modern cars. Another way, which is also useful for older cars, is using smartphones which are widely distributed nowadays. A first cheap solution for needing less fuel while driving is to drive as slow as possible and allowed which is described by Figure 1.2.

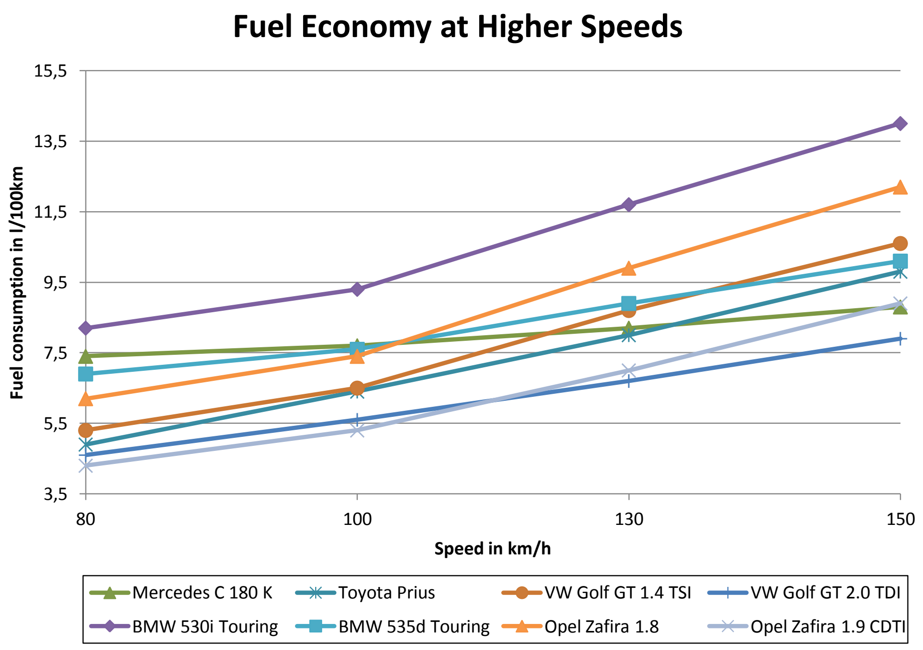


Figure 1.2: Fuel economy at higher speeds. Data taken from [14].

A disadvantage for using this method is that a car driver will not reach the destination as fast as possible. In this master thesis a solution should be discussed which takes care about needing as little fuel as possible while reaching a destination as fast as possible.

Chapter 2

Related Work

In order that this thesis does not only look into one specific field, two types of related work have to be differed. On the one hand, related work which is handling fuel saving driving, and on the other hand related work which is handling time saving driving.

2.1 Fuel-saving driving

There are two concepts of applications described in this thesis which should support and motivate users in fuel saving driving. Both concepts receive their car data from the OBD2 [16] diagnose interface which has to be installed in every car since 2001 [39].

2.1.1 AndroWi

AndroWi [2] is using a very simple collaborative concept. As Figure 2.1 illustrates, a user is building a Vehicular Adhoc Network. By using this network a user is connected to other cars in a given distance. In AndroWi's case the WiFi-Direct protocol is used. If the user receives a 'better' (based on 'fuzzy logic' which is mentioned but not more detailed described in [2]) driving data from the connected cars in his area, the driving data gets shown to the user.

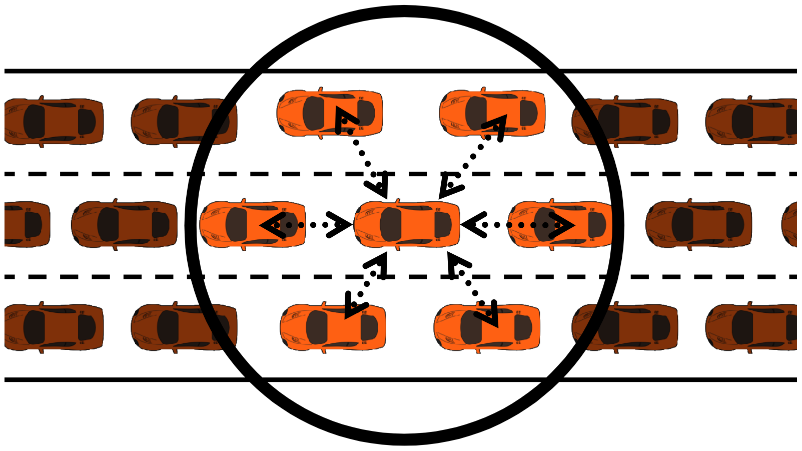


Figure 2.1: Concept of AndroWi by using a VANET.

2.1.2 GreenR

GreenR [23] which is developed by Josef and Stefan Wasserbauer is an iOS application to motivate users in fuel saving driving. The concept which is described by Stefan Wasserbauers master thesis from 2012 [6] is that all users are collaborating to paint the streets green on the map shown by the application (see Figure 2.2). To do so the current driving data of the car gets evaluated. If the user is driving economically which means that the car driver is not accelerating too quickly, the driven path will be painted green otherwise it is painted blue.

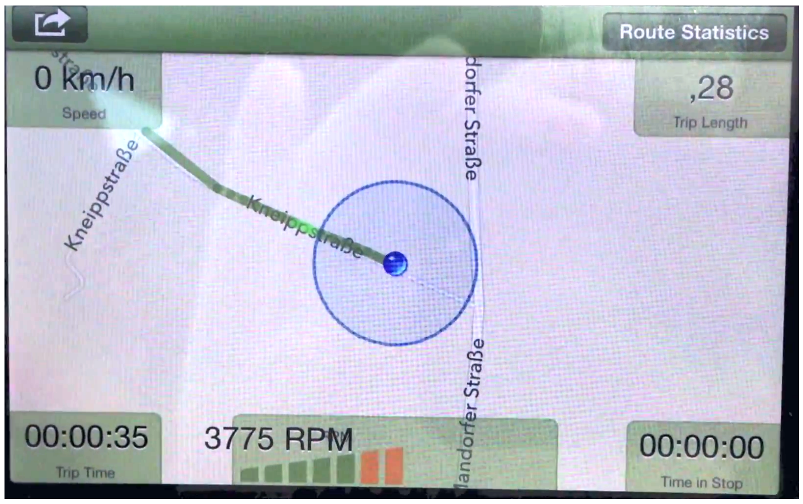
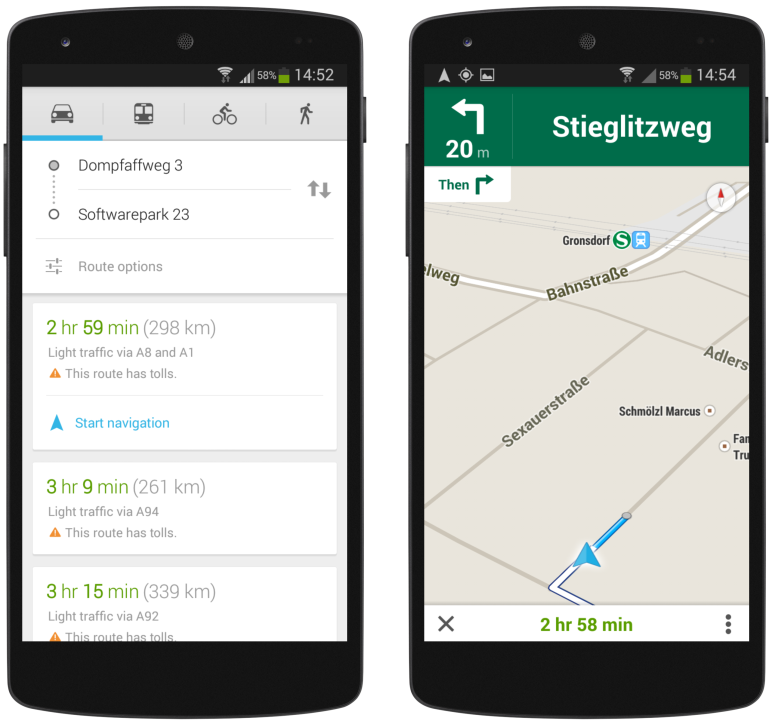


Figure 2.2: Concept of GreenR. Image is taken from [24].

2.2 Time-saving driving

2.2.1 Google Maps for Android/iOS

Google Maps which is available for Android [35] and iOS [20] is a server-side based navigation application which guides users on the fastest and therefore most time-saving way to their destination. Its download and usage is completely free. To start navigating a user has to enter the starting location (which can automatically be replaced with the current location) and end location. This data gets sent to the server which responds at least one suitable route (in most cases three) and the corresponding precalculated trip time, length and warnings if a route for example has a toll (see Figure 2.3a). In navigation mode the screen is simple and provides all information in a non disturbing way (see Figure 2.3b). The instructions are also provided via audio. Because it's a server-side application, the map and the driving instructions have to be downloaded which can lead to a lot of traffic for mobile devices like smartphones.

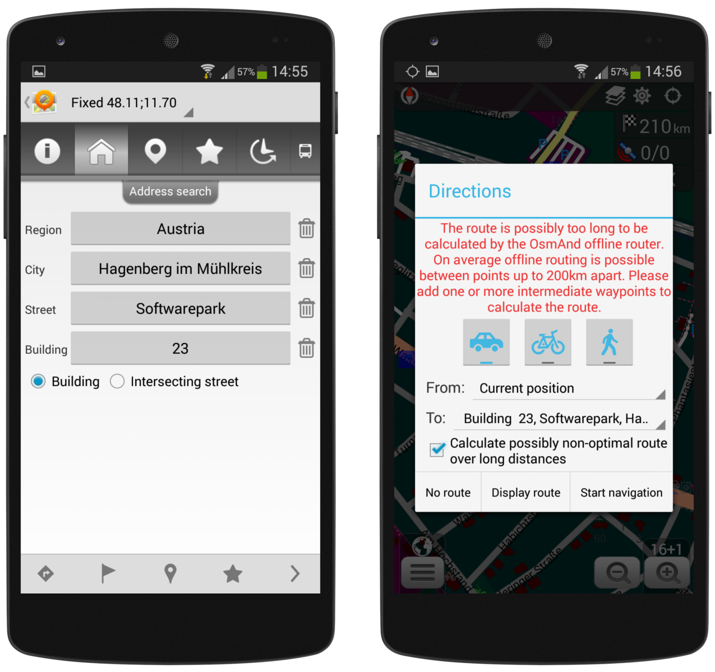


(a) Possible Routes (b) Navigation Mode

Figure 2.3: Google Maps: Routing and Navigation.

2.2.2 Osmand+

Osmand+ [43] which is only available for Android is another navigation application which guides users on the fastest way to their destination too. Contrary to Google's Maps application it has to download and store maps on the device which makes it to a client-side application. The maps which can be downloaded by Osmand+ are provided by OpenStreetMap [41]. In the free version a user can only download maps and additional packages for ten times. The full version which can be downloaded for €6.99 provides unlimited content support. To start navigating a destination has to be set. To set a destination its country has to be selected at first. After the destinations country has been selected, the desired town, street and house-number are selectable too (see Figure 2.4a). If the distance between the current location and the destination location is below 200 km the route will automatically be calculated, otherwise an alert is shown that no valid route can be found (see Figure 2.4b) which can either be accepted or an option selected which makes it possible to find a route which might not be optimal in meanings of the trip time. While navigating the screen of Osmand+ is not offensive or disturbing which is illustrated by Figure 2.5. All directions can clearly be seen and are also provided via audio if a suitable package is downloaded.



Region: Austria
City: Hagenberg im Mühlkreis
Street: Softwarepark
Building: 23
[✓ Building] Intersecting street

Directions

The route is possibly too long to be calculated by the OsmAnd offline router. On average offline routing is possible between points up to 200km apart. Please add one or more intermediate waypoints to calculate the route.

From: Current position
To: Building 23, Softwarepark, Ha..
☑ Calculate possibly non-optimal route over long distances

No route | Display route | Start navigation

(a) Adress Input **(b)** Trip length alert

Figure 2.4: Osmand+: Routing.

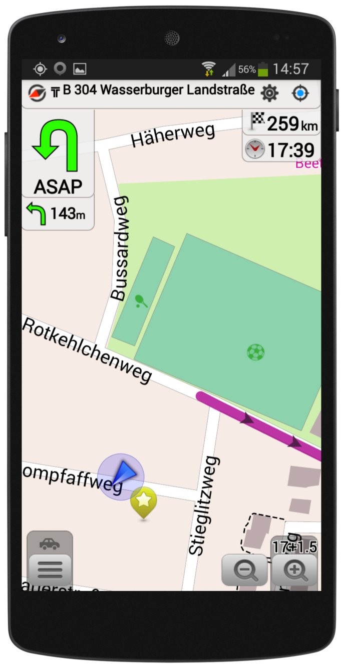


Figure 2.5: Osmand+: Nagivation Mode.

Chapter 3

The Concept

In this chapter of the thesis a concept should be presented which supports multiple users in fuel- and time-saving driving. This can be achieved by using a client-server structure. To design a concept which is suitable to such a client-server structure, the requirements shown in section 3.1 have to be collected and evaluated which leads later on to the results shown in section 3.2.

3.1 Requirements

Because of the above mentioned client-server structure the requirements for the server (shown in section 3.1.1) and the client (shown in section 3.1.2) have to be collected and evaluated. The results which are discussed in section 3.2 are showing the final concept.

3.1.1 Server

The servers requirements are that it has to be modular, high scaleable, persistent and secure because of the data sent to and received from the clients. In addition to that it must be able to receive and respond car telemetries, distinguish different cars and users and must be able to calculate routes from a given location to a given destination. All requirements mentioned before are more detailed described below.

Modularity

To make it easy for developers to extend a framework it has to be easily to be extended and configured. This can be achieved by modularity. The idea behind modularity in this case is that modules can easily be added or removed by developers. All modules are later loaded by the framework in its start-up phase.

High scaleability

In the worst case the concept has to deal with all car drivers in the area where the concept should be available. For example, in Germany and Austria a total count of 67.88 million cars were registered at the end of 2013 (see [7] and [50]). In the case that only 5% of Germany's and Austria's car drivers will use the concept it has to handle the connections and network-traffic of approximately 3.4 million car drivers. To handle such a big amount of connections the framework has to create a bunch of threads/processes which can lead to a slow or crashing server. To prevent such a circumstance a thread- or process-pool should be used.

Providing/Receiving car-telemetry

Car drivers can be supported in time- and in fuel-saving driving if they are supported with information where to drive (more detailed described in section "Navigating" on page 11), how fast to drive and which gear to use.

Due to the fact that there will be too much time effort and too much data to store to provide car telemetries of every location to guide a user to his destination as fast as possible while driving as economically as possible, the mathematics behind the concept described in this paper calculates a score for each dataset received from the user. The best data, which means the data with the lowest score, gets provided.

Mathematics

A users current location defined by his latitude and longitude can be mapped to a position on the worlds map. If this position meets certain requirements, in this case if the location is an intersection (which means that it is an element of the group of all intersection locations $\mathbb{L}$), the user reached the end of a section s, which is described by

$$s = [l_{\text{start}}, l_{\text{end}}]\text{ , where } l_{\text{start}},\, l_{\text{end}} \in \mathbb{L}\ . \tag{3.1}$$

A users dataset d_i which is described by the time Δt_i the user has needed to get from the sections start to the sections end, the needed average fuel consumption $\overline{\Delta f_i}$, the driven average speed $\overline{v_i}$ and the average gear $\overline{g_i}$, such that

$$d_i = \{\Delta t_i, \overline{\Delta f_i}, \overline{v_i}, \overline{g_i}\} \tag{3.2}$$

can be gained by collecting the users car-telemetries in short and regular intervals.

For each dataset d_i a score S_i can be calculated by the sum of the percentage difference of the users used time and average fuel to the average time $\overline{\Delta t_0} = \frac{1}{n}\sum_{i=1}^{n} t_i$ and average average fuel $\overline{f_0} = \frac{1}{n}\sum_{i=1}^{n} \overline{\Delta f_i}$ of all users such that

$$S_i = \frac{\Delta t_i}{\overline{\Delta t_0}} + \frac{\overline{\Delta f_i}}{\overline{f_0}} = \frac{\Delta t_i}{\frac{1}{n}\sum\limits_{i=1}^{n} \Delta t_i} + \frac{\overline{\Delta f_i}}{\frac{1}{n}\sum\limits_{i=1}^{n} \overline{\Delta f_i}} \ . \tag{3.3}$$

Over all scores $S_i \in \mathbb{S}$ of section s there is now at least one minimal score S_{min} which underlying data d_r defined by

$$d_r = \{S_{\text{min}} \in S | S_{\text{min}} \leq S_i\} \tag{3.4}$$

gets returned to the specific user.

A big drawback of the above mathematics is that it does not take care about different car fuel consumption averages. For example a Smart fortwo has a much lower fuel consumption average $\overline{\Delta f}$ than a Chrysler 300 C Touring which is shown by Table 3.1.

Manufacturer & Model	$\overline{\Delta f}$
Chrysler 300C Touring	10.8
Porsche Panamera	8.4
Volkswagen Polo III Variant	6.2
Kia Cee'd	6.0
Mini One	5.4
Smart fortwo	4.2

Table 3.1: Average fuel consumption of different cars: Data taken from [9].

To take care about different car types using the mathematics above, the system has to distinguish which datasets to take into the score calculation. One solution to achieve this is to combine each dataset with its corresponding car type (more detailed described below).

User Management

As mentioned above, different fuel consumption averages of cars have to be distinguished. A solution to such a problem can be user management. This means that every user has to register to get his own account. Now taking this account, users have the possibility to add cars (provided by the server) to their list of driven cars and select a car for driving. Whenever the users data is now sent to the server, the corresponding car gets added automatically which makes it possible for the server to distinguish the users data by different car fuel averages.

Navigating

The route with the shortest trip time has to be calculated to make it easier for car drivers to take an optimal route to their destination and therefore driving time-saving. This is already done by navigation systems mostly using the Dijkstra [3]- or A* [5]-algorithm, but also needs to be done in this concept to get a one device system to not depend on other applications and to not disturb users while driving.

Persistency

A lot of data (see section 3.1.1) has to be managed and handled. By making it persistent with the drawback to a delay due to reading it from disc to memory can prevent it from being lost due to an server shut-down, restart or crash.

Security

Due to the user management, sensitive data (like a persons email address, password, current location or the driven cars) has to be sent to the server. To make it hard for third parties to get this data it has to be secured. This can be achieved by securing the connection to the clients using SSL. Another important point of security is not to store passwords and other personal informations in human readable form.

3.1.2 Client

A part of the above mentioned requirements to the server of the client-server structure are also suitable for the client. These include that the client has to provide a way to show and receive car-telemetry as well as a user management and security. An additional point is that the client must be able to show the users current location on a map.

Providing maps and car-telemetries

Because of users should be navigated to their destination to drive as time-saving as possible, a map should be shown which shows the users current location and in which direction he should drive. Additional information which has to be provided to users while driving are car-telemtries (which are more detailed described in the servers requirements in section 3.1.1) to make it possible for users to drive as fuel-saving as possible. To receive car-telemetries, the users current car-telemetry has to be gained and sent to the server.

User management

The server must be able to distinguish different car types, therefore the client has to provide information about which kind of car the user is currently driving. To do so the user must be able to add his driven cars to a list and select the car which he is currently driving. This can be achieved by using user management.

Security

Because of a lot of sensible user data (like the users alias and password) is sent to the server the connection has to be secured to not make it easy for third parties to gain this information. As mentioned in the server-requirements (see section 3.1.1) the connection should therefore be SSL-secured. Another point in security is that the users alias and password should not be stored in human readable form whenever they are stored on the users device.

3.2 Result

Taking all of the above mentioned requirements into account leads to the concept described below. As mentioned in section 3.1 the preferred structure is a client-server-structure which is combined with a database and a SSL secured connection to match the security and persistency requirements (see Figure 3.1).

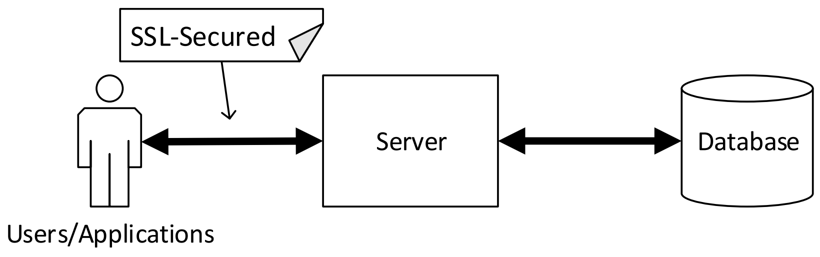


Figure 3.1: The concepts Client-Server structure.

3.2.1 Server

As Figure 3.2 shows the server of the concept is divided into different managers. The network manager whose task is connection handling, the event manager which makes it possible for modules to fire and handle events and the module manager which holds and loads the user module, tracking module, database module and other modules which may be provided.

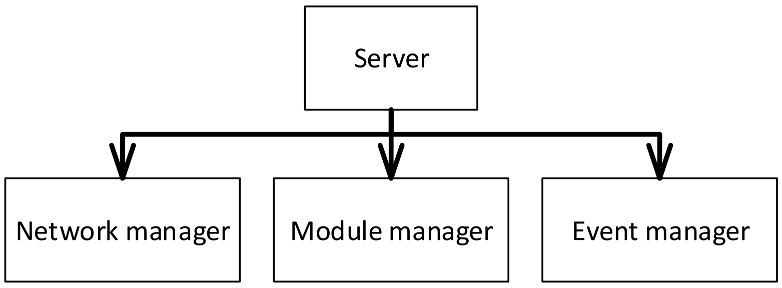


Figure 3.2: Server overview

Network manager

The network managers first task on server start-up is to open a port on the server on which the framework can listen on to handle SSL connection establishment. As a second task the network manager has to handle all the clients' incoming connections and delivered messages (see Figure 3.3). To achieve this a thread-pool which is mentioned in section "High scaleability" of page 9 is used (because a lot of threads can be created on server start-up where they can not cause any harmful delay) where every connection to a client is handled in its own thread.

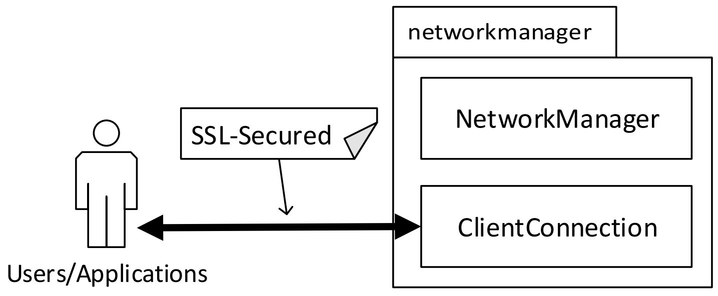


Figure 3.3: Network manager

Module manager

As seen in Figure 3.4 the module managers simple task is to load and hold the user module, the tracking module, the database module and all other modules which may be provided by the developer. New modules, which are not currently part of the framework can easily be added or exchanged by following a few guidelines which are described in detail in chapter 4.

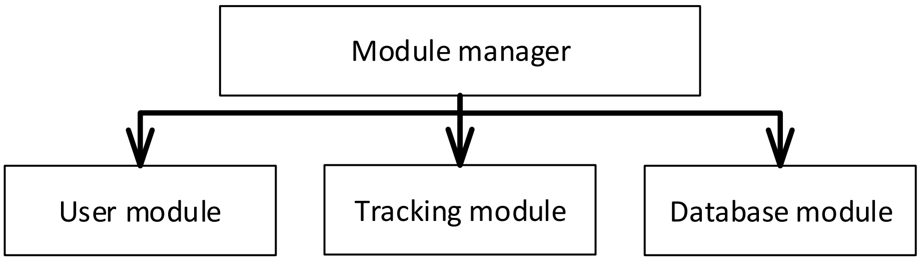


Figure 3.4: Module manager

User module

The requirements described in section "User management" of section 3.1.1 are fulfilled by the user module. Its tasks are creating, deleting and updating user accounts. Objectives which are part of updating user accounts are changing passwords and adding, deleting or selecting cars which are currently driven.

Tracking module

Receiving car data, calculating the car data's score (which includes distinguishing between different fuel average consumptions), providing the car-telemetry with the lowest score and calculating routes are fulfilled by the tracking module by using the database module.

Database module

The database module has to be implemented in a special way because it represents the database which is described in section 3.2.2. The database module has to provide methods to create, update or delete objects in the database. By using the database module, for the server it is not necessary to know to which specific type of database it is connected to. This could either be a MySQL-database or another type of database.

3.2.2 Database

As described above the type of the database is free to choose but in this concept and its subsequent implementation a MySQL-database is used which contains all information needed to enable user management and to be able to compare car telemetries without any redundancy (see Figure 3.5).

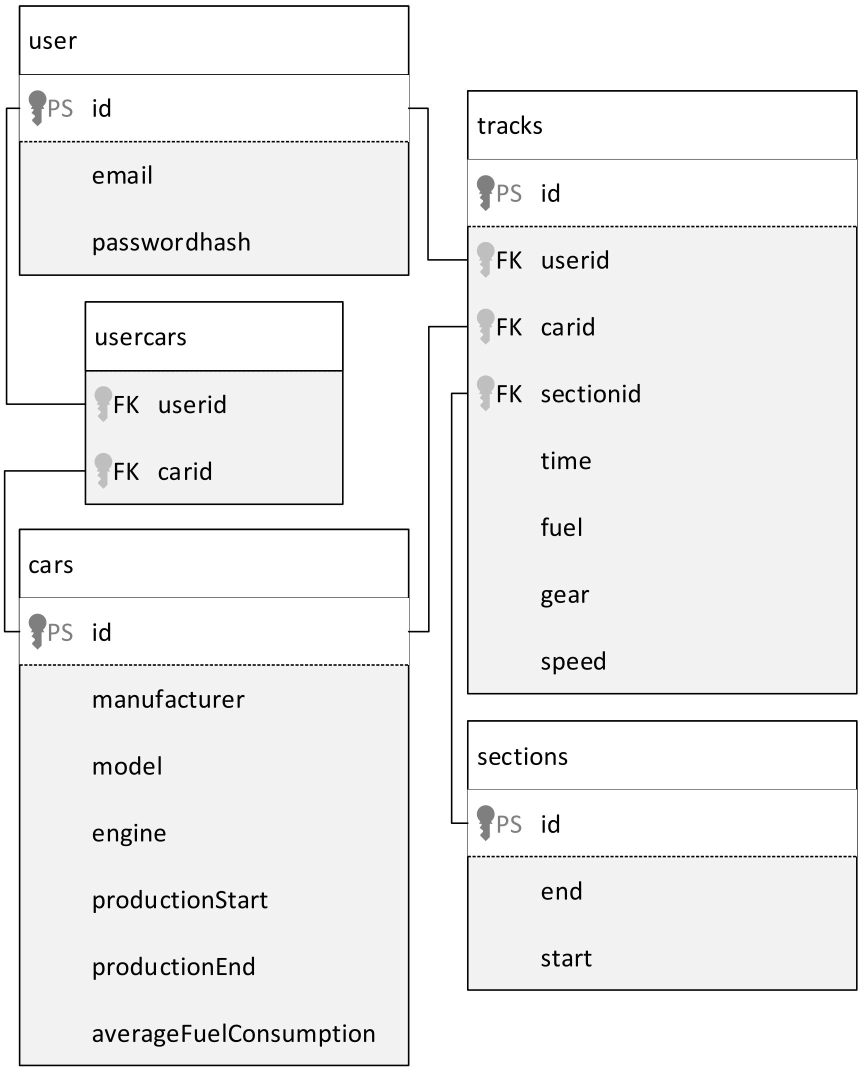


Figure 3.5: Database model

User and Cars

The least amount of data which has to be stored to enable user management is the users e-mail address, password and driven cars. To uniquely identify a car the cars manufacturer, model, engine (horsepower and fuel consumption average) and the cars production date range have to be stored.

Sections and Tracks

While only a sections start- and end-point is needed to store a in Equation 3.1 defined section, much more informations are needed to store a users track. As described in Equation 3.2 and Equation 3.3 the data which has to be saved to store a track are the tracks section, the users id, the users needed time, the used car, the fuel consumption average, the used speed average and the used gear average.

3.2.3 Client

As seen in Figure 3.6 the client consists of several managers. The network manager which handles the connection to the server, the view manager which provides different views, the user manager which is used for user management and the telemetry manager which includes the locationmanager and carmanager to gain the needed car telemetries.

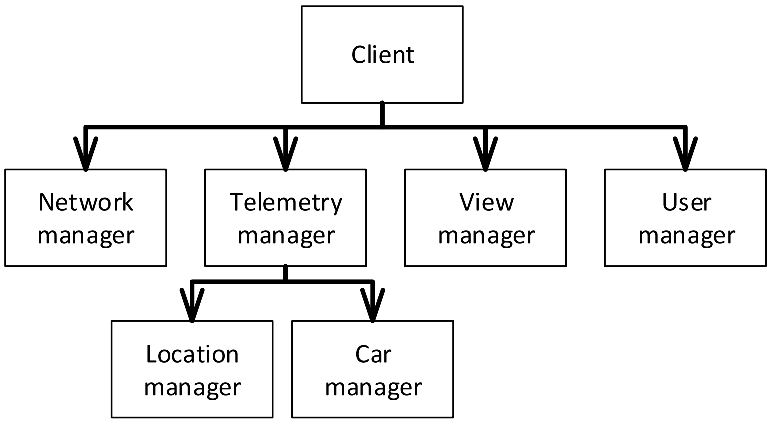


Figure 3.6: Client structure

Network manager

The clients network managers task is to establish and handle the connection to the server. Because of sensitive user data is sent to the server, SSL is used to secure the connection against third parties.

User manager

The task of the user manager is to create and manage the users account. This means that the manager has to be able to add cars to the users cars list and delete them if necessary. Another task of the user manager, is to make it easy for users to select which car they are driving.

Telemetry manager

Collecting car specific data which is relevant to be sent to the server is the telemetry managers task. Therefore the car managers task is to collect the users current speed, gear and fuel consumption average. To gain the cars current speed, gear and fuel consumption average it has to be connected to the cars diagnose interface described below. The current users location might not be direct a car telemetry data but has to be sent to the server too to make it possible for the server to achieve on which section the user actually drives. To get the users current location, the locationmanager is used.

OBD2

As described in an article of the Air Resources Board from the California Environmental Protection Agency [16], OBD2 is an acronym for On-Board Diagnostic II which is the second generation of an on-board self-diagnostic system which has to be installed in every car which is sold in California since 1996. In Germany the OBD2 system has to be installed in every super fuel car which is sold since 2001 and every diesel fuel car which is sold since 2004 (see [39]). The shape and bit configuration of the OBD2 interface which is shown by Figure 3.7 and Table 3.2 is regulated in Germany by ISO 15031-3. Which protocol to use, is free to the cars manufacturer choice. The J1850 protocol is mostly used by American cars, while K-Line is used in European cars. Since 2007/2008 the CAN protocol has to be supported in America and Europe.

To request specific values by using the CAN protocol, hex-coded codes have to be sent to the interface. The first byte specifies the mode to be used (see Table 3.3) while the second byte specifies the parameter ID (see Table 3.4) if it has to be used for the chosen mode. For example to request the current engine coolant temperature in °C the hex-coded request **0105** [39] has to be sent to the OBD2 interface. More but not all modes and parameter IDs which can be sent to the interface are shown by Table 3.3 and Table 3.4. The hex-coded response can be calculated to the real temperature by using

$$V = \frac{H}{256} \cdot [M - m] + m, \tag{3.5}$$

where V, H, M and m are the calculated value, the returned hex-coded value and the maximum and minimum value of the requested parameter. As described in [39] at first the returned hex-coded value has to be converted to a decimal-coded value. Later on it has to be divided by $1 \cdot 2^8$ bit as it is a 1 byte value. Because of this byte is a normalised value for a given range and offset the byte has to be multiplied by the the range and the offset has to be added. For example the hex-coded return value **66** using Equation 3.5 results in

$$\frac{66_{\text{hex}}}{256} \cdot [215 - (-40)] + (-40) = \frac{102}{256} \cdot [215 - (-40)] + (-40) = 61.60 \quad (3.6)$$

(see [39]) which means that the returned engine coolant temperature is round about 62°C.

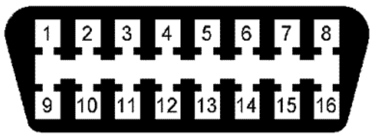


Figure 3.7: The shape of the OBD2 Interface. Image taken from [10].

Pin	Description
2	J1850+ Protocol
4	Ground
5	Ground
6	CAN-H Protocol
7	K-Line Protocol
10	J1850- Protocol
14	CAN-L Protocol
15	L-Line Protocol
16	Battery Voltage

Table 3.2: OBD2 interface pin configuration by ISO 15031-3.

Mode	Description
01	Live Data
02	Freeze Frames
03	Stored Trouble Codes
04	Clear/Reset Stored Emissions Related Data
05	Oxygen Sensors Test Results
06	On-Board System Tests Results
07	Pending Trouble Codes
08	Control of On-Board Systems
09	Vehicle Information
0A	Permanent Trouble Codes

Table 3.3: Possible OBD2 Modes: Data taken from [46].

PID	DBR	Description	Min	Max
05	1	Engine coolant temperature	−40 °C	+215 °C
06	1	Short term fuel % trim Bank 1	−100 %	+99,22 %
07	1	Long term fuel % trim Bank 1	−100 %	+99,22 %
08	1	Short term fuel % trim Bank 2	−100 %	+99,22 %
09	1	Long term fuel % trim Bank 2	−100 %	+99,22 %
0A	1	Fuel pressure	0 kPa	765 kPa
0B	1	Intake manifold absolute pressure	0 kPa	255 kPa
0C	1	Engine RPM	0 1/min	16383,75 1/min
0D	1	Vehicle speed	0 km/h	255 km/h
0E	1	Timing advance	−64 °	63,5 °

Table 3.4: Possible OBD2 Codes defined by SAE J1979 where DBR are the Data bytes returned: Data taken from [46], [38] and [40].

View manager

Because of the client has to provide a various of information (like the users cars list, location, where to drive, current speed, current gear, which speed to drive and which gear to use) to the user and the client will be used on a smartphone installed in a car, the clients design has to be simple, fit on a little screen and should not be disturbing. To achieve this a lot of different views will be used and therefore a view manager is needed.

Navigating
As seen in Figure 2.3b there is already a non-disturbing design to navigate users to their destination by just providing the information on top of the map. By customizing this user interface the information how fast the user is driving, how fast the user should drive, which gear the user is already using and which gear the user should use can simply be displayed. In this concepts case it is simply shown by little arrows beside the users speed and gear if the user should drive faster or slower or if the user should shift up or down (see Figure 3.8a). A simple button leads to a menu where the user can fill in his destination (see Figure 3.8b). If the destination can not be found by the server it responds at least one possible solution. If a destination is chosen, the destination gets sent to the server which responds a possible route which gets displayed on the applications map (see Figure 3.8a).

Showing Maps
There are two ways of showing maps on the users devices. Maps can be stored on the Server or on the clients device itself:

Offline
To deploy maps on user devices the raw data of the maps which can be rendered on the users device has to be stored on the server. To make it easy for users to access and download this data a user interface has to be created. An application which uses this type of providing map data is Osmand+ (see section 2.2.2). A drawback of using this type of showing maps is that it needs a lot of space to store the maps on the users device. For example the maps of Bavaria and Austria used by Osmand+ are consuming a space of approximately 1001 MB.

Online
Another way of showing maps which does not take as much as space as the method described above but needs a proper internet connection is by downloading and caching only the needed map for showing the users current location like it is done by Google Maps (see section 2.2.1).

User manager
To provide a simple user management the user has to register an account which can simply be done by the users email address and password (see Figure 3.9). Later on users have to log-in with their credentials if it is not done automatically when the users credentials are not stored on the device.

Car manager

If the user is logged in, the cars list and an easy way to add new cars (like by choosing the cars manufacturer, model, engine and build year), show a cars details, select the currently driven car and delete existing cars have to be provided (see Figure 3.10).

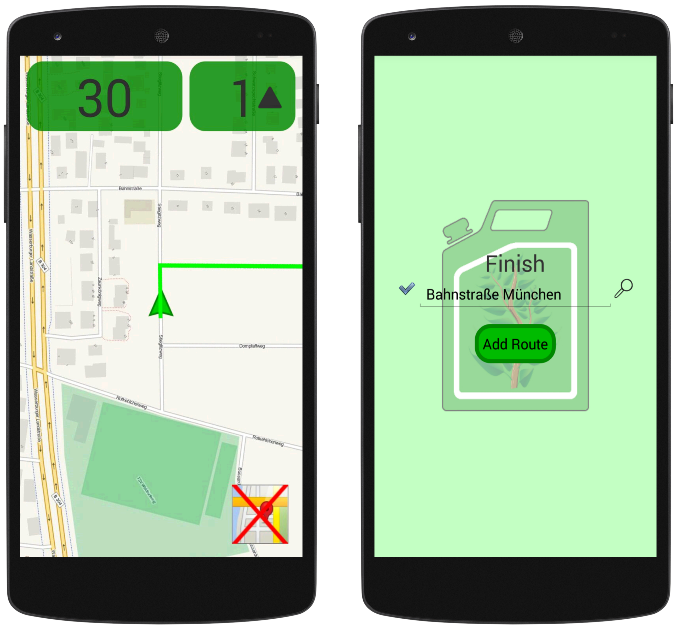
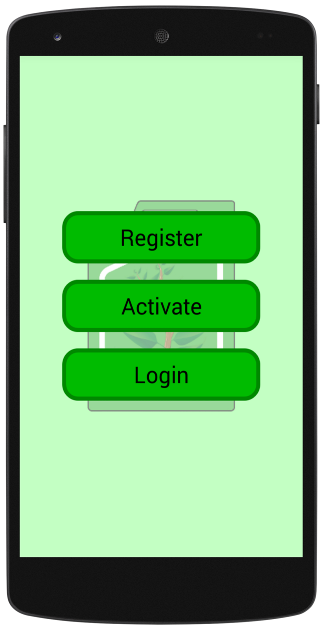
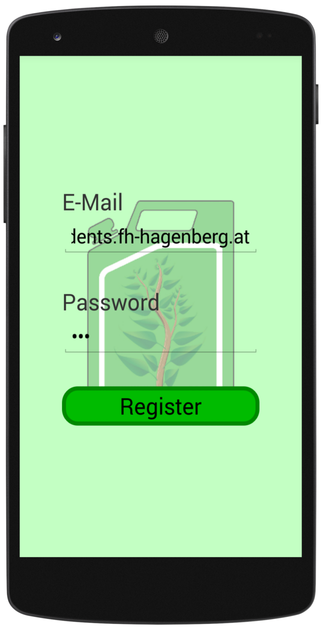


Figure 3.8: Navigating

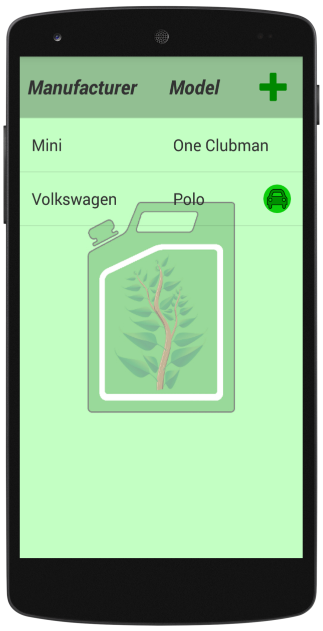


(a)

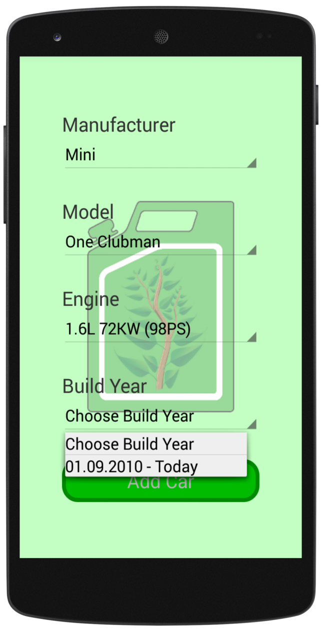


(b)

Figure 3.9: User manager



(a)



(b)

Figure 3.10: Car manager

Chapter 4

Implementation

In this chapter of the thesis, the server (discussed in section 4.2) and a suitable client (discussed in section 4.3) shall be implemented. Because of the server should be able to to run on most machines the programming language Java is used. Connections to clients are handled by using ServerSockets and a self-implemented and JSON based protocol which is more detailed described in section 4.4. A graphical user interface is implemented too by using the Standard Widget Toolkit to make it easier for programmers and users to start and configurate the server.

To reach as many smartphone users as possible, a client for the smartphone operating system with the most market share in the third quarter of 2013 is implemented (see section 4.1).

4.1 Smartphone operating systems

As seen in Table 4.1 there are a lot of available smartphone operating systems. To reach as many smartphone users as possible the smartphone operating system with the most market share of the third quarter of 2013 was chosen.

4.1.1 Android

As described in Stephan Brähler's paper "Analysis of the Android Architecture" [51] and in an article from an Android Blog [11], Android is a Linux based smartphone operating system whose five level software stack (see Figure 4.1) contains parts implemented in C++ (green parts) as well as parts implemented in Java (blue parts).

Operating System	3Q13 Units	3Q13 Market Share %
Android	205,022,700	81.9
iOS	30,330,000	12.1
Microsoft	8,912,300	3.6
Blackberry	4,400,700	1.8
Bada	633,300	0.3
Symbian	457,500	0.2
Others	475,200	0.2
Total	**250,231,700**	**100.0**

Table 4.1: Worldwide Smartphone Sales to End Users by Operating System in 3Q13. Source: Gartner (November 2013) [18].

Developing

Eclipse [15] is preferred to be used as development environment because debugging and deploying is much more easier than by using the command line which is besides the Netbeans development environment the only alternative to it. To make a basic Eclipse environment work together with Android phones, the Android Developer Tools Plugin [26] has to be used. A bundle which contains the Eclipse development environment and an already installed Android Developer Tool Plugin can be downloaded in the ADT Bundle section from the Android developer site [47].

Testing and Debugging

To test and debug applications on a Android device, it has to be connected to the Android Debug Bridge. This could either be a real device where USB debugging is activated and which is connected via USB, a so called Android Virtual Device (see Figure 4.2), or a virtual machine running a Android System which is installed through an image from the AndroidX86 project [13] (see Figure 4.3).

Graphical User Interfaces

Graphical User Interfaces in Android are created by using Java code or a Android Layout XML. The layouts and elements which could be used are shown by Figure 4.4. For example to create a simple HelloWorld application the Java code of program 4.2 or the XML code of program 4.1 could be used.

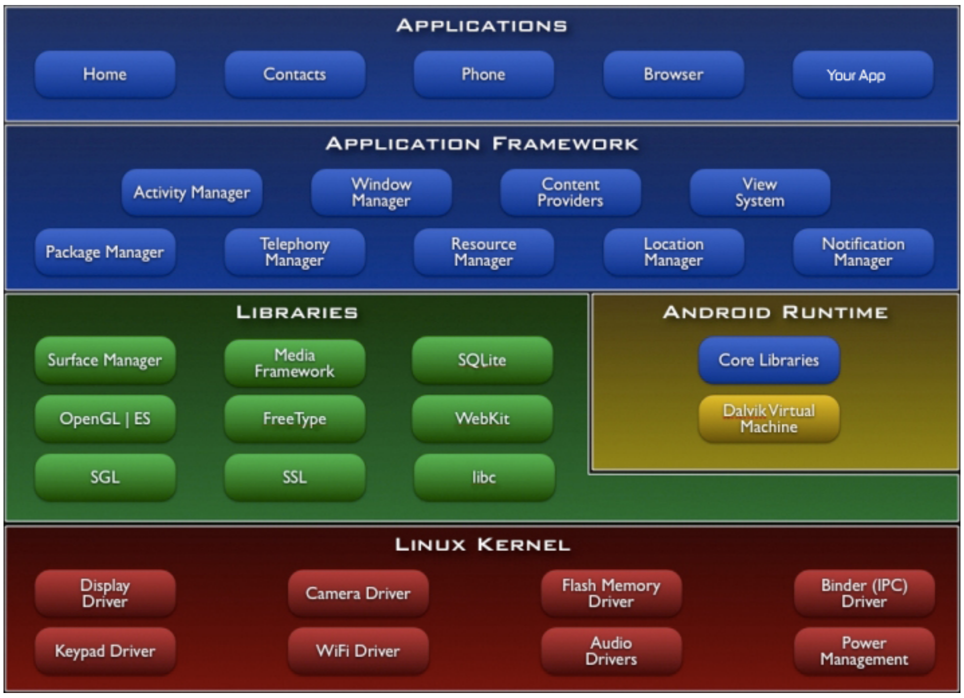
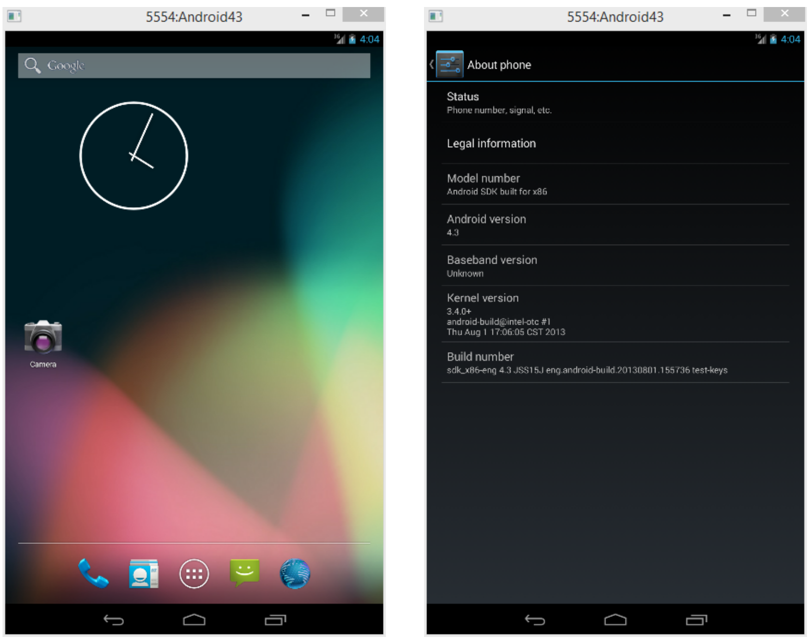


Figure 4.1: Android software stack. Image taken from [11].



About phone

Status
Phone number, signal, etc.

Legal information

Model number
Android SDK built for x86

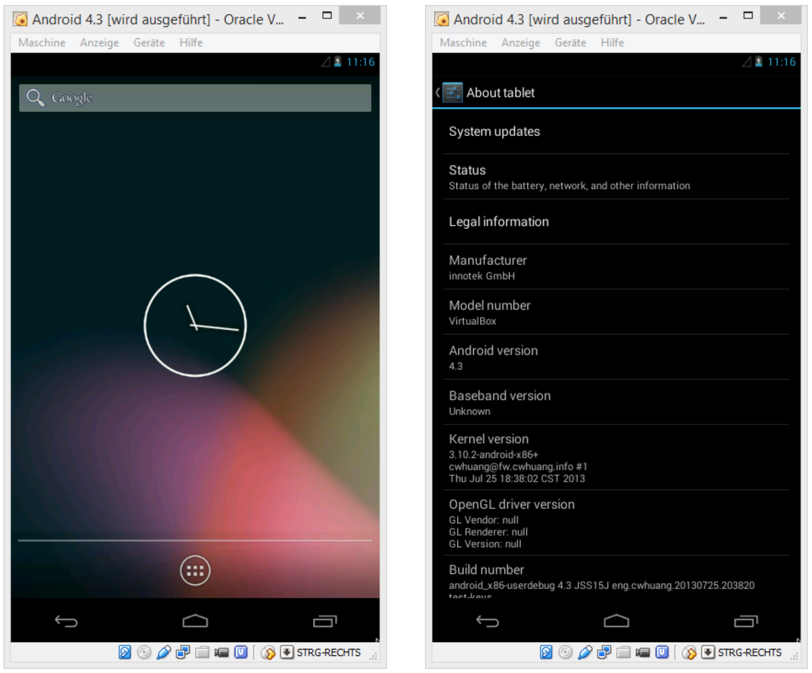
Android version
4.3

Baseband version
Unknown

Kernel version
3.4.0+
android-build@intel-otc #1
Thu Aug 1 17:06:05 CST 2013

Build number
sdk_x86-eng 4.3 JSS15J eng.android-build.20130801.155736 test-keys

Figure 4.2: Android Virtual Device running Android 4.3.


Android 4.3 [wird ausgeführt] - Oracle V...
Maschine Anzeige Geräte Hilfe
11:16
Google
STRG-RECHTS
Android 4.3 [wird ausgeführt] - Oracle V...
Maschine Anzeige Geräte Hilfe
11:16
STRG-RECHTS

About tablet

System updates

Status
Status of the battery, network, and other information

Legal information

Manufacturer
innotek GmbH

Model number
VirtualBox

Android version
4.3

Baseband version
Unknown

Kernel version
3.10.2-android-x86+
cwhuang@fw.cwhuang.info #1
Thu Jul 25 18:38:02 CST 2013

OpenGL driver version
GL Vendor: null
GL Renderer: null
GL Version: null

Build number
android_x86-userdebug 4.3 JSS15J eng.cwhuang.20130725.203820

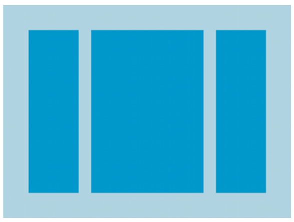
Figure 4.3: Oracle VirtualBox running Android 4.3 from the AndroidX86 project [13] given a custom resolution.

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${packageName}.${activityClass}" >

  <TextView
    android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="25sp"
      android:text="Hello World!" />
</RelativeLayout>
```

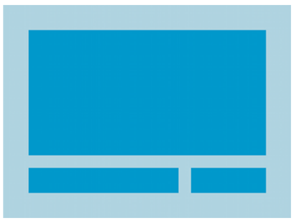
Program 4.1: HelloWorld application generated by using XML.

```
RelativeLayout layout = new RelativeLayout(this);
LayoutParams layoutLp = new LayoutParams(
  LayoutParams.MATCH_PARENT,
  LayoutParams.MATCH_PARENT
);

TextView textView = new TextView(this);
textView.setText("Hello World!");
textView.setTextSize(25);
layout.addView(textView);
```

Program 4.2: HelloWorld application generated programatically.

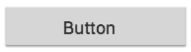


(a) Linear Layout



(b) Relative Layout

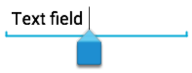
(c) Web View



(d) Button

(e) Label

(f) Slider



(g) Textfield

(h) SwitchButton

(i) Checkbox

(j) RadioButton

Figure 4.4: Android Layouts and Elements: (a) A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen. (b) Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent). (c) Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent). Images and Descriptions taken from [54].

4.2 Server

Figure 4.5 shows a low detailed overview where only the packages and main classes are visible while section 4.2.1 to section 4.2.6 give a more detailed view.

4.2.1 Main

As seen in program 4.3 and program 4.6 the main class of the so called FuelTime-Framework handles the start-up and shutdown phase of the server. On start-up it loads the configuration and initializes the different managers which are described in detail in section 4.2.2 to section 4.2.4 and lines 65 to 69 of program 4.3). If an error occurs while starting up or the server gets shut down, the server ends its managers (see lines 102 to 104 of program 4.6) and displays the occurred error. The logger which is configurated by the log4.properties file has to be loaded before starting the server. Therefore the method defined by program 4.5 is used.

Configuration

To make it easy for developers to configurate the server and that the server must no be re-compiled to use a new configuration, the configuration of all managers is saved in a single XML file (see program 4.4) which is located inside of the Config folder. Another file which lays in the Config folder is the log4j.properties file to configure Log4J which is used for logging.

```
58 public void start() throws Exception{
59   log("FuelSaveGame Server 1.0");
60   log("Design and Implementation: Robert Reichart (S1210629016)");
61   log("");
62   log("Starting FuelSaveGame Server");
63   log("------------------------------------------------------------");
64   try{
65     loadSettings();
66     initEventManager();
67     initMailSender();
68     initModuleManager();
69     initNetworkManager();
70   }catch(Exception ex){
71     stop(ExceptionUtils.getMessage(ex));
72     throw ex;
73   }
```

```
82 }
```

Program 4.3: The method used for starting the server.

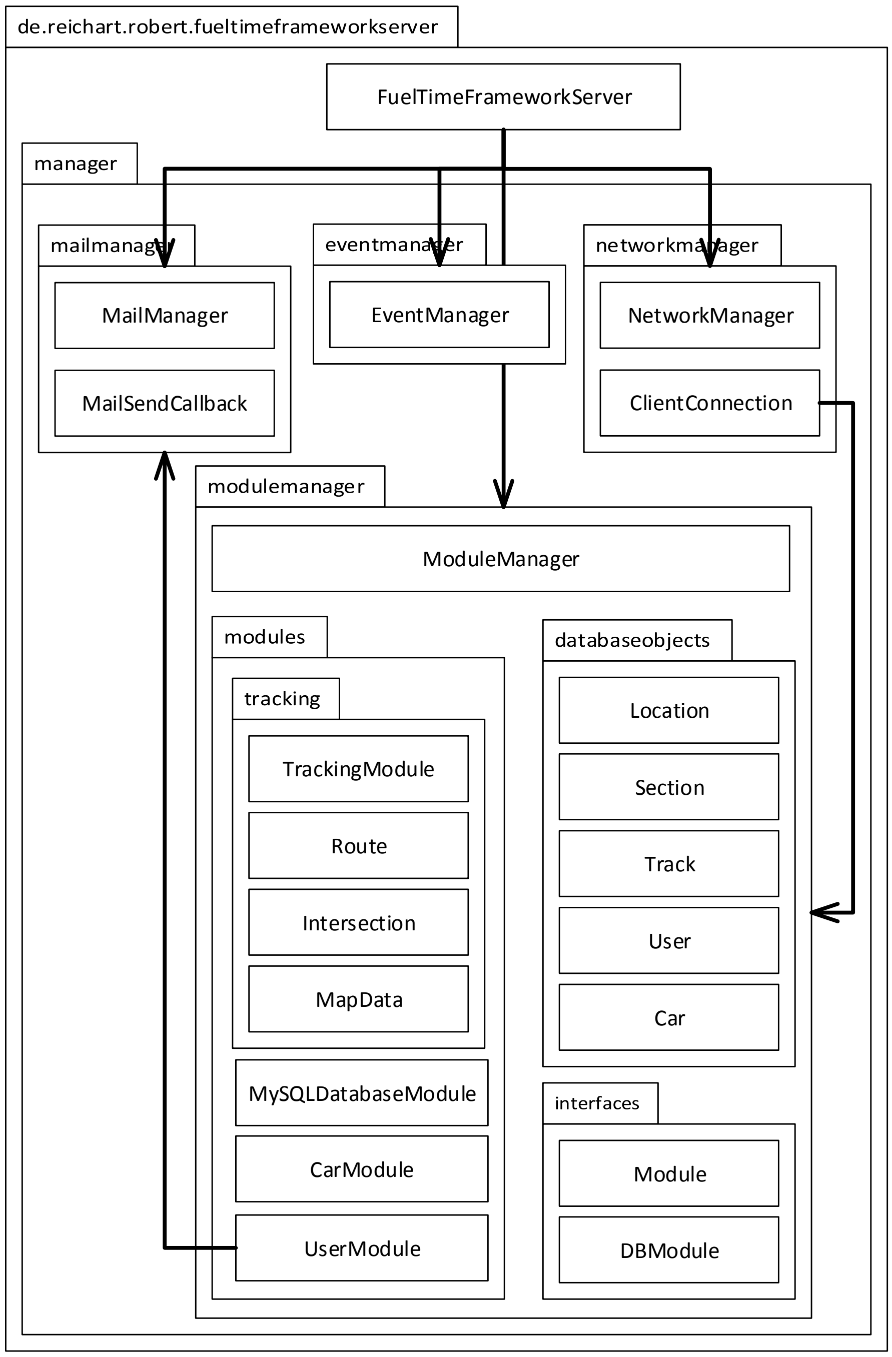


Figure 4.5: The structure of the implemented server.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Server>
  <NetworkManager>
    <Port>2000</Port>
    <ThreadPool>
      <Type>Cached</Type>
      <MaxCount>null</MaxCount>
    </ThreadPool>
  </NetworkManager>
  <MailSender>
    <ThreadPool>
      <Type>Cached</Type>
      <MaxCount>5</MaxCount>
    </ThreadPool>
    <SMTPStartTLS>No</SMTPStartTLS>
    <SMTPHost>smtp.gmail.com</SMTPHost>
    <SMTPPort>587</SMTPPort>
    <SMTPAuth>true</SMTPAuth>
    <SMTPUser>reichart.robert@gmail.com</SMTPUser>
    <SMTPPassword>AngelikaPeter</SMTPPassword>
    <From></From>
    <ReplyTo></ReplyTo>
  </MailSender>
  <MySQLDatabase>
    <Host>localhost</Host>
    <Database>fuelsavegame</Database>
    <User>fuelsavegame</User>
    <Password>PRO2</Password>
  </MySQLDatabase>
  <Modules>
  </Modules>
</Server>
```

Program 4.4: server.xml

```java
public void loadLog4J() throws Exception{
  Properties properties = new Properties();
  properties.load(new File(CONFIG_PATH + "log4j.properties"));
  PropertyConfigurator.configure(properties);
  log = Logger.getRootLogger();
}
```

Program 4.5: The method used for loading Log4J.

```
public void stop(String error){
  if(error!=null){
    if(!error.trim().equals("")){
      error = "\nERROR MESSAGE:\n" + error;
    }
  }else{
    error="";
  }

  log("============================================================");
  log("Server is shutting down");
  try{mailSender.shutdown();}catch(Exception ex){}
  try{networkManager.shutdown();}catch(Exception ex){}
  try{moduleManager.unloadModules();}catch(Exception ex){}
  log(error);
  log("============================================================");
  log("Server ended");
  isRunning=false;
}
```

Program 4.6: The method used for stopping the server.

4.2.2 Network manager

As seen in Figure 4.6 and described in section 3.1.1 and section 3.2.1 the SSL secured connection to the clients is established by the network manager using a threadpool. The threadpool can be created in Java by using the ExecutorService. This can either be a cached threadpool

```
executorService = Executors.newCachedThreadPool();
```

which creates new threads on demand or a fixed threadpool

```
executorService = Executors.newFixedThreadPool(maxThreadCount);
```

which creates the given number of threads when it is created.

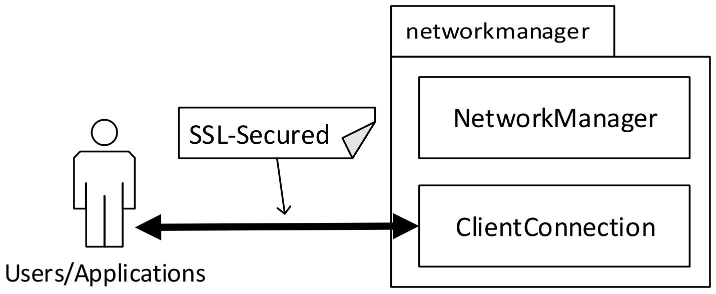


Figure 4.6: Network manager

In Java to establish a connection to a client which should be SSL secured the so called SSLServerSocketFactory is used. It contains the method *createServerSocket(int port)* which returns a traditional Java ServerSocket which can be cast to a SSLServerSocket. Once the secured socket has been created its cipher suites have to be enabled.

```
sslServerSocket = (SSLServerSocket) serverSocketFactory.
    createServerSocket(PORT);
sslServerSocket.setEnabledCipherSuites(sslServerSocket.
    getSupportedCipherSuites());
```

If now a client connects to the server the returned Socket object from the SSLServerSocket's *accept()* method can be cast to a SSLSocket. Giving the SSLSocket to a Runnable which is executed by the ExecuterService using its *execute(Runnable runnable)* method makes it possible to handle every client in its own thread.

```
SSLSocket sslSocket = (SSLSocket)sslServerSocket.accept();
executorService.execute(new ClientConnection(this,sslSocket));
```

ClientConnection

As seen in the lines of the above shown code-snippet, for every client which connects to the server a ClientConnection object gets created. The lines below show the main part of the ClientConnection. Whenever a client sends a message, a JSONObject [30] gets created and is given to the module manager (see section 4.2.3) which passes it to the loaded modules and returns a JSONArray (containing one JSONObject per module) which then gets sent back to the client.

```
JSONObject request = new JSONObject(inputLine);

JSONArray response = moduleManager.getResponse(socket,request);
output.write(response.toString()+"\n");
output.flush();
```

4.2.3 Module manager

The main part of the module manager loads all modules and handles incoming messages from clients by passing them through to every module which is registered for the specific type specified in the clients request (see program 4.7).

The user module, tracking module and MySQL-database module which are described below and which are provided by the framework, follow specific guidelines described below to get automatically loaded by the module manager:

Folder

All modules must lay inside the modules folder or one of its sub folders.

Naming

Databasemodule

Every databasemodules name has to end with 'DatabaseModule' to be loaded as a database module.

Generic Modules

A module whose name ends with 'Module' is automatically recognized as a module. Otherwise its name has to be saved in the configuration file described in section 4.2.1.

Abstract Class

A module has to extend the abstract class 'Module' (see program 4.8). A databasemodule has to extend the abstract class 'DatabaseModule' (see program 4.9).

```
public JSONArray getResponse(Socket socket, JSONObject request){
  JSONArray response = new JSONArray();

  for(Module module: modules.get(request.getString("TYPE"))){
    response.put(module.handle(socket,request));
  }

  return response;
}
```

Program 4.7: Module manager

```
public abstract class Module {
  public Module(FuelTimeFrameworkServer server){}

  public abstract void stop();
  public abstract String getType();
  public abstract JSONObject handle(Socket socket, JSONObject toHandle);
}
```

Program 4.8: Module Interface

```java
public abstract class DatabaseModule {
  public DatabaseModule(FuelTimeFrameworkServer server){}
  public abstract void stop();

  public abstract User getUserById(int id);
  public abstract User getUserByEmail(String email);
  public abstract void saveUser(User user);
  public abstract void deleteUser(User user);

  public abstract Car getCarById(int id);
  public abstract Car getCarByDetails(String manufacturer,String model,
    String engine, long buildFrom, long buildTo);
  public abstract ArrayList<String> getCarManufacturers();
  public abstract ArrayList<String> getCarModels(String manufacturer);
  public abstract ArrayList<String> getCarEngines(String manufacturer,
    String model);
  public abstract ArrayList<HashMap<String,String>>
    getCarProductionDates(String manufacturer, String model, String
    engine);

  public abstract void saveTrack(Track track);
  public abstract ArrayList<Track> getTracksBySectionID(long id);

  public abstract Section getSectionByRoadID(int id, Location
    startLocation);
  public abstract ArrayList<Section> getSectionsByStartPoint(Location
    startLocation);
}
```

Program 4.9: DatbaseModule Interface

User module

The user module manages the creation which means registering and activating of a user. Another part of the user module is adding and deleting cars from the users cars list. To create an account a user has to register (using a valid email address and password) and activate the account using the activation code which is sent by email by using the mail manager which is particularly described in section 4.2.4.

Whenever a user gets created or the users data changes, the user is saved in a database using the MySQL-database module described in section 4.2.3.

Trackingmodule

As described in section 3.2.1 the tracking module which lays in the *tracking* folder inside the *modules* folder, handles the main work of the framework which is receiving and responding car telemetries.

In a first step the users location conditions have to be determined to give clients suitable car telemetries corresponding to their current location. In this implementations case the condition to the users location are retrieved using the overpass-api [45] (see program 4.12). In the case that the users location can be assumed as a streets intersection the tracking information of all users from this streets section (defined by Equation 3.1 and the users current location as end location l_{end}) using a car with similar fuel consumption average to the clients car is taken into account to calculate the users "score" using Equation 3.3 (see program 4.10).

To give the user time to react, the user must be provided with car telemetries of the section which is in front. To predict which street the user will drive next and to support users in time saving driving, the server has to be able to calculate a route to the users destination (defined by its latitude and longitude). This can be achieved by using the graphhopper library [21] (see program 4.11).

A way which is used in this implementation to get an addresses latitude and longitude is by using a nominatim service (see 4.13).

```
//Calculating the average time and fuel of all tracks
double averageTimeSum = 0;
double averageFuelSum = 0;
for(Track track: tracks){
  averageTimeSum+=track.getDeltaTimeInSeconds();
  averageFuelSum+=track.getFuelAverage();
}
double averageTime = averageTimeSum/tracks.size();
double averageFuel = averageFuelSum/tracks.size();

//Calculating each tracks score
for(Track track: tracks){
  double percentTimeDiff = (track.getDeltaTimeInSeconds()/averageTime);
  double percentFuelDiff = (track.getFuelAverage()/averageFuel);
  double score = percentTimeDiff + percentFuelDiff;
  track.setScore(score);

}
```

Program 4.10: Calculating track scores.

```
GHRequest routeRequest = new GHRequest(
  start.getLatitude(),
  start.getLongitude(),
  end.getLatitude(),
  end.getLongitude()
);
GHResponse routeResponse = graphhopper.route(routeRequest);
```

Program 4.11: Calculating a route using the Graphhopper library [21].

```
private MapData getMapData(Location location, double meters) throws
    Exception{
  try {
    final String QUERY = "<query type=\"way\"><around lat=\""+location.
    getLatitude()+"\" lon=\""+location.getLongitude()+"\" radius=\""+
    meters+"\"/></query><union><item/><recurse type=\"down\"/></union><
    print/>";

    HttpClient httpclient = HttpClientBuilder.create().build();
    HttpPost httppost = new HttpPost("http://overpass-api.de/api/
    interpreter");

    List<NameValuePair> nameValuePairs=new ArrayList<NameValuePair>(1);
    nameValuePairs.add(new BasicNameValuePair("data", QUERY));
    httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

    ResponseHandler<String> responseHandler=new BasicResponseHandler();
    String responseBody = httpclient.execute(httppost, responseHandler);
    return new MapData(responseBody);
  } catch (Exception e) {
    return null;
  }
}
```

Program 4.12: Retrieving MapData using the Overpass API.

```
URI uri = new URIBuilder()
  .setScheme("http")
  .setHost("nominatim.openstreetmap.org")
  .setPath("/search.php")
  .setParameter("q", new String (request.getString("ADDRESS").getBytes()
    ,"UTF-8"))
  .setParameter("format", "json")
  .setParameter("polygon", "1")
  .build();

CloseableHttpResponse httpResponse = HttpClients.createDefault().execute
    (new HttpGet(uri));
```

Program 4.13: Getting a adressess latitude and longitude using a nominatim service.

MySQLDatabaseModule

The MySQLDatabaseModule extends the abstract class 'DatabaseModule' shown by program 4.9 such that the server can call the specified methods without knowing anything about the database structure behind. To connect to the database the JDBC-Driver [37] and the configuration entries in the configuration file (see program 4.4) are used.

Because of the users email address and password should not be stored in human readable form they have to be encrypted. To encode human readable data the method shown by program 4.14 is used while the method shown by program 4.15 is used to decode encrypted data to human readable form. Before encoding or decoding the cipher has to be initialised by using program 4.16. The encryption used in this thesis is based on [12].

```
public static String encode(String plainText){
  init();
  try{
    byte[] cleartext = plainText.getBytes("UTF8");
    Cipher cipher = Cipher.getInstance("DES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    return base64encoder.encode(cipher.doFinal(cleartext));
  }catch(Exception ex){ex.printStackTrace();return null;}
}
```

Program 4.14: Method used for encoding a String.

```java
public static String decode(String cipherText){
  init();
  try{
    byte[] encrypedPwdBytes = base64decoder.decodeBuffer(cipherText);
    Cipher cipher = Cipher.getInstance("DES");
    cipher.init(Cipher.DECRYPT_MODE, key);
    byte[] plainTextPwdBytes = (cipher.doFinal(encrypedPwdBytes));
    return new String(plainTextPwdBytes, "UTF-8");
  }catch(Exception ex){ex.printStackTrace();return null;}
}
```

Program 4.15: Method used for decoding a String.

```java
private static void init(){
  if(keyFactory==null){
    try{
      keyFactory = SecretKeyFactory.getInstance("DES");
    }catch(Exception ex){ex.printStackTrace();}
  }
  if(keySpec==null){
    try{
      keySpec = new DESKeySpec("RobertReichartS1210629016".getBytes("
      UTF8"));
    }catch(Exception ex){ex.printStackTrace();}
  }
  if(key==null){
    try{
      key  = keyFactory.generateSecret(keySpec);
    }catch(Exception ex){ex.printStackTrace();}
  }
}
```

Program 4.16: Initializing the cipher.

4.2.4 Mailmanager

The main task of the MailManager is to send the activationcode to newly created users. As seen in program 4.4 the MailManager is configurated by the configuration file. To send a mail by using JavaMail [29], a session is needed. The session gets created by passing the mail servers' login credentials as an argument. After the session has been created, the transport object has to be created using the transport method as argument. Once the transport object is created, it is connected to the mail server using the earlier provided login credentials (see lines 62 to 64 of program 4.17). In a final step, the email itself which is handled as a message gets sent to its recipients using the earlier created transport object (see lines 146 to 159 of program 4.17).

```
session = Session.getDefaultInstance(sessionProperties, null);
transport = session.getTransport("smtp");
transport.connect(HOST,USER,PASSWORD);
```

```
Message message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.setReplyTo(InternetAddress.parse(replyTo));
message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(
    recipient));
message.setSubject(subject);
message.setText(text);
```

```
transport.sendMessage(message, message.getAllRecipients());
```

Program 4.17: Sending a mail using JavaMail.

4.2.5 EventManager

The Eventmanager shown by program 4.18 handles the registration and deleting of EventListeners as well as the directing of Events (which are only containing the constructor to set the events type, message and data object and getters to access those values) to the EventListeners which were registered to the events type (see Table 4.2).

Value	**Name**
1	USER_CREATED
2	USER_ACTIVATED
3	CLIENT_CONNECTED
4	CLIENT_DISCONNECTED
5	EMAIL_SENT

Table 4.2: Event types

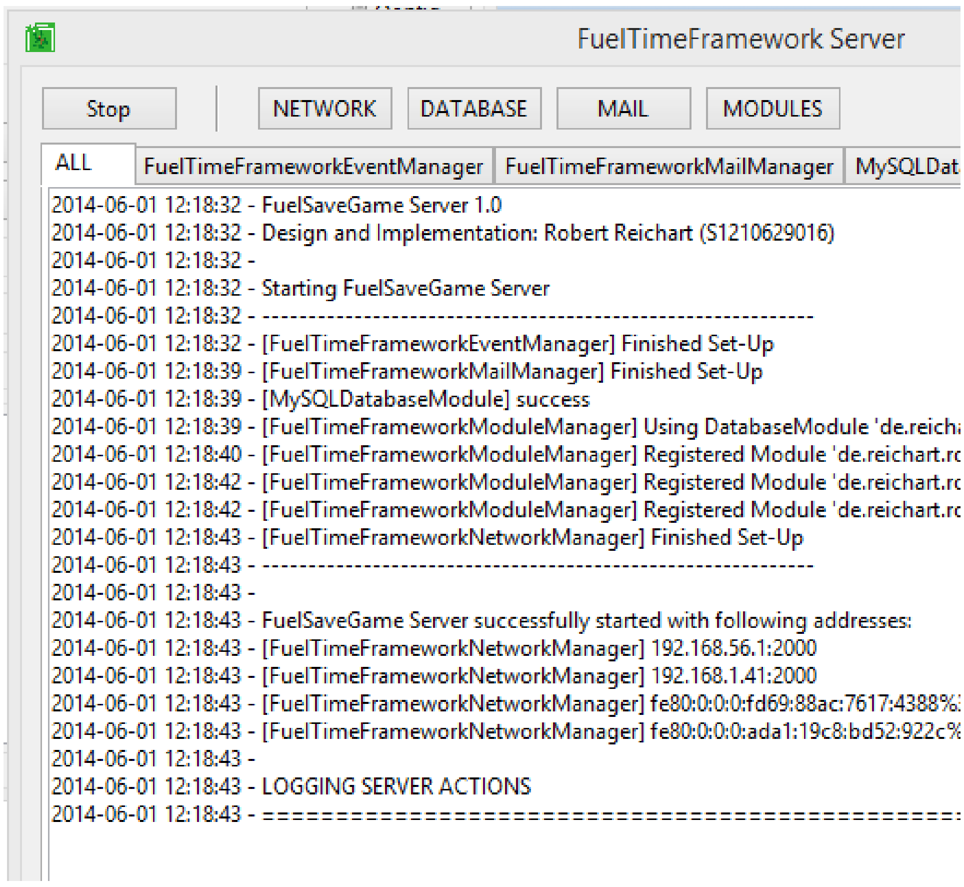
```java
public void handle(FuelTimeFrameWorkEvent fuelTimeFrameWorkEvent){
  if(listeners.get(fuelTimeFrameWorkEvent.getType())!=null){
    for(FuelTimeFrameWorkEventListener listener: listeners.get(
    fuelTimeFrameWorkEvent.getType())){
      listener.onEvent(fuelTimeFrameWorkEvent);
    }
  }
}

public void addEventListener(int eventType,
    FuelTimeFrameWorkEventListener listener){
  if(listeners.get(eventType)==null){
    listeners.put(eventType, new ArrayList<
    FuelTimeFrameWorkEventListener>());
  }
  listeners.get(eventType).add(listener);
}

public void removeEventListener(FuelTimeFrameWorkEventListener listener)
    {
  for(ArrayList<FuelTimeFrameWorkEventListener> list: listeners.values()
    ){
    list.remove(listener);
  }
}
```

Program 4.18: Event manager

4.2.6 Graphical user interface

Figure 4.7 and Figure 4.9 are showing the server's final graphical user interface which makes it easier for administrators to use and configure the server. The application is implemented by using the Standard Widget Toolkit (SWT) and changes the log4j properties in a way that whenever a manager or module writes to the log a new tabular with only the managers or modules output gets opened and no output is written to the console.

As shown by Figure 4.8 to Figure 4.9c the server can easily be configured by using the graphical user interface such that the servers configuration file must not be edited by hand which could cause problems.



```
2014-06-01 12:18:32 - FuelSaveGame Server 1.0
2014-06-01 12:18:32 - Design and Implementation: Robert Reichart (S1210629016)
2014-06-01 12:18:32 -
2014-06-01 12:18:32 - Starting FuelSaveGame Server
2014-06-01 12:18:32 - ----------------------------------------------------------
2014-06-01 12:18:32 - [FuelTimeFrameworkEventManager] Finished Set-Up
2014-06-01 12:18:39 - [FuelTimeFrameworkMailManager] Finished Set-Up
2014-06-01 12:18:39 - [MySQLDatabaseModule] success
2014-06-01 12:18:39 - [FuelTimeFrameworkModuleManager] Using DatabaseModule 'de.reich
2014-06-01 12:18:40 - [FuelTimeFrameworkModuleManager] Registered Module 'de.reichart.r
2014-06-01 12:18:42 - [FuelTimeFrameworkModuleManager] Registered Module 'de.reichart.r
2014-06-01 12:18:42 - [FuelTimeFrameworkModuleManager] Registered Module 'de.reichart.r
2014-06-01 12:18:43 - [FuelTimeFrameworkNetworkManager] Finished Set-Up
2014-06-01 12:18:43 - ----------------------------------------------------------
2014-06-01 12:18:43 -
2014-06-01 12:18:43 - FuelSaveGame Server successfully started with following addresses:
2014-06-01 12:18:43 - [FuelTimeFrameworkNetworkManager] 192.168.56.1:2000
2014-06-01 12:18:43 - [FuelTimeFrameworkNetworkManager] 192.168.1.41:2000
2014-06-01 12:18:43 - [FuelTimeFrameworkNetworkManager] fe80:0:0:0:fd69:88ac:7617:4388%
2014-06-01 12:18:43 - [FuelTimeFrameworkNetworkManager] fe80:0:0:0:ada1:19c8:bd52:922c%
2014-06-01 12:18:43 -
2014-06-01 12:18:43 - LOGGING SERVER ACTIONS
2014-06-01 12:18:43 - ==================================================
```

Figure 4.7: Graphical User Interface

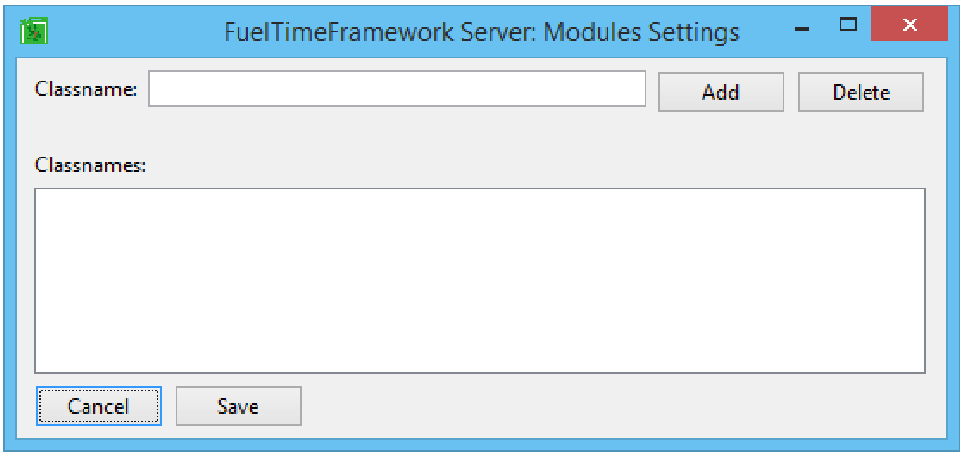
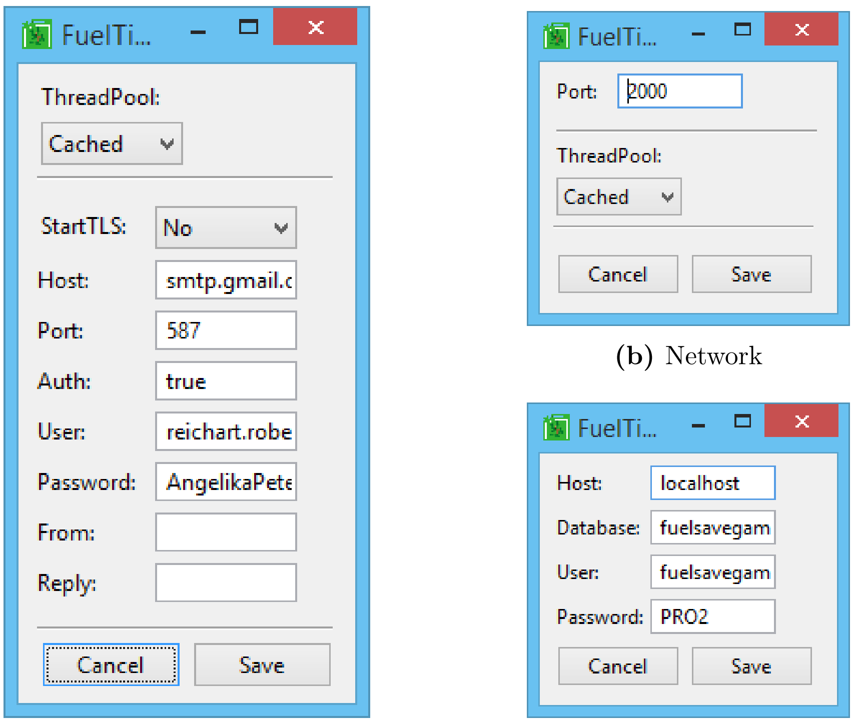


Figure 4.8: Module Classnames



ThreadPool:	Cached
StartTLS:	No
Host:	smtp.gmail.c
Port:	587
Auth:	true
User:	reichart.robe
Password:	AngelikaPete
From:	
Reply:	

(a) Mailsender

Port:	2000
ThreadPool:	Cached

(b) Network

Host:	localhost
Database:	fuelsavegam
User:	fuelsavegam
Password:	PRO2

(c) Database

Figure 4.9: Mailsender, Network and Database settings.

4.3 Client

4.3.1 Implementation

Figure 4.10 shows a low detailed (only packages and classes) overview while section 4.3.2 to section 4.3.6 give a more detailed view.

4.3.2 Main

The clients main part (see Figure 4.11 and program 4.19) which is located inside the FuelTimeFrameworkClient class initializes the network manager (see section 4.3.3), the user manager (see section 4.3.4), the telemetry manager (see section 4.3.5) and the view manager (see section 4.3.6).

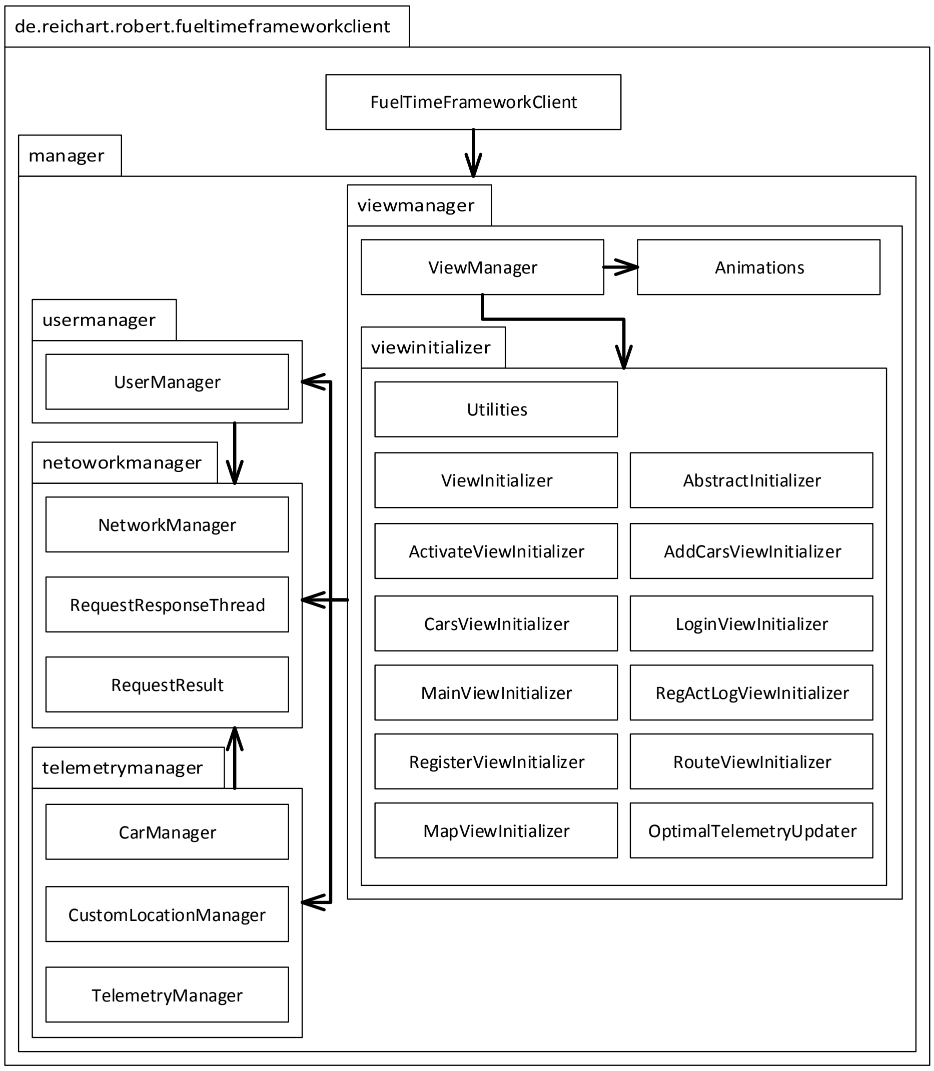


Figure 4.10: The implemented Client structure.

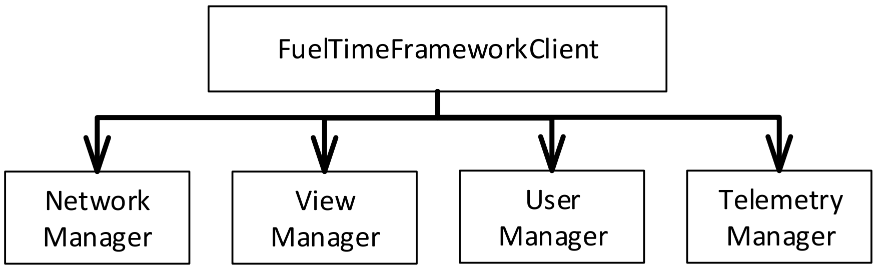


Figure 4.11: The main part of the client: Initializing all used managers.

```
networkManager = NetworkManager.getInstance(IP,PORT);
userManager = UserManager.getInstance(this);
telemetryManager = TelemetryManager.getInstance(this);
viewManager = ViewManager.getInstance(this);
viewManager.addViewInitializer(new MainViewInitializer(this));
viewManager.addViewInitializer(new MapViewInitializer(this));
viewManager.addViewInitializer(new CarsViewInitializer(this));
viewManager.addViewInitializer(new RegActLogViewInitializer(this));
viewManager.addViewInitializer(new RegisterViewInitializer(this));
viewManager.addViewInitializer(new ActivateViewInitializer(this));
viewManager.addViewInitializer(new LoginViewInitializer(this));
viewManager.addViewInitializer(new AddCarsViewInitializer(this));
```

Program 4.19: The main part of the client: Initializing all used managers.

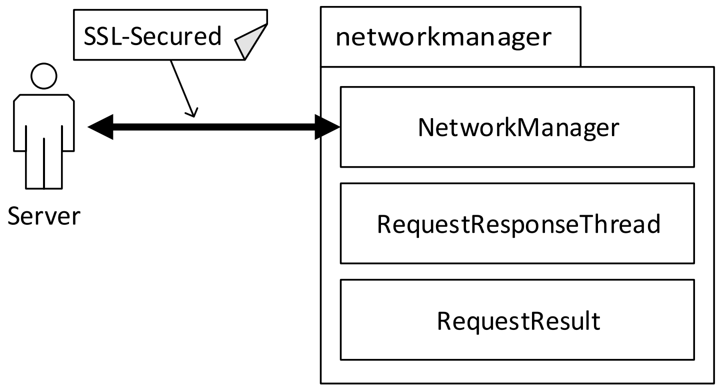


Figure 4.12: Network manager

4.3.3 Network manager

The task of the network manager shown in Figure 4.12 is to secure the connection to the server SSL and handle the requests to and responses from the server (described in section 3.1.1 and section 3.2.3). To do so in Android the servers address and listening port are passed to the SSLServerSocketFactory which creates a Socket which can be cast to a SSLSocket. Once the SSLSocket is created, its cipher suites have to be enabled.

```
sslSocket = (SSLSocket)socketFactory.createSocket(IP, PORT);
sslSocket.setEnabledCipherSuites(sslSocket.getSupportedCipherSuites());
```

The previous established connection can now be used to send messages to and receive messages from the server. In Android this has to be done in a extra thread because of the possible blocking reading and writing methods of the socket, otherwise the method calls would end in an exception. To handle writing and reading in a synchronous way while using an extra thread, the methods wait() and notify() could be used (see program 4.20).

```
public JSONArray requestResponse(JSONObject request,int timeout) throws
    IOException{
  String ticketName = generateTicketName();
  new RequestResponseThread(this,ticketName,request,timeout);
  while(!requestResponseTickets.containsKey(ticketName)){
    synchronized(this){
      try{wait();}catch(Exception ex){}
    }
  }
  RequestResult result = requestResponseTickets.remove(ticketName);
  if(result.isValid()){
    return result.getResult();
  }else{
    throw result.getException();
  }
}

protected void setRequestResult(String ticket, RequestResult result) {
  requestResponseTickets.put(ticket, result);
  synchronized(this){notifyAll();}
}
```

Program 4.20: Handling asynchronous requests in a synchronous way.

4.3.4 User manager

The user manager provides methods for user account management. This includes registering, activating and login into an account. To automatically login, the account credentials should be stored on the device. To achieve this in Android the SharedPreferences could be used. To read from there, they have to be loaded by following line of code.

```
sharedPreferences = main.getPreferences(Context.MODE_PRIVATE);
```

By using a *getType* method to request a value of the SharedPreferences a default value has to be passed which gets returned if the requested value cannot be found (see following code).

```
return Encryption.decode(sharedPreferences.getString("user_email", ""));
```

To store data in the shared preferences the editor object has to be used which can be got by calling the *edit()* function from the SharedPreferences object. After the data has been created or updated the *commit()* method has to be called to take effect.

```
editor.putString("user_email", Encryption.encode(email));
editor.putString("user_password", Encryption.encode(password));
editor.commit();
```

Because of the users email address and password should not be stored in human readable form they have to be encrypted. To encode human readable data the method shown by program 4.21 is used while program 4.22 is used to decode encrypted data to human readable form. Before encoding or decoding the cipher has to be initialized by the method shown in program 4.23. The encryption used in this thesis is based on [12].

```
public static String encode(String plainText){
  init();
  try{
    byte[] cleartext = plainText.getBytes("UTF8");

    Cipher cipher = Cipher.getInstance("DES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    return Base64.encodeToString(cipher.doFinal(cleartext),Base64.
    DEFAULT);

  }catch(Exception ex){Log.e("Encryption","",ex);return "";}
}
```

Program 4.21: Method used for encoding a String.

```java
public static String decode(String cipherText){
  init();
  try{
    byte[] encrypedPwdBytes = Base64.decode(cipherText,Base64.DEFAULT);

    Cipher cipher = Cipher.getInstance("DES");// cipher is not thread safe
    cipher.init(Cipher.DECRYPT_MODE, key);
    byte[] plainTextPwdBytes = cipher.doFinal(encrypedPwdBytes);

    return new String(plainTextPwdBytes, "UTF-8");
  }catch(Exception ex){
    Log.e("Encryption","",ex);
    return "";
  }
}
```

Program 4.22: Method used for decoding a String.

```java
private static void init(){
  if(keyFactory==null){
    try{
      keyFactory = SecretKeyFactory.getInstance("DES");
    }catch(Exception ex){Log.e("Encryption","",ex);}
  }
  if(keySpec==null){
    try{
      keySpec = new DESKeySpec("RobertReichartS1210629016".getBytes("
    UTF8"));
    }catch(Exception ex){Log.e("Encryption","",ex);}
  }
  if(key==null){
    try{
      key  = keyFactory.generateSecret(keySpec);
    }catch(Exception ex){Log.e("Encryption","",ex);}
  }
}
```

Program 4.23: Initializing the cipher.

4.3.5 Telemetry manager

The telemetry manager shown by Figure 4.13 acts as a facade to the CarManager and the CustomLocationManager.

To gain the current users location, the CustomLocationManager is used. It simply acts as a LocationListener to the Androids built-in LocationManager. Because the locations should be as precise as possible, only locations retrieved by GPS are used.

The implementation of the CarManager contains until now only setter and getter methods for the actual speed, gear and fuel consumption. Later on it should retrieve information about the cars telemetry using the cars OBD2 diagnose interface (see section 6.1).

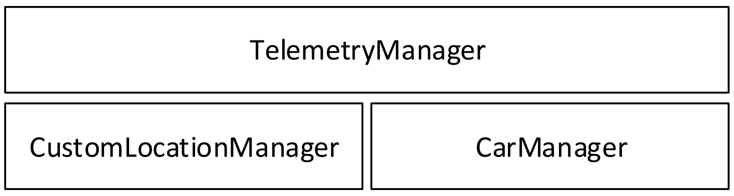


Figure 4.13: TelemetryManager

4.3.6 View manager

The methods of the view manager (shown by Figure 4.14 and program 4.25) make it possible to transit between different views. This is achieved by using Androids ViewFlipper which behaves like an array of views where views can easily be added (shown by line 46) or removed (shown by line 61). While lines 47 to 49 show the few steps which are needed to switch to the next view lines 58 to 60 show how to switch to the previous view. The lines shown by program 4.24 can be used to create a simple animation where the next view slides in from the right hand side.

```
public static Animation inFromRightAnimation(){
  Animation inFromRight = new TranslateAnimation(
    Animation.RELATIVE_TO_PARENT, +1.0f,
    Animation.RELATIVE_TO_PARENT, 0.0f,
    Animation.RELATIVE_TO_PARENT, 0.0f,
    Animation.RELATIVE_TO_PARENT,   0.0f
  );
  inFromRight.setDuration(ANIMATION_LENGTH);
  inFromRight.setInterpolator(new AccelerateInterpolator());
  return inFromRight;
}
```

Program 4.24: Method used for generating a in from right Animation.

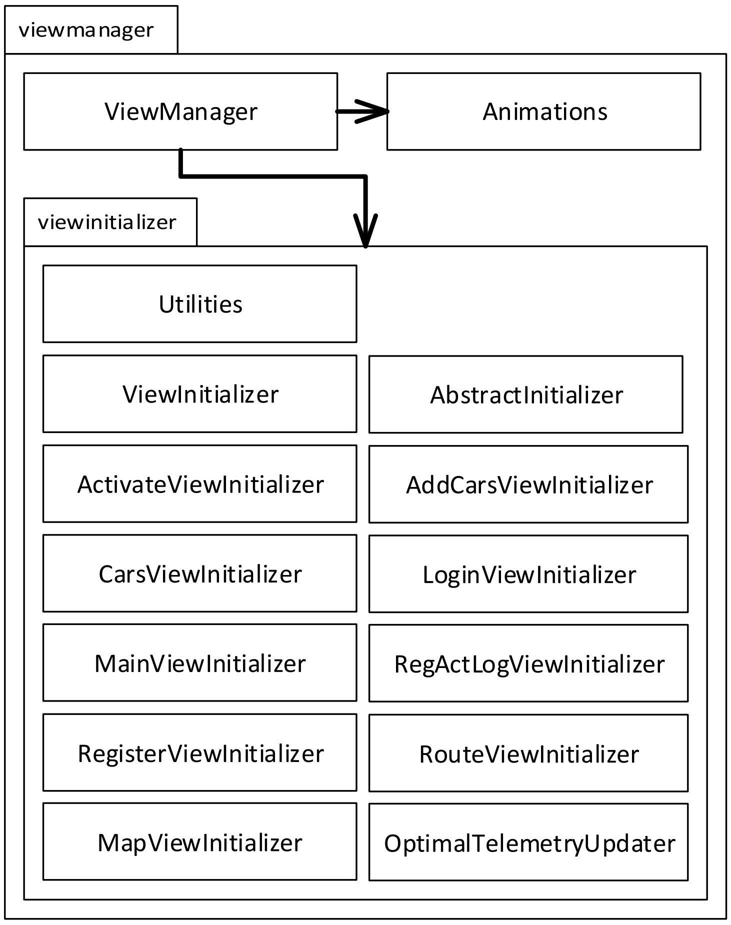


Figure 4.14: ViewManager

```
public void addViewInitializer(ViewInitializer viewInitializer){
viewInitializerByRootLayoutID.put(viewInitializer.getRootLayoutID(),
    viewInitializer);
  viewInitializerByLayoutResourceID.put(viewInitializer.
    getLayoutResourceID(), viewInitializer);
  if(viewInitializerByRootLayoutID.size()==1){
      viewFlipper.addView(inflater.inflate(viewInitializer.
    getLayoutResourceID(), null));
    try{viewInitializer.initialize();}catch(Exception ex){}
    viewFlipper.showNext();
  }
}

public void showNext(int layoutID) throws Exception{
  viewInitializerByRootLayoutID.get(viewFlipper.getChildAt(viewFlipper.
    getChildCount()-1).getId()).deinitialize();
  viewFlipper.addView(inflater.inflate(layoutID, null));
  viewFlipper.setInAnimation(Animations.inFromRightAnimation());
  viewFlipper.setOutAnimation(Animations.outToLeftAnimation());
  viewFlipper.showNext();

  viewInitializerByLayoutResourceID.get(layoutID).initialize();

}

public void showPrevious() throws IOException{
  viewInitializerByRootLayoutID.get(viewFlipper.getChildAt(viewFlipper.
    getChildCount()-1).getId()).deinitialize();
  viewInitializerByRootLayoutID.get(viewFlipper.getChildAt(viewFlipper.
    getChildCount()-2).getId()).initialize();
  viewFlipper.setInAnimation(Animations.inFromLeftAnimation());
  viewFlipper.setOutAnimation(Animations.outToRightAnimation());
  viewFlipper.showPrevious();
  viewFlipper.removeViewAt(viewFlipper.getChildCount()-1);
}
```

Program 4.25: ViewManager

MapView

On Android devices, creating a map which shows the users current location can be done by using the Osmdroid library [44]. The library provides a simple map using map-tiles and data from OpenStreetMap [41]. Because maps from OpenStreetMap are collected by users and do not underlay any license, they are free to use.

Following lines could be used to initialize a map which is created by Androids layout XML.

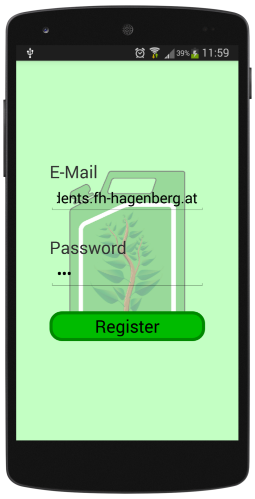
```
MapView mapView = (MapView)main.findViewById(R.id.mapView);
mapView.setTileSource(TileSourceFactory.MAPQUESTOSM);
```

Changing the maps current location, orientation and zoom level can be done with the lines below.

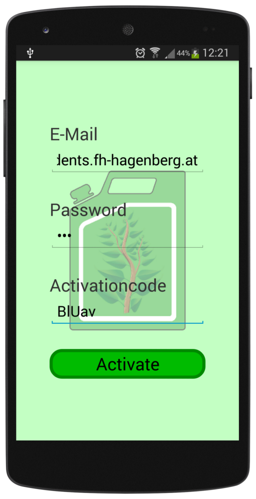
```
mapView.getController().setZoom(mapView.getMaxZoomLevel());
mapView.setMapOrientation(mapOrientation);
mapView.getController().animateTo(new GeoPoint(location));
```

User Interface

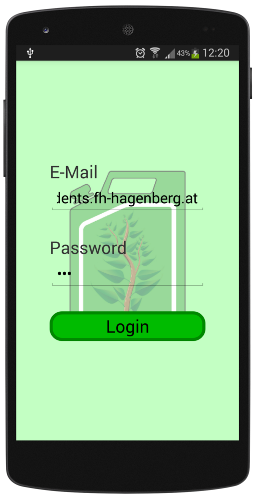
Figure 4.15, Figure 4.16 and Figure 4.17 are showing screenshots of the finished Android client user interface.



(a) Registration

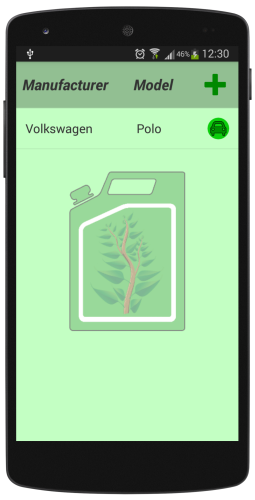


(b) Activation

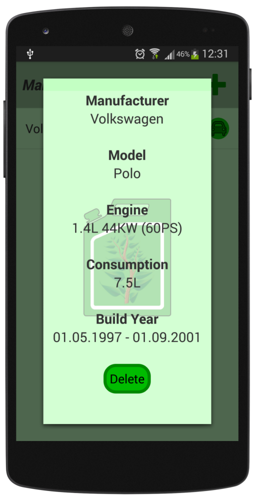


(c) Login

Figure 4.15: User Registration, Activation and Login View.


Manufacturer Model
Volkswagen Polo

(a) Overview


Manufacturer
Volkswagen

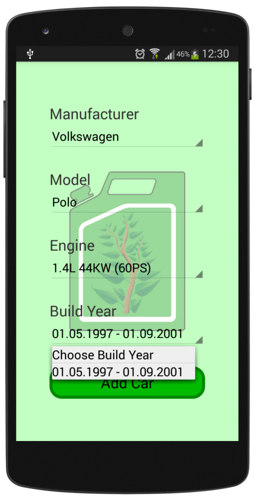
Model
Polo

Engine
1.4L 44KW (60PS)

Consumption
7.5L

Build Year
01.05.1997 - 01.09.2001

(b) Car details


Manufacturer
Volkswagen

Model
Polo

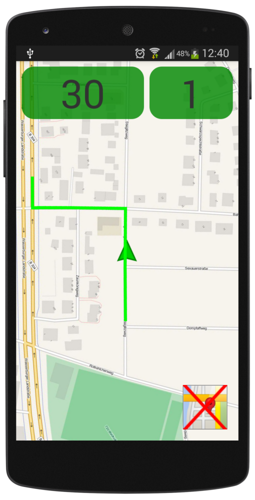
Engine
1.4L 44KW (60PS)

Build Year
01.05.1997 - 01.09.2001

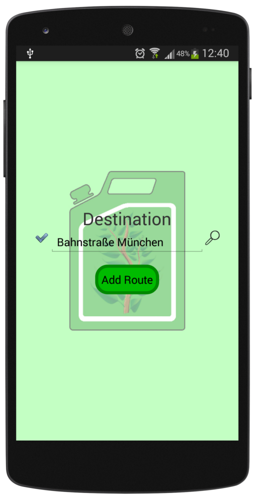
Choose Build Year
01.05.1997 - 01.09.2001

(c) Adding a car

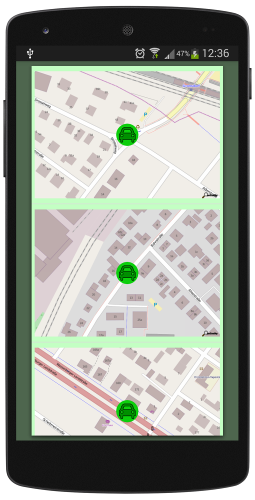
Figure 4.16: Carlist, car details and adding a car.



(a) Navigating


Destination
Bahnstraße München

(b) Add Route



(c) Destinations

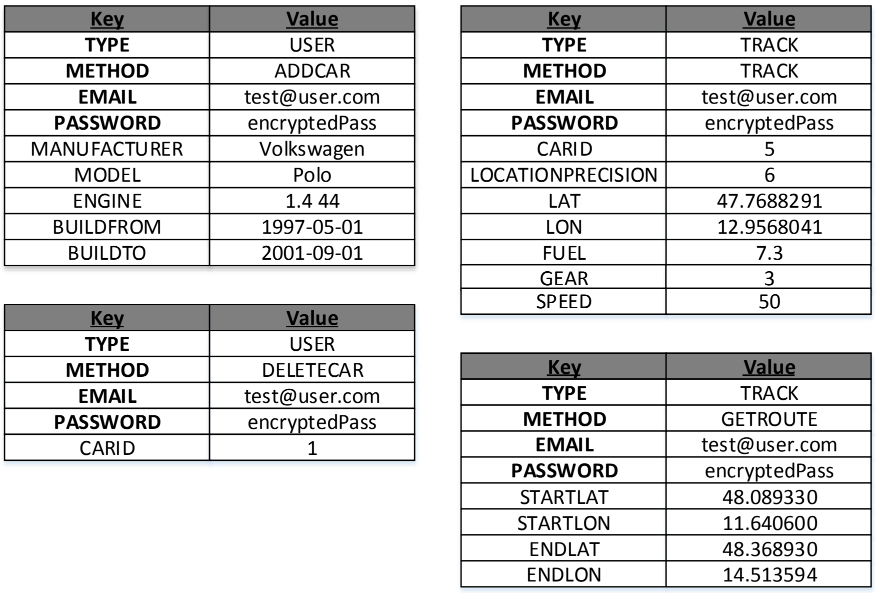
Figure 4.17: Navigating, adding a route and available destinations.

4.4 Protocol

In this implementation of the concept, a stateless and request/response based protocol is used because clients are most of the time connected via mobile internet which can easily break down and has a limited traffic.

Whenever a request should be sent to the server, a JSONObject containing uppercase values for at least the uppercase keys 'TYPE', 'METHOD', 'EMAIL' and 'PASSWORD' has to be sent. The servers response to each request consists of a JSONArray containing one JSONObject per module which is registered to in the request specified type.

Figure 4.19 shows some special responses to given requests of type 'USER'. Figure 4.18 shows the most common requests while Figure 4.20 shows the most common responses.



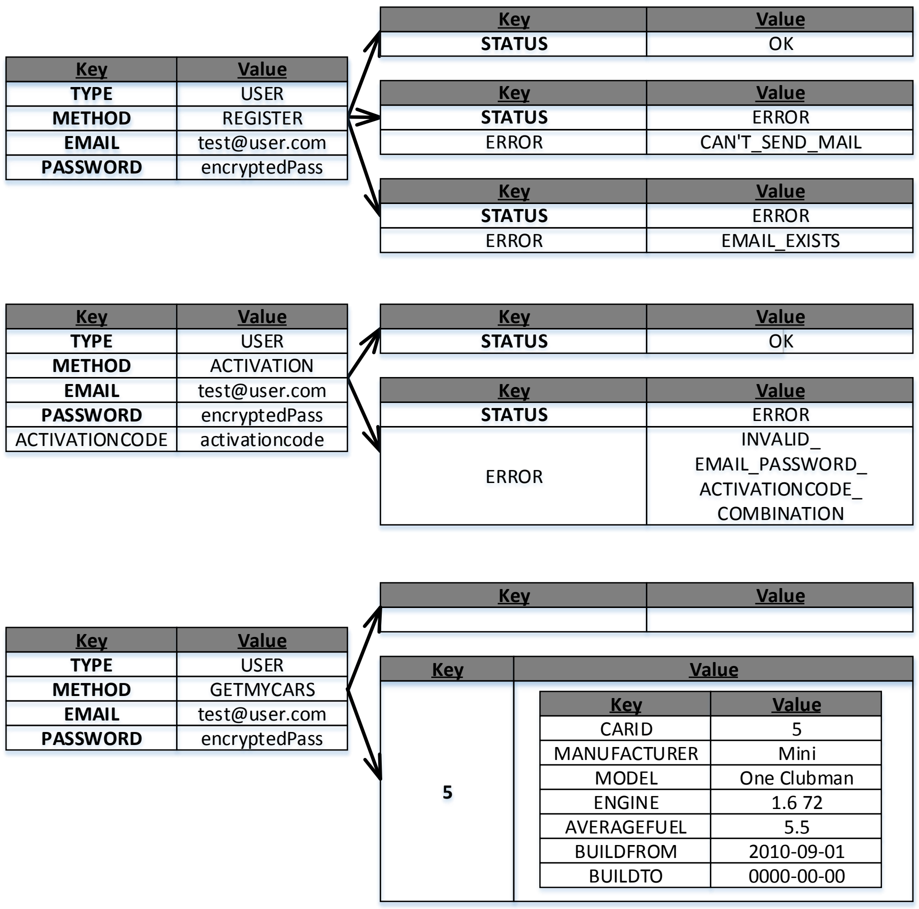
Key	Value
TYPE	USER
METHOD	ADDCAR
EMAIL	test@user.com
PASSWORD	encryptedPass
MANUFACTURER	Volkswagen
MODEL	Polo
ENGINE	1.4 44
BUILDFROM	1997-05-01
BUILDTO	2001-09-01

Key	Value
TYPE	USER
METHOD	DELETECAR
EMAIL	test@user.com
PASSWORD	encryptedPass
CARID	1

Key	Value
TYPE	TRACK
METHOD	TRACK
EMAIL	test@user.com
PASSWORD	encryptedPass
CARID	5
LOCATIONPRECISION	6
LAT	47.7688291
LON	12.9568041
FUEL	7.3
GEAR	3
SPEED	50

Key	Value
TYPE	TRACK
METHOD	GETROUTE
EMAIL	test@user.com
PASSWORD	encryptedPass
STARTLAT	48.089330
STARTLON	11.640600
ENDLAT	48.368930
ENDLON	14.513594

Figure 4.18: Most common requests.



Key	Value
TYPE	USER
METHOD	REGISTER
EMAIL	test@user.com
PASSWORD	encryptedPass

Key	Value
STATUS	OK

Key	Value
STATUS	ERROR
ERROR	CAN'T_SEND_MAIL

Key	Value
STATUS	ERROR
ERROR	EMAIL_EXISTS

Key	Value
TYPE	USER
METHOD	ACTIVATION
EMAIL	test@user.com
PASSWORD	encryptedPass
ACTIVATIONCODE	activationcode

Key	Value
STATUS	OK

Key	Value
STATUS	ERROR
ERROR	INVALID_ EMAIL_PASSWORD_ ACTIVATIONCODE_ COMBINATION

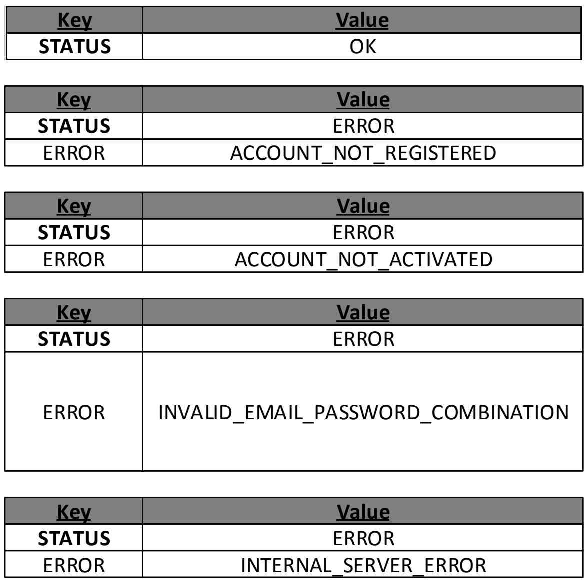
Key	Value
TYPE	USER
METHOD	GETMYCARS
EMAIL	test@user.com
PASSWORD	encryptedPass

Key	Value

Key	Value
5	

Key	Value
CARID	5
MANUFACTURER	Mini
MODEL	One Clubman
ENGINE	1.6 72
AVERAGEFUEL	5.5
BUILDFROM	2010-09-01
BUILDTO	0000-00-00

Figure 4.19: Special responses to special requests of type 'USER'.



Key	Value
STATUS	OK

Key	Value
STATUS	ERROR
ERROR	ACCOUNT_NOT_REGISTERED

Key	Value
STATUS	ERROR
ERROR	ACCOUNT_NOT_ACTIVATED

Key	Value
STATUS	ERROR
ERROR	INVALID_EMAIL_PASSWORD_COMBINATION

Key	Value
STATUS	ERROR
ERROR	INTERNAL_SERVER_ERROR

Figure 4.20: Most common responses.

Chapter 5

Conclusion

As seen in section 2.1.1 and section 2.1.2 there are concepts to support users in fuel-saving driving as well as there are concepts to support users in time-saving driving (see section 2.2.1 and section 2.2.2), but concepts which can support users in fuel- and time-saving driving cannot be found. Therefore, this thesis is a main part in the field of driving fuel- and time-saving.

5.1 Benefits and Drawbacks

The benefits which came up while designing the concept to the framework and implementing it are that the framework can easily be extended, customized, setup and used (more detailed discribed in section 5.1.1).

The drawback which has to be handled is that the concept as well as the framework are not motivating drivers to drive fuel-saving (more detailed described in section 5.1.2).

5.1.1 Benefits

Easily to be extended and customized

As described in section 4.2.3, modules can easily be added by just following a few guidelines. For example by replacing the provided MySQLDatabaseModule another type of database can be used without recompiling or changing parts from the source code of the server.

Easily to be setup and used

Because of Java which is used for implementing, the server can be run at every machine containing a Java runtime environment.

The provided graphical user interface which must not be used makes it possible to configure the server in an easy way without causing any harm which could prevent the server from starting.

5.1.2 Drawback

The only but crucial drawback of the concept is the feedbacks quality which falls an rises by the count of users. Because of the feedback is generated by comparing the users car telemetry (see section 3.1.1) a lot of users are needed to give a good advise. Another point why the quality rises and falls with the user count is, that the users car telemetry is compared to other users cars telemetries with an relatively close fuel consumption average, therefore the less user take part, the probability of users with relatively close fuel consumption averages falls and therefore the less data can be compared.

5.2 Result

Taking the above described benefits and drawbacks into account leads to the result that if there will be a lot of users, especially experts in fuel- and time-saving driving, the concept and its implementation could be a gain to support users in fast and economic driving.

Chapter 6

Future Work

As seen in chapter 3 and chapter 4 the connection of an Android device to the users car diagnose-interface OBD2 was mentioned but not discussed (see section 6.1). Another part which has to be solved in future work is to implement and connect and iPhone client to reach even more smartphone users (see section 6.2). The last but not least step to do in future is to design and evaluate a concept to motivate drivers in fuel-saving driving by using the framework described and discussed in this thesis (see section 6.3).

6.1 Connecting an Android device to the car

Because of it will be a too big topic to be discussed in this thesis the connection of a Android device to the cars OBD2 interface was not shown.

The theoretical way of connecting a Android device to a cars diagnostic interface is at first to find a way in which the Android device can connect to the interface. This can be achieved by using a OBD2-to-Bluetooth adapter (see Figure 6.1).

Figure 6.1: Lescars OBD2-Bluetooth Adapter. Image taken from [33].

6.1.1 Problems gaining car telemetries using OBD2

As seen in Table 3.4 there is no way to get information about the current used gear. One way to solve this problem is by ignoring it because it does not affect the mathematics used to calculate the score of the received car telemetries (see section 3.1.1). The cars current gear could just be an optional value because the user could just use the highest gear possible while driving the returned speed.

6.2 iPhone client

After implementing a Android Client and to reach as many as possible smartphone users, a iOS Client should be implemented. Referring to Apples iOS application development starting guide [48] a mac computer running at least OS X 10.8 (Mountain Lion), XCode [56] and the iOS SDK are needed. Mostly, the iOS SDK is already included in XCode which can be found in Apple's AppStore. To test applications on Apple's smartphone device a real device (where a development certificate is needed [55]) or a iOS Simulator [28] (see Figure 6.2) which is already included in XCode can be used.

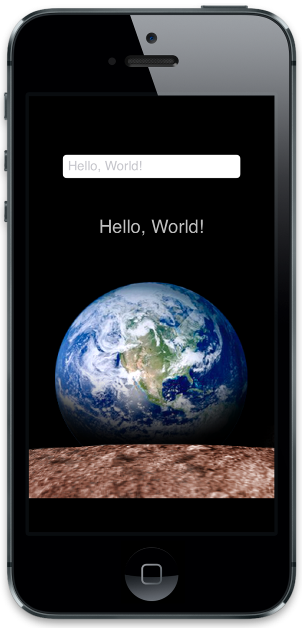


Figure 6.2: iPhone simulator. Image taken from [28].

6.2.1 Connecting an Apple device to the car

To connect a iPhone device to the cars diagnose interface a OBD2-Wifi-Adapter [34] which is shown in Figure 6.3 can be used. Referring to the manufacturers requirements the OBD2-Bluetooth interface [33] will not work.

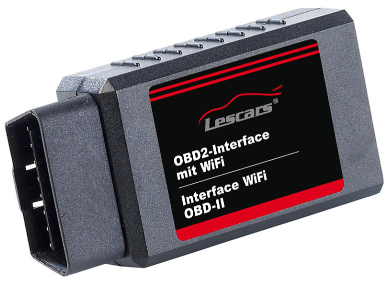


Figure 6.3: Lescars OBD2-Wifi Adapter. Image taken from [34].

6.3 Motivating drivers

A part which is missing in the concept (described in chapter 3) and implementation (described in chapter 4) and which is the main part of GreenR [23] described in section 2.1.2 is motivating the users to drive in a fuel-saving way. To motivate car drivers to drive in a fuel-saving way by using the concept described in this thesis is by setting it up as game where all registered drivers can win prizes by driving as economically and fast as possible. To finance the prizes the implemented clients could be sold or commercials could be shown.

Appendix A

CD-ROM Content

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 PDF-Files

Path: /Thesis

MasterThesisReichart.pdf	This thesis

Path: /Thesis/Images

*.pdf	Images used in this thesis
FuelTimeFramework.pdf	FuelTimeFramework logo (same as icon in the Server GUI and Android launcer icon)

Path: /CD

MasterThesisCDLabel.pdf	CD-Label
MasterThesisCDCover.pdf	CD-Cover

A.2 Source Code

Path: /SourceCode

workspace	Eclipse workspace of the FuelTimeFramework server, the server GUI and the Android client

References

Literature

[2] VíctorCorcoba Magaña and MarioMuñoz Organero. “AndroWI: Collaborative System for Fuel Saving Using Android Mobile Devices”. In: *Ambient Intelligence - Software and Applications*. Ed. by Ad Berlo et al. Vol. 219. Advances in Intelligent Systems and Computing. Springer International Publishing, 2013, pp. 49–55. URL: http://dx.doi.org/10.1007/978-3-319-00566-9_7 (cit. on p. 3).

[3] D. Medhi. *Network Routing: Algorithms, Protocols, and Architectures*. The Morgan Kaufmann Series in Networking. Elsevier Science, 2010 (cit. on p. 11).

[5] L. Steinke. *Spieleprogrammierung*. Das bhv-Taschenbuch. bhv, 2007 (cit. on p. 11).

[6] S. Wasserbauer. *Eco-Feedback im Automobil: Eine Analyse kontemporärer Eco-Feedbacks in Kraftfahrzeugen und Entwicklung eines Eco-Feedback Simulators*. AV Akademikerverlag, 2012. URL: http://books.google.de/books?id=1xMrLgEACAAJ (cit. on p. 4).

Online sources

[7] *61,5 Millionen: So viele Fahrzeuge wie nie in Deutschland*. URL: http://www.rp-online.de/leben/auto/news/615-millionen-so-viele-fahrzeuge-wie-nie-in-deutschland-aid-1.4090173 (cit. on p. 9).

[9] *ADAC - Autodatenbank*. URL: http://www.adac.de/infotestrat/autodatenbank/default.aspx (cit. on p. 10).

[10] *Allgemeines - OBD-2.net - Das Fahrzeugdiagnose Informationsportal*. URL: http://www.obd-2.de/obd-2-allgemeine-infos.html (cit. on p. 18).

[11] *Android Architecture – The Key Concepts of Android OS*. URL: http://www.android-app-market.com/android-architecture.html (cit. on pp. 23, 25).

[12] *Android Cipher encrypt/decrypt - Stack Overflow.* URL: http://stackoverflow.com/questions/14022934/android-cipher-encrypt-decrypt (cit. on pp. 37, 46).

[13] *Android-x86 - Porting Android to x86.* URL: http://www.android-x86.org/ (cit. on pp. 24, 26).

[14] *Der AUTO BILD Verbrauchs-Test - autobild.de.* URL: http://www.autobild.de/artikel/der-auto-bild-verbrauchs-test-55631.html (cit. on p. 2).

[15] *Eclipse - The Eclipse Foundation open source community website.* URL: https://www.eclipse.org/ (cit. on p. 24).

[16] *Frequently Asked Questions (FAQ) About On-Board Diagnostic II (OBD II) Systems.* URL: http://www.arb.ca.gov/msprog/obdprog/obdfaq.htm (cit. on pp. 3, 17).

[18] *Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013.* URL: http://www.gartner.com/newsroom/id/2623415 (cit. on p. 24).

[20] *„Google Maps“ für iPhone, iPod touch und iPad im App Store von iTunes.* URL: https://itunes.apple.com/de/app/google-maps/id585027354?mt=8 (cit. on p. 5).

[21] *GraphHopper Road Routing in Java with OpenStreetMaps.* URL: http://graphhopper.com (cit. on pp. 35, 36).

[23] *GreenR - Economic Car Driving | Wassx's Blog.* URL: http://wassx.wordpress.com/green-street (cit. on pp. 4, 60).

[24] *GreenR - Economic Car Driving | Youtube.* URL: http://www.youtube.com/watch?v=3NoFM9uIs4A (cit. on p. 4).

[26] *Installing the Eclipse Plugin | Android Developers.* URL: http://developer.android.com/sdk/installing/installing-adt.html (cit. on p. 24).

[28] *iOS Simulator User Guide: Getting Started in iOS Simulator.* URL: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/GettingStartedwithiOSStimulator/GettingStartedwithiOSStimulator.html (cit. on p. 59).

[29] *JavaMail API Reference Implementation: Wiki: Home — Project Kenai.* URL: https://java.net/projects/javamail/pages/Home (cit. on p. 38).

[30] *JSON.* URL: http://www.json.org/java/index.html (cit. on p. 32).

[31] *Kraftstoff-Durchschnittspreise.* URL: http://www.adac.de/infotestrat/tanken-kraftstoffe-und-antrieb/kraftstoffpreise/kraftstoff-durchschnittspreise/default.aspx?ComponentId=51587&SourcePageId=185107 (cit. on p. 1).

[32] *Kraftstoffpreise in Österreich.* URL: http://www.oeamtc.at/media.php?id=,,,,ZmlsZW5hbWU9ZG93bmxvYWQlM0QlMkYyMDExLjAxLjI3JTJGMTI5N%20jEzNjU1Ny5wZGYmcm49S3JhZnRzdG9mZnByZWlzZSUyMGluJTIwJUQ%202c3RlcnJlaWNo (cit. on p. 1).

[33] *Lescars OBD2-Profi-Adapter mit Bluetooth-Übertragung für Androidgeräte.* URL: http://www.pearl.de/a-NX3014-1523.shtml (cit. on pp. 58, 60).

[34] *Lescars OBD2-Profi-Adapter mit WiFi für iPhone/iPad.* URL: http://www.pearl.de/a-NX3027-1523.shtml (cit. on p. 60).

[35] *Maps - Android-Apps auf Google Play.* URL: https://play.google.com/store/apps/details?id=com.google.android.apps.maps (cit. on p. 5).

[37] *MySQL :: Download Connector/J.* URL: http://dev.mysql.com/downloads/connector/j/3.1.html (cit. on p. 37).

[38] *OBD II PIDs | Hangas.* URL: http://hangas.com/?p=128 (cit. on p. 19).

[39] *OBD on CAN - emotive GmbH & Co. KG.* URL: https://www.emotive.de/doc/car-diagnostic-systems/protocols/dp/obd-on-can (cit. on pp. 3, 17, 18).

[40] *OBD-II PIDs » OBD-II Resource.* URL: http://obdcon.sourceforge.net/2010/06/obd-ii-pids/ (cit. on p. 19).

[41] *OpenStreetMap.* URL: http://www.openstreetmap.org (cit. on pp. 6, 50).

[43] *Osmand+.* URL: https://play.google.com/store/apps/details?id=net.osmand.plus (cit. on p. 6).

[44] *osmdroid - OpenStreetMap-Tools for Android - Google Project Hosting.* URL: https://code.google.com/p/osmdroid (cit. on p. 50).

[45] *Overpass API.* URL: http://overpass-api.de (cit. on p. 35).

[46] *Programmierer Tips - OBD-2.net - Das Fahrzeugdiagnose Informationsportal.* URL: http://www.obd-2.de/programmierer-tips.html (cit. on p. 19).

[47] *Setting Up the ADT Bundle | Android Developers.* URL: http://developer.android.com/sdk/installing/bundle.html (cit. on p. 24).

[48] *Start Developing iOS Apps Today: Setup.* URL: https://developer.apple.com/library/iOS/referencelibrary/GettingStarted/RoadMapiOS/index.html (cit. on p. 59).

[50] *STATISTIK AUSTRIA - Kraftfahrzeuge - Bestand.* URL: http://www.statistik-austria.at/web_de/statistiken/verkehr/strasse/kraftfahrzeuge_-_bestand/index.html (cit. on p. 9).

[51] *Stephan Brähler: Analysis of the Android Architecture.* URL: http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf (cit. on p. 23).

[54] *User Interface | Android Developers.* URL: http://developer.android.com/guide/topics/ui/index.html (cit. on p. 27).

[55] *Xcode Overview: Run Your App.* URL: https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/RunYourApp/RunYourApp.html (cit. on p. 59).

[56] *Xcode - What's New - Apple Developer.* URL: https://developer.apple.com/xcode/ (cit. on p. 59).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —

Breite = 100 mm
Höhe = 50 mm

— Diese Seite nach dem Druck entfernen! —