

Technical Documentation

Project: Green Energy Shifting
Name: Mahnaz Hajibaba, Sarvenaz Golchin, Veronika Henk
Date: 01.10.2014
Version: 1.5

Table of contents

1. Introduction.....	3
1.1. Technical Details.....	3
2. Software Architecture	4
2.1. System Overview	4
2.2. Structure of the System.....	6
2.3. Functionalities of the System	7
2.3.1. Signing Up.....	7
2.3.2. Logging In.....	8
2.3.3. Obtaining Green Energy Information	9
2.3.4. Updating the Twitter Status	9
2.3.5. Setting a Notification.....	10
3. Data Structure	12
3.1. User Data.....	12
3.1.1. Table <i>ge_login</i>	12
3.2. Green Energy Data	13
3.2.1. Data Acquisition	13
3.2.2. Table <i>energydata</i>	14
3.3. Green Energy Prediction Data	14
3.3.1. Table <i>predictiondata</i>	15
4. Web Services (PHP files).....	16
4.1. File <i>signup.php</i>	16
4.2. File <i>login.php</i>	16
4.3. File <i>getscore.php</i>	17
4.4. File <i>setscore.php</i>	17
4.5. File <i>getenergynow.php</i>	17
4.6. File <i>getpredictions.php</i>	18
5. Testing	20
5.1. Unit Testing	21
5.2. Integration Testing	22
6. Sources	24

1. Introduction

The intention of this technical documentation is to give an overview and an insight on how the system operates. Moreover the various aspects of the developed application and its connection to external sources are described. For facilitating the comprehension of this system depicting the software architecture has become an essential part of this document.

Hereafter, the structure of this document is pointed out. The next passage of this document contains a list of technical details regarding the Android application as well as tools which were used within the process of development.

The following section "Software Architecture" outlines the structure of the whole development and illustrates several functional sequences in detail. As the Android application itself is the core of this development the main focus relating to software architecture is on the classes placed in the application.

The section "Data Structure" displays which kind of data is used in the development. It also presents research results and which were achieved in the process of data acquisition as well as the data's source. Moreover, this part explains which parts of the application are connected to the data structure. The data is stored in a database on the web server. The corresponding table to every class of data and its structure is outlined as well.

In the section "Web Services" the web services which are PHP files located on the web server are linked to the developed system. Also the purpose of every PHP file is mentioned. Furthermore, a description of each web service is provided. The description consists of the service's input parameters, the processing it accomplishes, the tables which are accessed, as well as the result which is delivered by the PHP file. If alternatives which appear useful for the system's further development and which are related to web services were identified, they are also mentioned in this part of the document.

The last section of this document is "Testing". This part of the technical documentation outlines which aspects of software testing had to be taken into account for this project. Moreover, this passage enables to reconstruct our proceeding when testing the Android application.

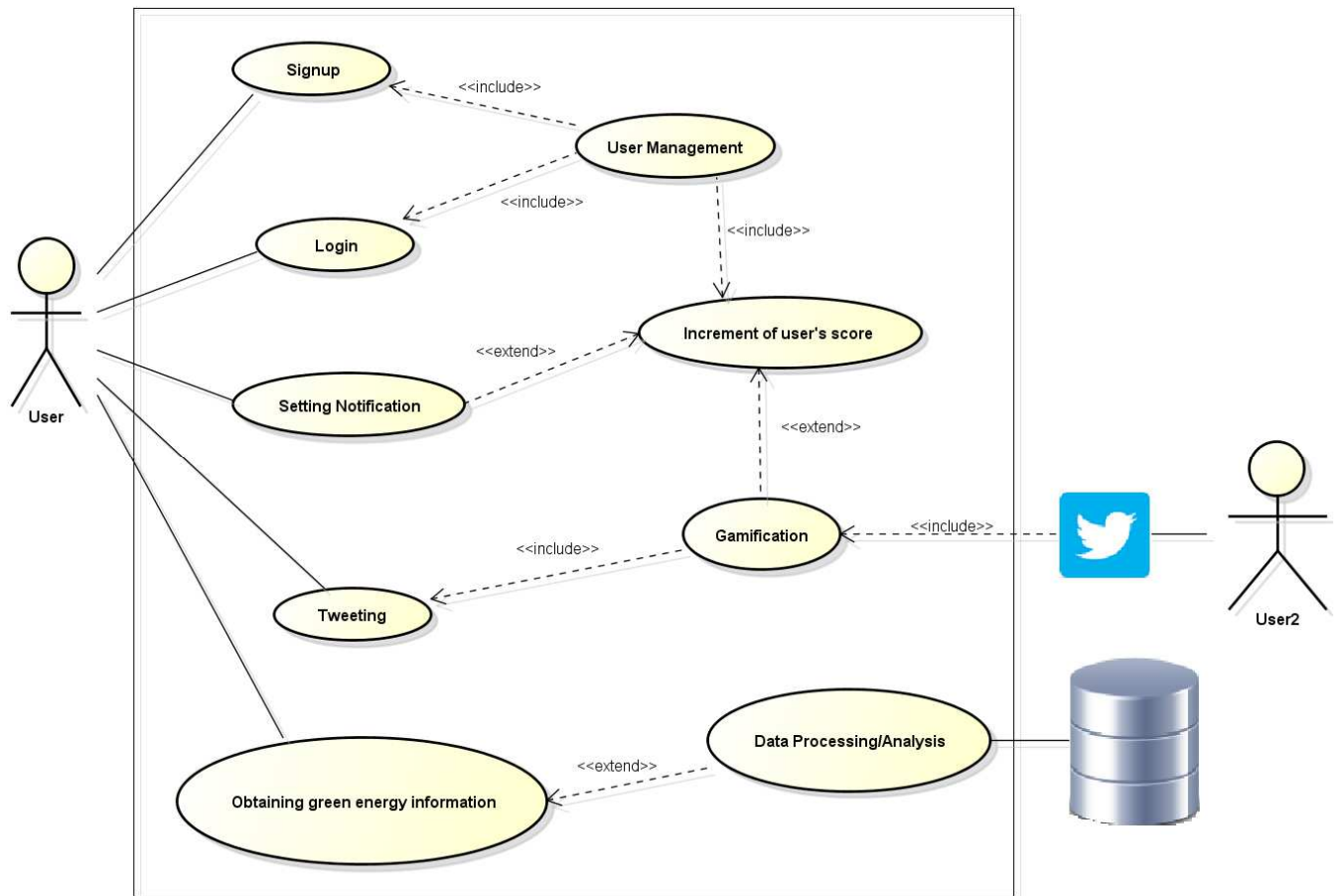
Describing and explaining each and every class of the Android application in this document would get out of hand and destroy the clearness of the technical documentation. Therefore, the JAVA-code created for the purposes of this project was annotated and documented using *Javadoc*. Hence details about the each and every class and package of the Android application can be found in the (archive-) folder "Green Energy Javadoc" and are extending this technical documentation. As mentioned above the web services which are available as PHP files are elucidated in this document. A more detailed impression on those files can be gained by regarding the PHP code itself and its comments.

1.1. Technical Details

- Developed for the Android versions 2.3 - 4.4.4
- Tested for the Android versions 2.3.4, 4.0.4, 4.3, and 4.4.4
- PHP files run on a web server with PHP version 5.2.17
- MySQL version of the external database is 5.6.19

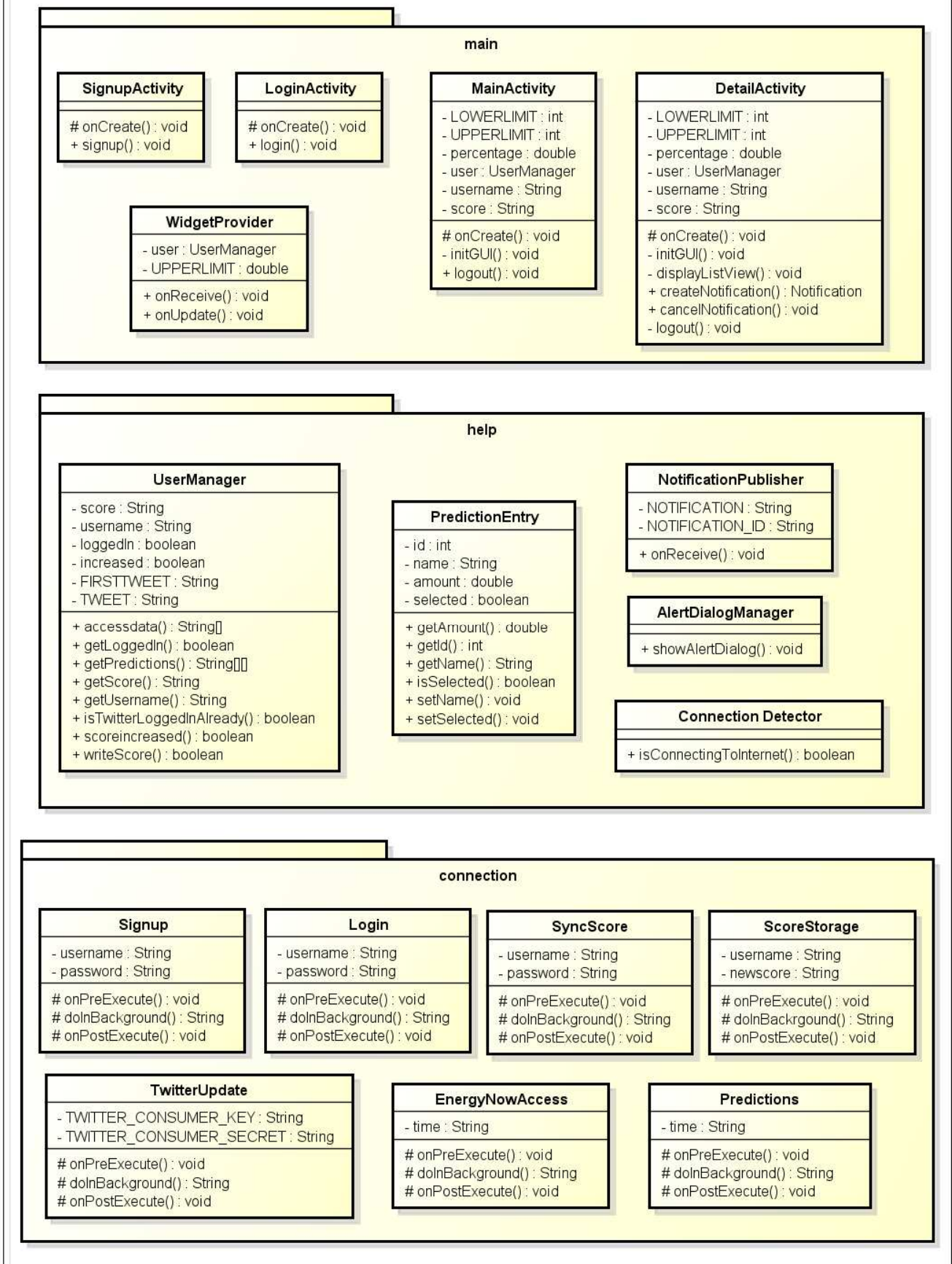
2. Software Architecture

2.1. System Overview



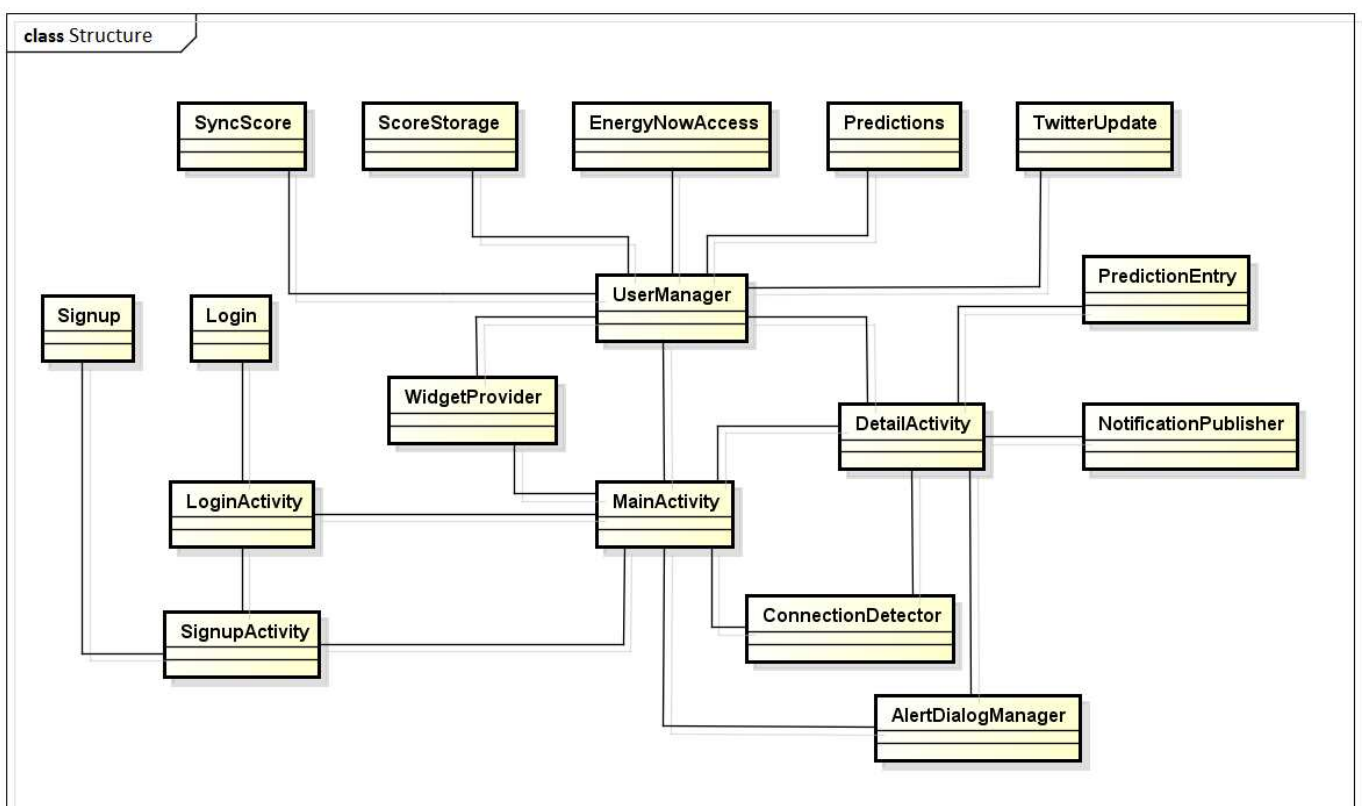
The use case diagram above illustrates the main functionalities of the developed system and their connections to system internal components as well as to components outside of the system itself. Here the system representation takes place on another level as the system's representation in the use case diagram contained in the requirements specification. Using the Android application the user can utilize the use cases *Signup*, *Login*, *Setting Notification*, *Tweeting*, and *Obtaining green energy information*. The use case *User Management* is an essential component inside the system and is composed of the use cases *Signup*, *Login*, and *Increment of user's score*. The use case *Gamification* depends on the use case *Tweeting* and is also connected to the social network *Twitter* which is not a component of this system but can be accessed by this or other users. The use cases *Setting Notification* and *Gamification* can also contain the behavior of the use case *Increment of user's score*, but as this is not always the case they do have an extend-relation. Furthermore the behavior of the use case *Obtaining green energy information* can be obtained by the use case *Data Processing/Analysis* which is responsible for accessing the external database.

The package diagram on the next page provides an overview of the Android application's packages and the classes they contain. The most important attributes and methods of the individual classes are also illustrated. To obtain the clarity of this diagram the relations between each and every class as well as the relations between the packages are omitted. See the class diagram in section 2.2 to gain information about the connection between the classes.



As can be seen in the diagram on the previous page, the Android application's classes are divided into three packages: *main*, *connection*, and *help*. The system was structured based on the Model View Controller-architecture pattern. Consequently, the package *connection* contains solely *Async*-task classes which are responsible for establishing a connection to the web server and retrieving data from the external database. Therefore, those classes are part of the application's model. For security reasons it is no longer possible to access external networks from the main thread of an Android application. Hence, those connections need to be established by the *Async*-tasks using subthreads. These classes send requests to the PHP files on the web server. Those files are web services which can access the external database and return the retrieved data back to the *Async*-tasks. The view component of this system is provided by the layout files stored in the *res*-folder of the Android application. Those files contain the layout of the development defined in XML. The classes stored in the package *main* represent the different views of the application which the user of the mobile app sees. Although they display the layout, they belong to the application's controller as they manage the most of the application's behavior. The package *help* contains auxiliary classes which are also part of the application's controller as they support the classes stored in the package *main*. Especially the class *UserManager* is of crucial importance for this system as it manages almost the whole data flow of the application and controls most of the application's structure. The individual classes of the Android application are explicated in the *Javadoc* documentation.

2.2. Structure of the System



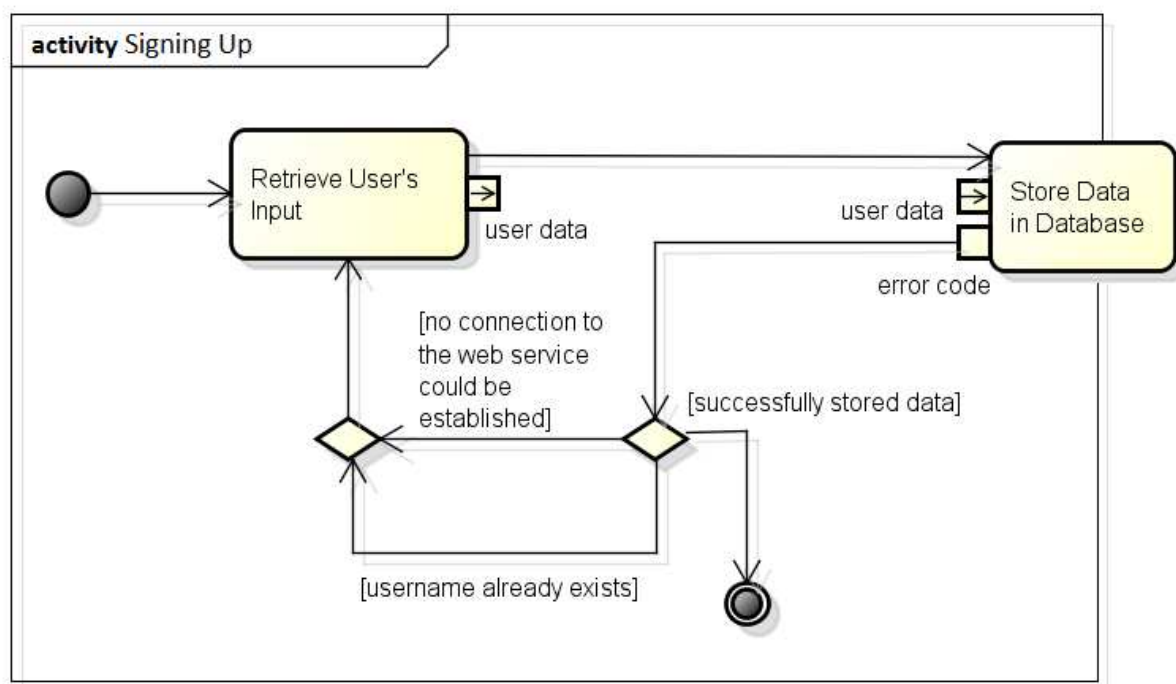
powered by Astah

The simplified class diagram above shows the classes of the Android application and how they are connected to each other. The central component in this diagram is the class *UserManager* as it con-

trols nearly the whole data flow. As described in section 2.1. the class *UserManager* is assigned to the system's controller part. An instance of this class is created and its methods are accessed by classes which represent one of the application's views. Therefore this class is connected to the activities *MainActivity* and *DetailActivity* as well as to the class *WidgetProvider*. As this class is crucial for the application's data view, it also initiates most of the database accesses by executing `Async`-tasks. Consequently, this class is also connected to the classes *SyncScore*, *ScoreStorage*, *EnergyNowAccess*, *Predictions*, and *TwitterUpdate*, which are extending the class `Android.os.AsyncTask`. Another key component of this diagram is the class *MainActivity*. This activity represents the main view of the application. As this is the launcher-activity of the project, it is connected to all the application's other views, which are represented by the classes *LoginActivity*, *SignupActivity*, *DetailActivity*, and *WidgetProvider*. When the application is launched and it is detected that no user is logged in, *MainActivity* forwards to the *LoginActivity*. This class represents the log in-view of the application and is connected to the `Async`-task *Login* as well as to the class *SignupActivity*. The *SignupActivity* represents the application's sign up-view and is connected to the `Async`-task *Signup*. The class *DetailActivity* which represents the detail view of the application is connected to the auxiliary classes *PredictionEntry* and *NotificationPublisher*. An instance of the class *PredictionEntry* represents an entry in the list displayed on the detail view containing green energy predictions for the next ten hour. The class *NotificationPublisher* enables the pushing of a notification at a certain point of time in the future. Furthermore, the classes *MainActivity* and *DetailActivity* are both connected to the auxiliary classes *ConnectionDetector* and *AlertDialogManager* which render the verification of a working internet connection and the display of an alert dialog if no internet connection could be detected. The classes and their components are described in greater detail in the *Javadoc* documentation.

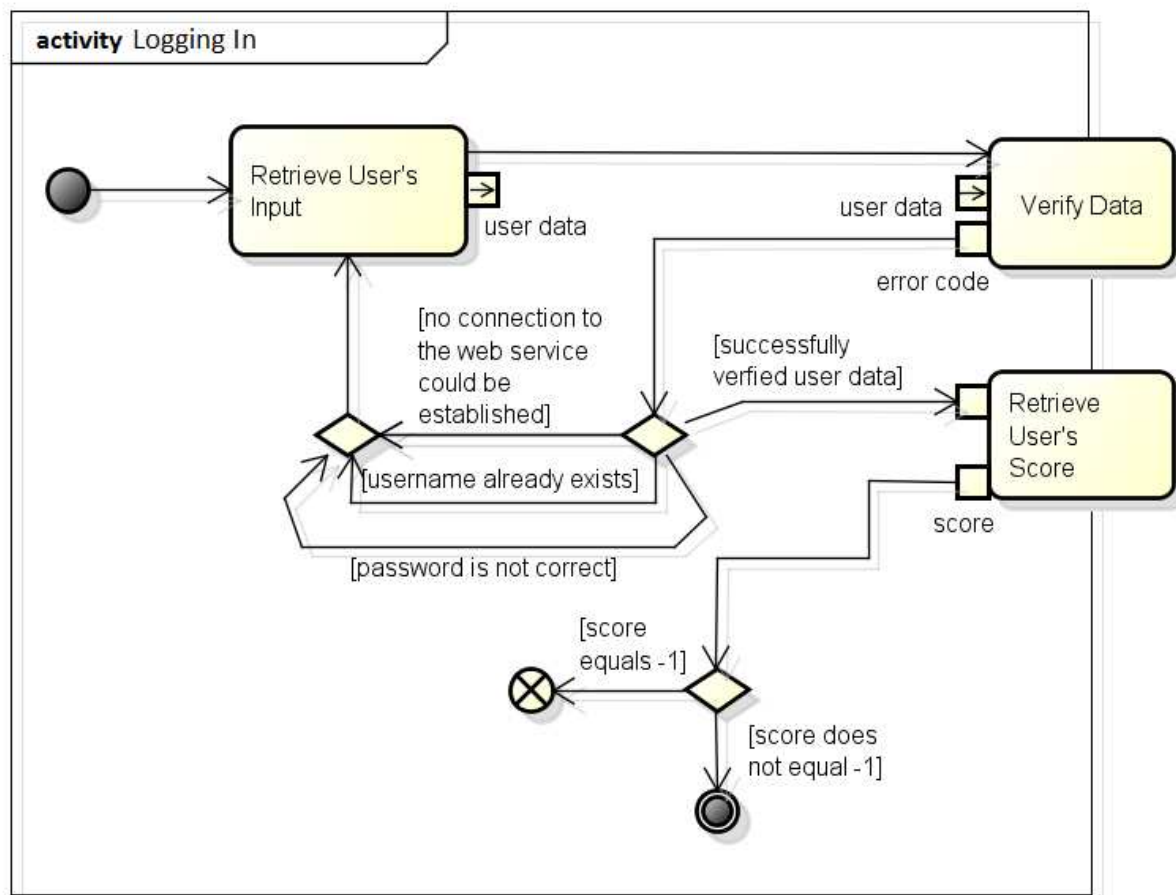
2.3. Functionalities of the System

2.3.1. Signing Up



The activity diagram displayed on the previous page represents the process of signing up a new user. The first action of this activity is to retrieve the data which was inserted by the user in the sign up - view of the Android application. Subsequently, the user data is stored in the external database by executing an `Async-task` within the application. If the returned error code equals 0, storing the data was successful and the activity is terminated. Otherwise, if the error code does not equal 0 the user-name does either already exist in the database or no connection to the web service could be established. Both of these cases lead back to the application's sign up-view where the user needs to enter his data again or at least needs to click the "Sign Up "-button again.

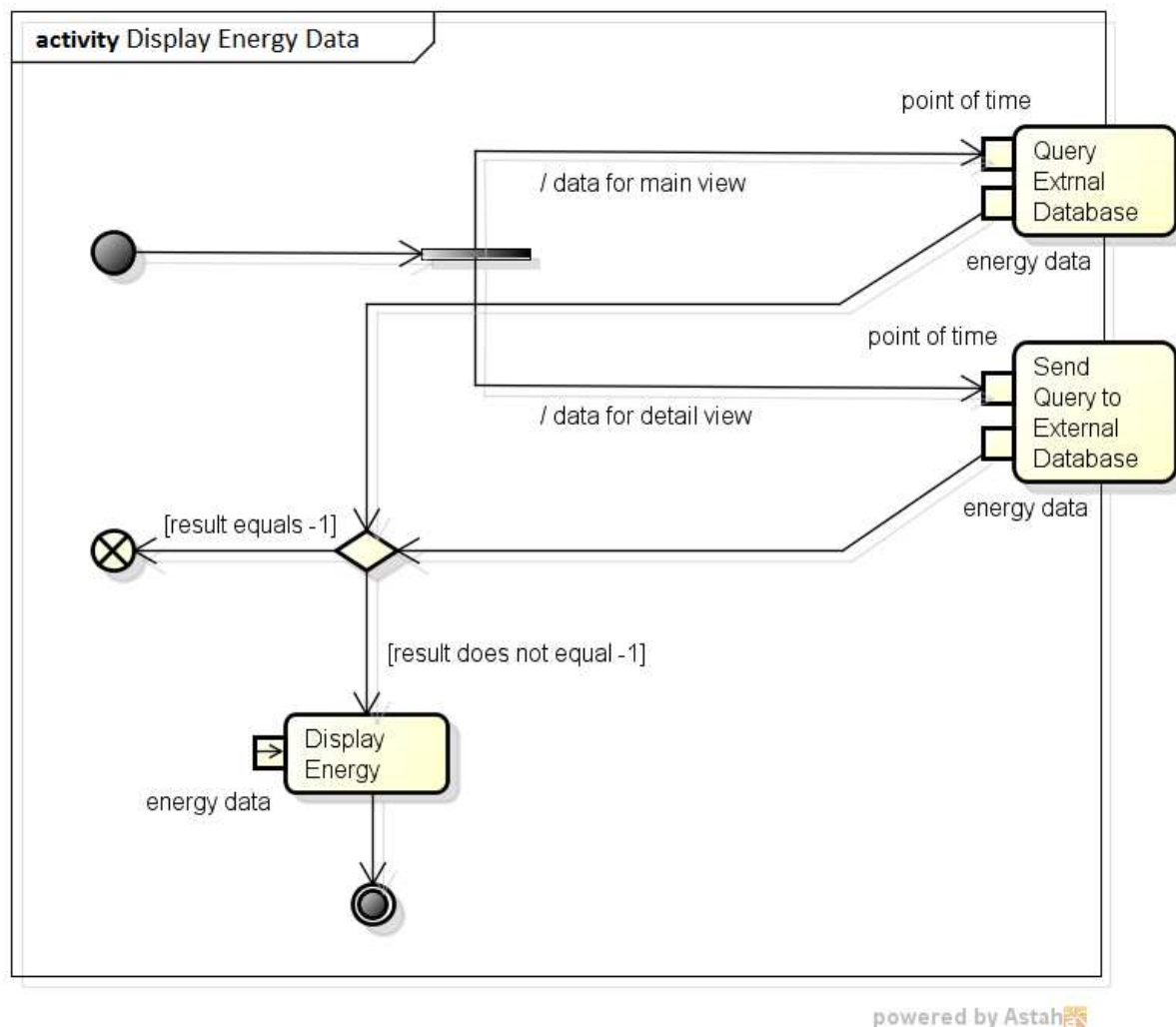
2.3.2. Logging In



powered by Astah

The activity diagram displayed above represents the process of logging the user into the Android application. The first action of this activity is to retrieve the data which was inserted by the user in the log in-view of the application. Next, the user data is verified by comparing it to data stored in the external database by executing an `Async-task` within the application. If the returned error code equals 0, the user data was verified successfully. Otherwise, if the error code does not equals 0 an error occurred. This case leads back to the application's log in-view where the user needs to enter his data again or at least needs to click the "Log In"-button again. If the user data was verified successfully, the user's score is retrieved by executing another `Async-task` within the application. If the returned result does not equals `-1` the score was retrieved successfully and the activity is terminated. Otherwise, if the result equals `-1` a crucial error occurred and the activity is aborted.

2.3.3. Obtaining Green Energy Information

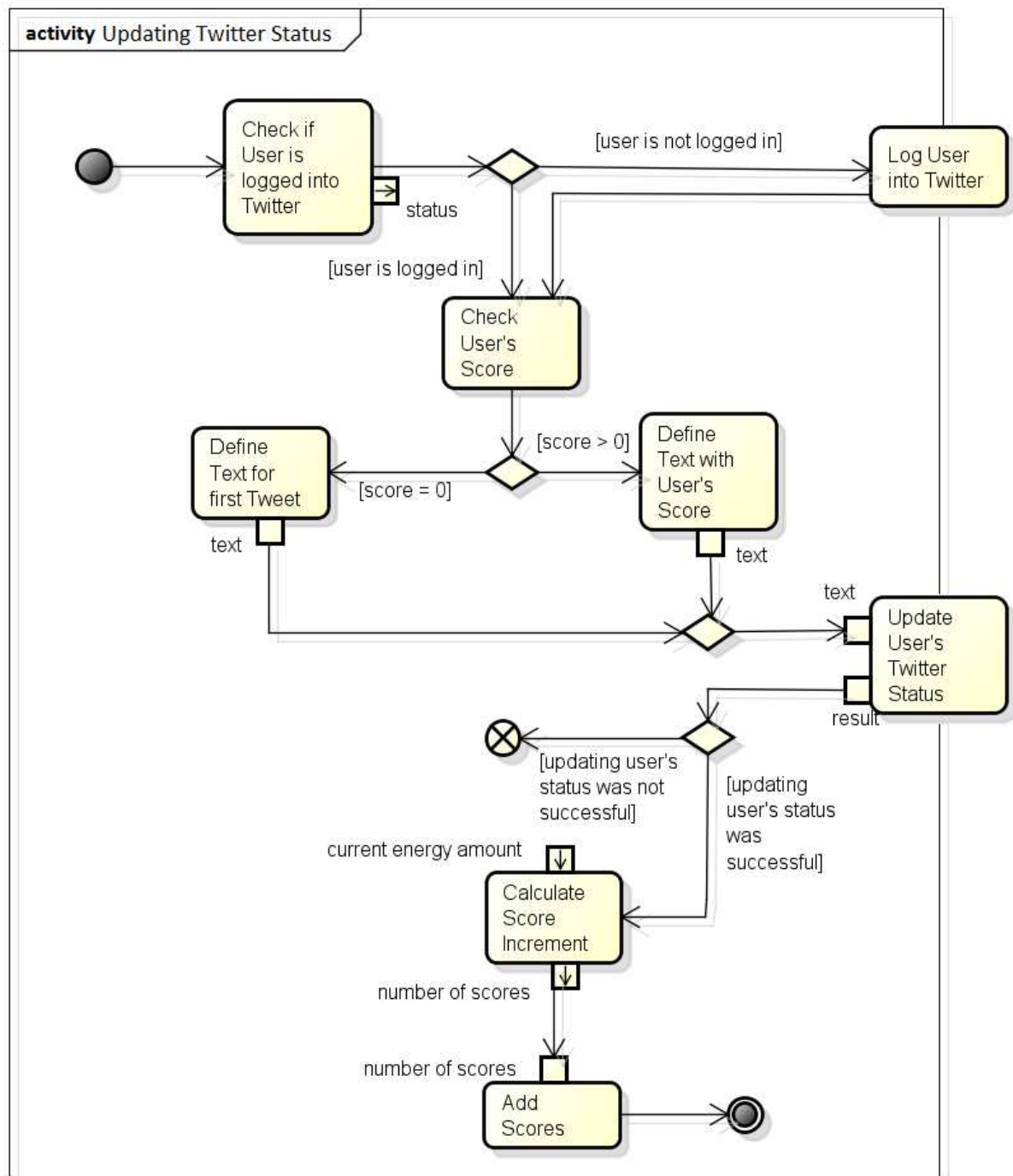


Above, the activity diagram is shown which represents the process of obtaining green energy information and displaying it to the user. The first action of this activity is depending on whether data for the application's main view or for the application's detail view should be obtained. Corresponding to this choice the related `Async`-task is executed within the Android application in order to obtain the energy data. If the result returned by the `Async`-task equals `-1` a crucial error occurred and the activity needs to be aborted. If the result does not equal `-1` it needs to be processed and the energy data is displayed to the user. Hereafter, the activity is terminated.

2.3.4. Updating the Twitter Status

The activity diagram shown on the following page represents the process of updating the user's Twitter status. The first action of this activity is to check if the user is already logged into Twitter. If he is not already logged in, this is accomplished by executing the `Async`-task `TwitterUpdate` within the Android application. After that or if the user was already logged in anyways, the user's score is checked. Depending on whether the user already gained score using the application a text for the status update is generated. Hereafter the user's Twitter status is updated. If the user's status was not updated successfully, the activity is terminated. If the status was updated, the score gained by the user for tweeting is calculated. The score is dependent on the current amount of green energy. If this

amount is above the defined upper limit, the user's score increases by 5. If the amount is between the upper and the lower limit, the user's score increases by 3. Otherwise, if the amount is even less than the lower limit, the user gets only 1 score. After the new score was added, the activity is terminated.

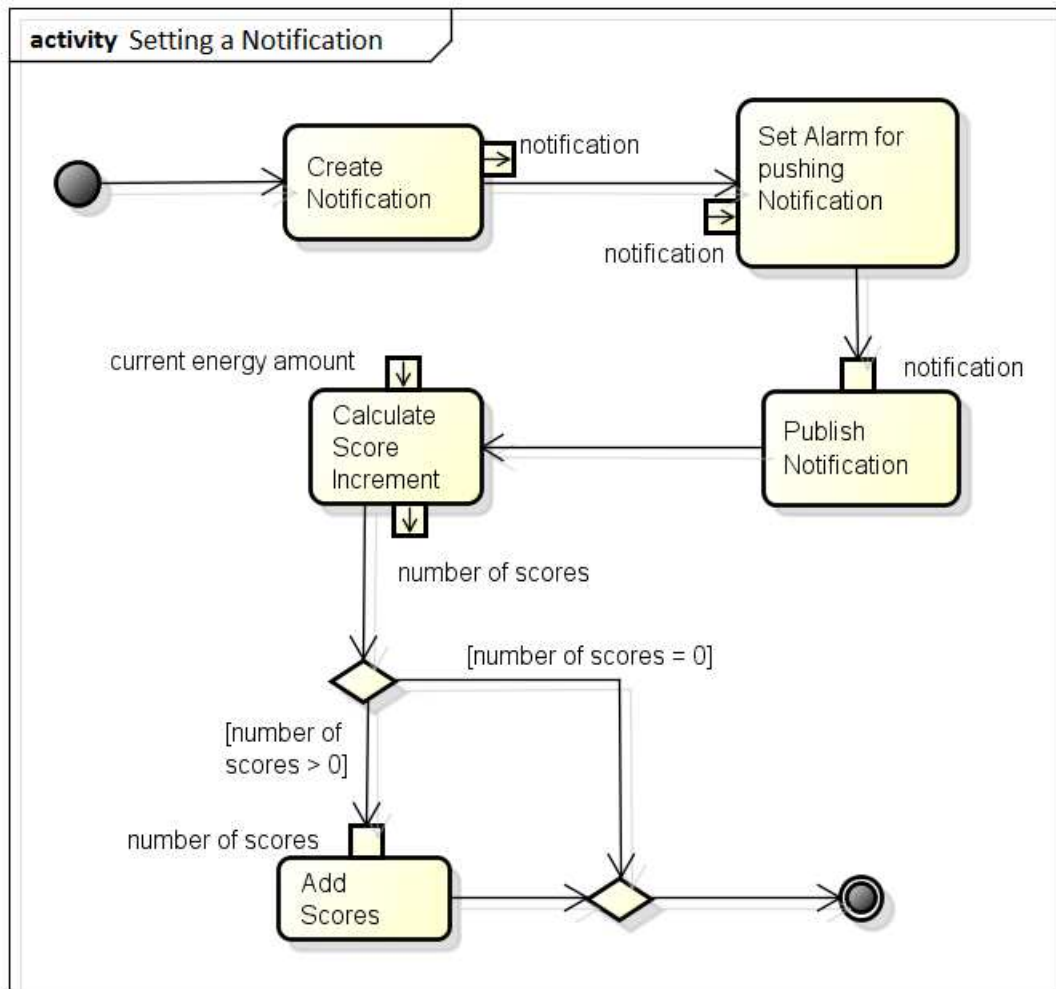


powered by Astah

2.3.5. Setting a Notification

The activity diagram on the next page represents the process of setting a notification which the user should receive at a certain time of point in the future. the first action of this activity is to create the notification itself. Hereafter the alarm needs to be set in order to display the notification not right away but at a point of time defined by the user. When this instant of time arrives, the notification is

published to the user and the main part of this activity is accomplished. Subsequently, the score gained by the user for setting the notification is calculated. The score is dependent on the predicted amount of green energy and related to the instant of time when the notification is pushed. If the predicted amount of green energy is above the specified upper limit, the user's score is increased by 5. Otherwise, if the amount is between the upper and the lower limit, the user's score is increased by 3. If the amount is even less than the lower limit, the user's score is not incremented at all. In case the user gets new scores this activity is terminated after the score was added. If the predicted energy amount is too low, the activity was already terminated when calculated score increment was evaluated.



powered by Astah

3. Data Structure

The data is stored in an external database. This database is just as well as the web services currently located on a private web space. Initially the plan was to store the data and the PHP scripts on a machine in the EIS lab. The hindrance which was encountered in this regard is that no server or computer of the university can be accessed from outside of the university network. It was assessed that connecting with a virtual machine from within an Android application is not possible, hence the data needs to stay located on the private web space. This is a drawback as for a subsequent development by other persons the data has to be moved to another location.

3.1. User Data

The user data in this system consists of the user's nickname, password, and the score he received through using the application. The user data is stored in the external database to assure that the score can be synchronized automatically when the Android application is used on various mobile devices. Hence the user data needs to be on a server which is accessible from within the application. To assure that a user can only access his own score the creation of user accounts is unavoidable. As the user should not be discouraged to sign up for this application and because we do not want to collect more private user data than necessary the data is narrowed to a nickname and a password chosen by the user. The users' passwords are encrypted using the unidirectional secure hash algorithm (SHA) to assure that they are not compromised if the database is accessed by an unauthorized person.

3.1.1. Table *ge_login*

The login data is stored in the external database's table *ge_login*. The "ge" in this table's name is an abbreviation for "green energy" and was assigned because a table named "login" already exists in the same database. Every dataset in this table has a unique ID, which was generated automatically. The *ID*-column is the primary key of the Table *ge_login*. Furthermore, every dataset contains the name of the corresponding user in the column *username*. As *username* can have a maximum length of 30 characters. The values stored in the column *username* have to be unique in this table. Within the web services, which are part of this development, it is also secured programmatically that no dataset with an already existing is inserted in this table. The column *password* contains the user's password encrypted using the secure hash algorithm. Hence every record in this column has an exact length of 40 characters. The amount of scores the user achieved is stored in the table's column *score*. As the score is always a numerical value the data type for this column is `INTEGER`. The default value in the column *score* is 0, because when a new dataset representing a new user is inserted, the user is not supposed to have a higher score already. No column in the table *ge_login* allows `NULL` values, as the record of every column is essential for using the application.

ID
username
password
score

3.2. Green Energy Data

Green energy data refers to information representing the current amount of green energy corresponding to a certain point in time.

3.2.1. Data Acquisition

As the application should help to shift energy consumption to the time slots when the amount of green energy is higher, data about the amount of green energy in relation to the amount of conventional energies is necessary. Therefore it would be ideal to access an existing data source with the current renewable energy amount. For example, such a source could be a database where the information is updated within hours or at least several times per day. Unfortunately no current data source of the form described above could be retrieved. We could not find the data we needed provided via an application programming interface which is free accessible. The alternative to a freely accessible data source was to ask companies which are reliable to provide such data. An email was sent to the companies "Stadtwerke Bonn Energie und Wasser" and "RheinEnergie". Unfortunately neither of them showed any consideration for the request. As we did not want to risk losing too much time needed for the development of the system with searching for data providers, the usage of test data in this project was decided.

The data containing information about the amount of green energy which is used as test data was retrieved from the "Transparency Data Interface Specification" by "EEX Information Products". The data containing information about the actual energy amount in mega watt is real data (Ex-Post) quantified hourly on May 15th 2012. It refers to energy in Germany provided by "Amprion GmbH". This company is an important transmission system operator in Europe. Furthermore, it operates a German extra-high voltage grid with a length of 11,000 km.


To specify an upper and a lower limit for the evaluation of the green energy amount within the Android application in order to assign a score to the user, further research needed to be accomplished. As the acquisition of up-to-date energy data was difficult, researching on the average percentage rate of green energy in Germany for the different seasons of the year was onerous likewise. The amount of green energy per season of the year could not be identified, but the amount of green energy per year or semi year could be detected and are visualized in the table below.

<i>Year 2011</i>	20.2 %
<i>Year 2012</i>	22.8 %
<i>First half year 2013</i>	24.5 %
<i>Year 2013</i>	23.4 %
<i>First half year 2014</i>	28.5 %

Depending on the average green energy amount of the values in the table above and on the prognosis that the percentage rate of green energy amount in Germany will increase further, the lower limit has been specified as 27%. Furthermore, the German government scheduled that the percentage rate of green energy should climb to 40-45% until the year 2020 and to 55-60% until the year 2035. Therefore the upper limit was intentionally fixed to the relatively high rate of 60%.

3.2.2. Table *energydata*

The data which represents the current amount of green energy is stored in the external database's table *energydata*. The table contains the column *ID*, which contains the primary key of each record. Hence the values in this column are unique in the table and allow to unambiguously identify each dataset. Moreover, every data set of this table stores a time of day in the column *timehour* to indicate to which interval of time the energy amount corresponds. The records in the column *timehour* do have the data type `time`. Currently the values in this column represent each hour of the day and represent the energy amount for this point in time until one hour later. For example, this column's value in the first dataset is '00:00:00'. Hence the information given by this dataset represents the energy amount from midnight until 00:59 a.m. The next dataset's value for this column is '01:00:00', hence it is valid from 01:00 a.m. until 01:59 a.m. As the test data does only comprises information about the energy amount for one day, there is currently no column for the date to which the information applies.

energydata	
 ID	
timehour	
solar	
wind	
conventional	
total	
percentage	

The columns *solar*, *wind*, and *conventional* contain the amount of the corresponding energy type in mega watt and relate to the time defined by the column *timehour*. Those columns do have the data type `DOUBLE`. In contrast to the other columns introduced so far, the values in these columns do not have to be unique in the table *energydata*. The sum of the values recorded in the columns *solar*, *wind*, and *conventional* is stored in the column *total*, which also has the data type `DOUBLE`. The green energy amount referred to in this project is composed of the amount of solar and wind energy stored in this table. The amount of this green energy in relation to conventional energies, which are stored in the column *conventional* is stored as percentage rate with data type `DOUBLE` in the column *percentage*. Although the data stored in the column *percentage* is the only energy data of this table which is actually accessed from the application, the other columns are likely to get useful for a further development of the application. For example an extended version of this system could contain more detailed visualizations of the green energy amount by displaying the different kinds of renewable energies and their amounts in a chart. Those reflections moved us to keep the data in the database despite it is not needed in the current Android application. No column in the table *energydata* allows `NULL` values.

3.3. Green Energy Prediction Data

Green energy prediction data refers to information representing the amount of green energy predicted for a certain interval in the near future. The data containing such information which is used as test data for this development was retrieved from the "Transparency Data Interface Specification" by "EEX Information Products". The data containing information about the predicted energy amount in mega watt is real data (Ex-Ante) which was actually predicted on May 15th 2012 in an hourly interval. It refers to energy in Germany provided by "Amprion GmbH". As mentioned in section 3.2.1. this company is an important transmission system operator and operates a large German extra-high voltage grid.

3.3.1. Table *predictiondata*

The data which contains the predicted amount of green energy is stored in the external database's table *predictiondata*. The primary key of each record in this table is stored in the column *ID*. Therefore the values in this column are unique in the table and enable the ambiguous identification of each dataset. Similarly to the table *energydata* this table also stores a time of day in the column *timehour* to indicate to which interval of time the predicted energy amount corresponds. the records in the column *timehour* do have the data type `time`. Currently the values in this column represent 24 intervals within a day and the predicted energy amount for this point in time until one hour later. For example, this column's value in the first data set is '23 : 45 : 00'. Hence the information given by this dataset represents the predicted energy amount from a quarter before midnight until 00:44 a.m. on the next day. The next dataset's value for this column is '00 : 45 : 00', hence it is valid from 00:45 a.m. until 01:45 a.m. As the test data does only comprise information about the predicted energy amount for one day, there is currently no column for the date to which the information applies.

predictiondata

ID
timehour
solar
wind
conventional
total
percentage

The predicted amount of the corresponding energy type in mega watt is stored in the columns *solar*, *wind*, and *conventional*. just as in the table *energydata*, the amount of energy in the table *predictiondata* is stored as `DOUBLE` and relates to the time defined by the column *timehour*. The sum of the values recorded in the columns *solar*, *wind*, and *conventional* is stored in the column *total*, which also has the data type `DOUBLE`. In the Android application we refer to the predicted green energy amount consisting of the amount of solar and wind energy stored in this table. This predicted green energy amount related to conventional energies stored in the column *conventional* constitutes the percentage rate which is stored in the column *percentage* with the data type `DOUBLE`. Although the data stored in the column *percentage* is the only prediction data of this table which is actually accessed from the application, the other columns are likely to get useful for a further development of the application. No column in the table *predictiondata* allows `NULL` values.

4. Web Services (PHP files)

The purpose of the web services is to enable accessing an external database from within the Android application. This means we need to access a database which is located on a server and not on the devices where the application is installed because the energy data needs to be updateable.

It was waived to create extra files to describe the PHP-files using *PHPDoc* for commenting, as including their descriptions here is much more clearly and they are not the central part of this development.

4.1. File *signup.php*

This web service is accessed from within the Android application using the Async-task `Signup`. The purpose of this file is to insert a new dataset into the table containing user data.

The input of this web service is the username and the password which were inserted by the user on the application's sign up-view. First, a connection from the web service to the external database needs to be established. If this cannot be accomplished successfully, the error code 1 is assigned to the variable `$result`, which is later returned as result of the web service. If the connection was established successfully, the table `ge_login` is queried for datasets where the username equals the username given as input. This needs to be included to check if the username does already exist in this table. Hence, if this is the case the error code 2 is assigned to the above mentioned variable `$result`. If the username does not exist already, the password is encrypted by applying the secure hash algorithm (SHA). Then a new dataset is created by inserting the data into the table `ge_login`. If the insertion is accomplished successfully, 0 is assigned to the variable `$result`; otherwise the error code -1 is assigned. Finally, the variable `$result` is returned as result of this web service.

4.2. File *login.php*

This web service is accessed from within the Android application using the Async-task `Login`. The purpose of this file is to verify a set of user data by comparing it to data stored in the database.

The input of this web service is the username and the password which were inserted by the user on the application's log in-view. First, a connection from the web service to the external database needs to be established. If this cannot be accomplished successfully, the error code 1 is assigned to the variable `$result`, which is later returned as result of the web service. If the connection was established successfully, the table `ge_login` is queried to retrieve the encrypted password of the dataset where the username equals the username given as input. If this database query does not retrieve a result, the username does not exist in the table `ge_login` and the error code 2 is assigned to the `$result` variable mentioned above. If the result of the database query is not empty, the inserted password is encrypted and compared to the data retrieved from the database. If the password cannot be verified, the error code 3 is assigned to the variable `$result`. In contrast, if the password was verified successfully, 0 is assigned to `$result` and the variable is returned as result of this web service.

4.3. File *getscore.php*

This web service is accessed from within the Android application using the Async-task `SyncScore`. The script performs the verification of a set of user data and enables to access a user's score stored in the database.

The input of this web service is the username and the password which are locally stored on the user's Android device. First, a connection from the web service to the external database needs to be established. If this cannot be accomplished successfully, `-1` is assigned to the variable `$result`, which is later returned as result of the web service. If the variable `$result` has the value `-1`, it indicates that the web service's purpose was not achieved. If the database connection was established successfully, the table `ge_login` is queried to retrieve the encrypted password as well as the score of the dataset where the username equals the username retrieved from the data stored on the user's device. If this database query does not retrieve a result, the corresponding dataset does not exist in the table `ge_login` anymore and `-1` is assigned to the `$result` variable. If the result of the database query is not empty, the password given as input is encrypted and compared to the data retrieved from the database. In case the password cannot be verified, `-1` is assigned to the variable `$result`. Otherwise if the password was verified successfully, the score data retrieved from the table `ge_login` is assigned to the variable `$result`. Finally, `$result` is returned as result of this web service.

4.4. File *setscore.php*

This web service is accessed from within the Android application using the Async-task `ScoreStorage`. The script accomplishes an update of a dataset accessing the table `ge_login`. Hence this web service is called when the user's score was increased and therefore needs to be changed in the database as well to assure the successful synchronization of the user's score.

The input of this web service is the username and the new score of the user which are temporary stored in the Android application. First, a connection from the web service to the external database needs to be established. If this cannot be accomplished successfully, `-1` is assigned to the variable `$data`, which is later returned as result of the web service. If the variable `$data` has the value `-1`, it indicates that the web service's purpose was not achieved. If the database connection was established successfully, the table `ge_login` is updated replacing the dataset's old score with the new one where the username equals the username retrieved as input. If this update query could be accomplished successfully, `0` is assigned to the variable `$data`; otherwise `-1` is assigned. Finally, `$data` is returned as result of this web service.

4.5. File *getenergynow.php*

This web service is accessed from within the Android application using the Async-task `EnergyNowAccess`. The purpose of this web service is to retrieve the current amount of green energy in relation to conventional energies as well as the predicted green energy amount for in one, three, and five hours from the external database.

The input of this web service is solely the time which is the initial point for retrieving the energy data. First, a connection from the web service to the external database needs to be established. If this

cannot be accomplished successfully, `-1` is assigned to the variable `$result`. If the variable `$result` has the value `-1`, it indicates that the web service's purpose was not achieved. If the database connection was achieved successfully, the time given as input is formatted. Then the table *energydata* is queried in order to retrieve the dataset corresponding to the formatted time in order to get the current amount of green energy. Next, the points in time for the predictions are calculated, by adding and converting one, three, and five hours to the time given as input. Furthermore, the table *predictiondata* is queried in order to retrieve the data sets corresponding to the calculated and formatted point in time. The purpose of this is to retrieve the predicted green energy amount. If the data was retrieved successfully, the inner class `Edata` is defined to store the green energy amounts and the according identifier as objects. An array with the size of four elements is created as well and the objects created for each entry of green energy amount are stored in the array. Finally the array is converted into the JSON-array `EData` which is stored in the variable `$output`, then `$output` is returned as the result of this web service.

The data retrieved through this script is necessary to display green energy information on the main view of the Android application. This script and the file described in section 4.6. replace an earlier web service version to retrieve energy data. In the initial version only one dataset with energy data could be retrieved at once. The kind of data to retrieve was defined using parameters. The previous version allowed to access data in a more dynamic way. Anyway the big drawback of the initial solution was the runtime performance which was much worse than it is now. As only one dataset per request was retrieved the script had to be called several times. For example, in order to retrieve predictions for the application's detail view ten queries to the web service had to be made and each dataset had to be processed individually. Hence the more static solution was evaluated as more efficient. The initial web service is replaced by this one for data needed in the application's main view and by *getpredictions.php* (see next passage) for data needed in the application's detail view.

4.6. File *getpredictions.php*

This web service is accessed from within the Android application using the Async-task `Predictions`. Retrieving the predicted green energy amount for the next ten hours from the external database is the purpose of this web service.

The input of this web service is solely the time which is the initial point for retrieving predicted energy data. First, a connection from the web service to the external database needs to be established. If this cannot be accomplished successfully, `-1` is assigned to the variable `$result`. If the variable `$result` has the value `-1`, it indicates that the web service's purpose was not achieved. If the database connection was achieved successfully, the points in time for the predictions are calculated and formatted based on the time given as input. Furthermore, the table *predictiondata* is queried using the retrieved points in time as condition to access the corresponding datasets. The result of this query is the predicted green energy amount in relation to conventional energies for the next ten hours following the point of time given as input. If the database access was completed successfully, the inner class `Edata` is defined to store the green energy amounts, the according identifier, and a color depending on the green energy amount as objects. The color depends on whether the amount of green energy is higher than the defined upper limit and whether it is higher than the defined lower limit. An array with the size of ten elements is created as well and the objects created for each entry

of predicted green energy amount are stored in the array. Finally the array is converted into the JSON-array `PData` which is stored in the variable `$output`, then `$output` is returned as the result of this web service.

The data retrieved through this web service is used to display predicted green energy data in a list on the detail view of the Android application. The background color of each list entry corresponds to the color assigned in this script. If the predicted amount of green energy is higher than the lower limit, the background color of this list entry is green. If it is between the higher and the lower limit, the background color is grey. If the predicted green energy amount is even lower than the lower limit, the list entry's background color is orange.

5. Testing

We perform testing on our application in order to:

- Improve testing coverage
- Test application assumptions
- Validate application functionality

We used two methods for testing our application:

- a) Unit Testing: Many software methodologies these days work compatibly with unit testing. Unit tests can greatly increase the stability and quality of an application and reduce testing costs. Unit tests come in many forms: UI testing and Intent testing.
- b) Integration (Functional/Behavioral) Testing: Functional testing involves verifying that individual application components work together as expected by the user. For example, you can create a functional test to verify that an Activity correctly launches a target Activity when the user performs a UI interaction.

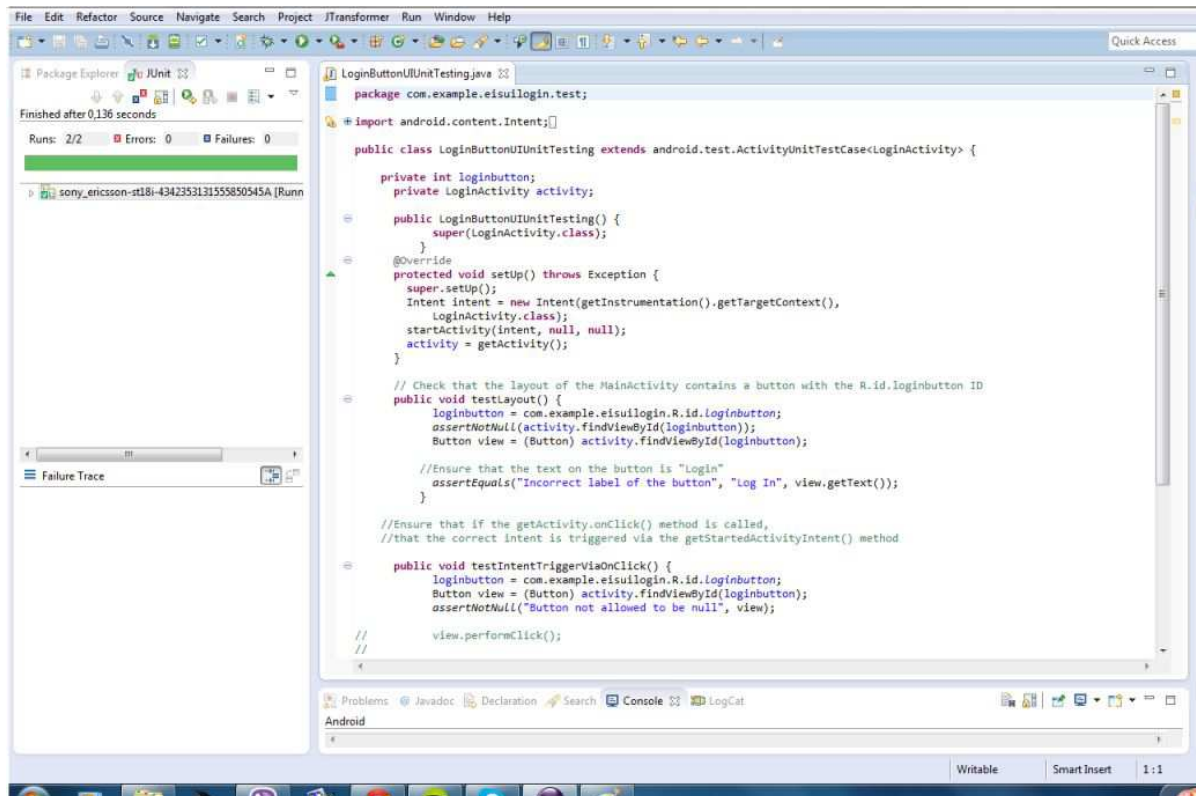
The preferred way of organizing tests is to keep them in separate Android test projects or source folders. Therefore, we built a separate Android project named *GreenAppTest*. The project under test, *Green Energy Shifting* must be added as dependency to the test project, *GreenAppTest*.

The *AndroidManifest.xml* file of *Green Energy Shifting* must also specify that the test project uses the *android.test.runner* library and specifies the test runner for the unit test. The test project also specifies the package of the application to test in the *AndroidManifest.xml* file under the `android:targetPackage` attribute.

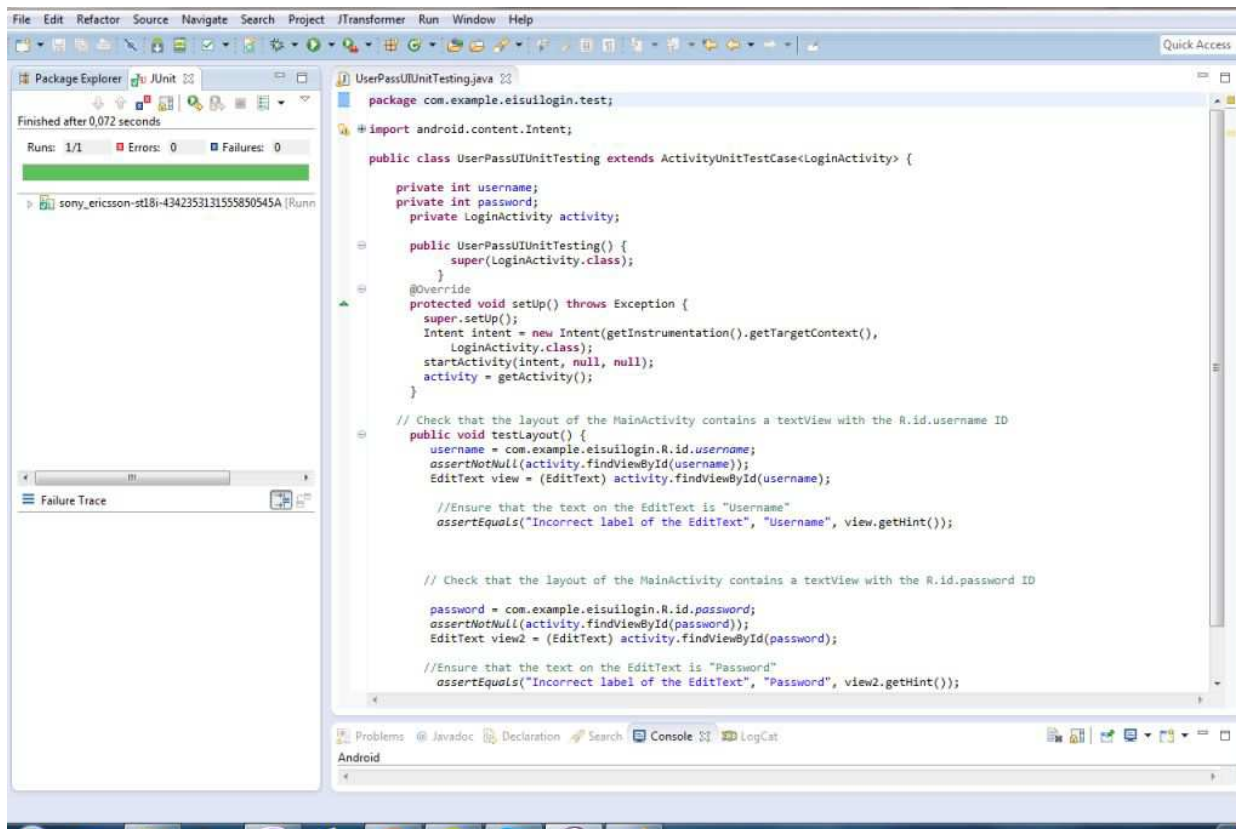
Android provides different classes for unit and integration tests. Our Android test application contains five different test classes. For Unit testing, *android.test.ActivityUnitTestCase* class and for Integration testing, *android.test.ActivityInstrumentationTestCase2* class is used.

5.1. Unit Testing

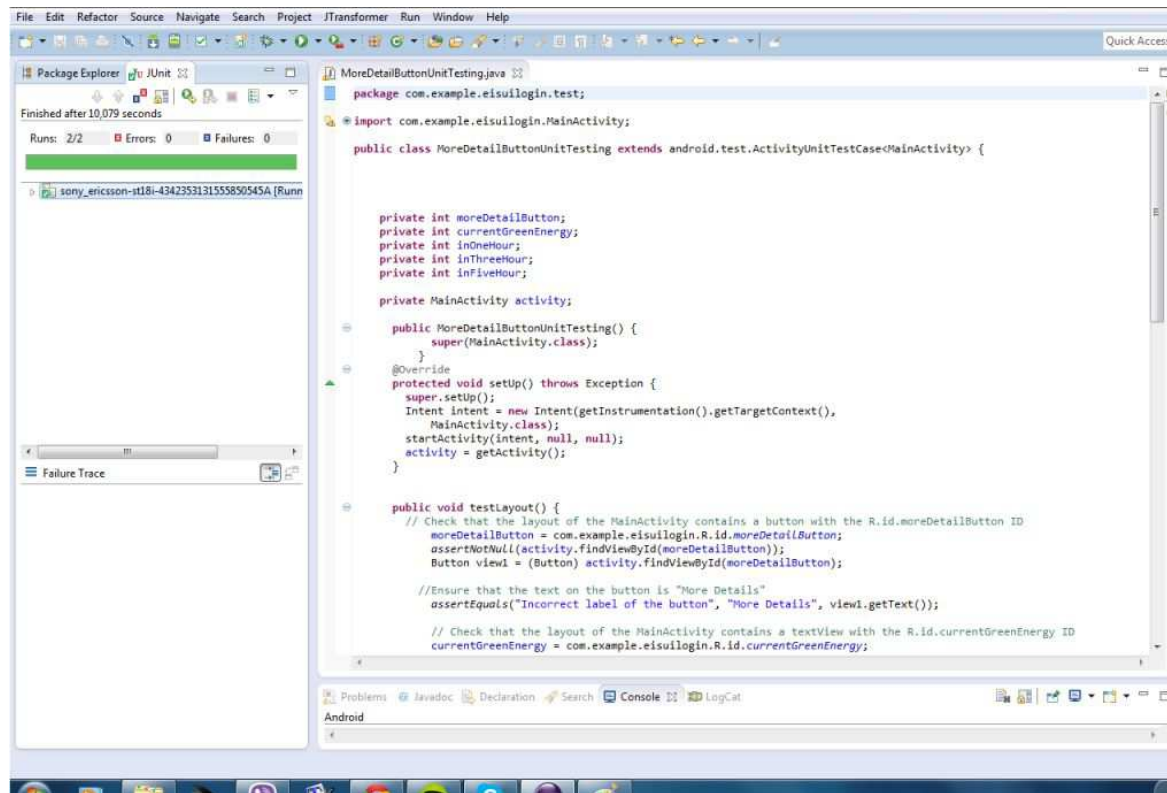
1. *LoginButtonUIUnitTesting*: In this class, we check if there is a button and if its text is "Log In".



2. *UserPassUIUnitTesting*: This class checks if there is an edit text for "Username" and another edit text for "Password". It also checks the label of the hint of these edit texts.

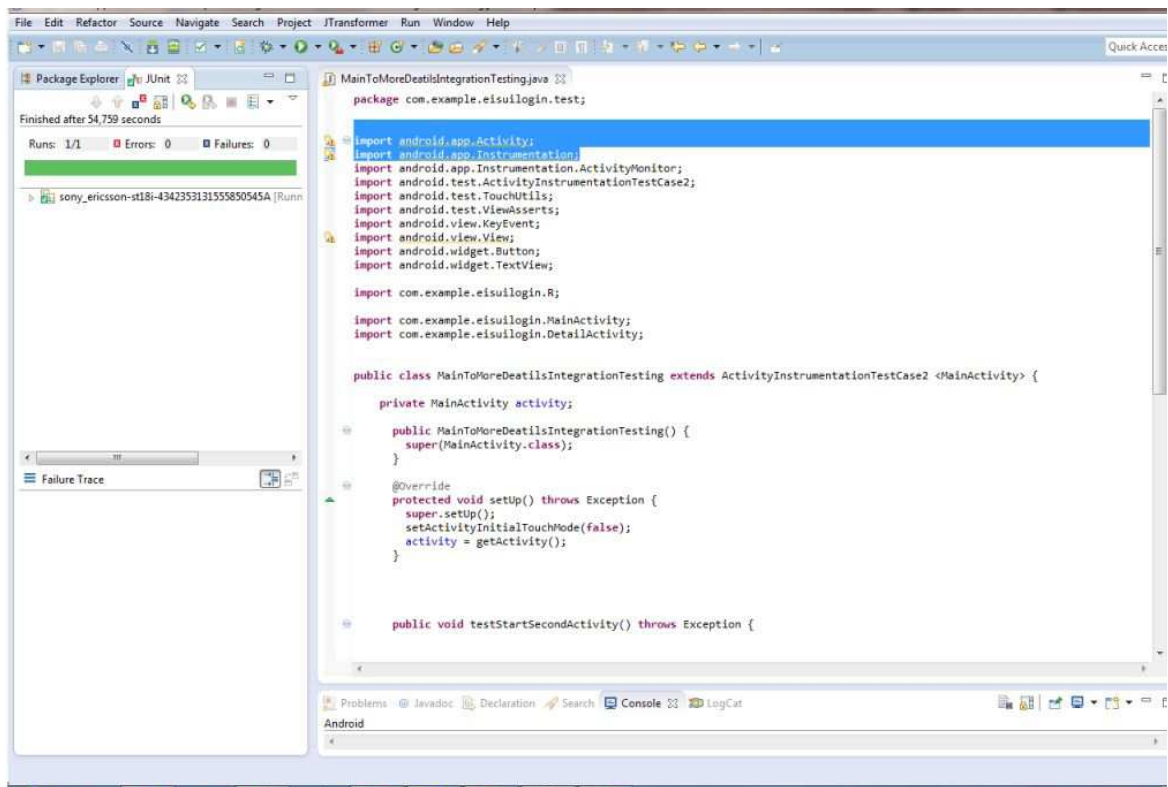


3. *MoreDetailButtonUnitTesting*: First, we check if there is a button and if its text is "More Details". Next, it determines if by clicking the "More Details" button, the corresponding intent was issued, not if the second activity was started. To implement this, we send a parameter to the second created activity using `putExtra()` and then we checked it using `getExtras()`. Additionally, this class ensures if the text on the text views of the predictions are equal to what we defined in our code.



5.2. Integration Testing

4. *MainToMoreDetailsIntegrationTesting*: This class checks if by clicking the "More Details" button, the application goes from the main activity to the second activity which is `DetailActivity` and the activity is correctly started. To implement this, it checks if the title of the second page is on the screen, as you can see in the screenshot on the next page.



6. Sources

- <http://www.eex.com/de/marktdaten/marktdaten-download/produktinhalte/transparenzdaten>
- <http://de.statista.com/statistik/daten/studie/1807/umfrage/erneuerbare-energien-anteil-der-energiebereitstellung-seit-1991>
- http://de.wikipedia.org/wiki/Erneuerbare_Energien
- <http://www.bmwi.de>
- <http://www.bdew.de>
- <http://rheinenergie.com>
- <http://www.stadtwerke-bonn.de/energieundwasser.html>
- <http://www.umweltbundesamt.de>