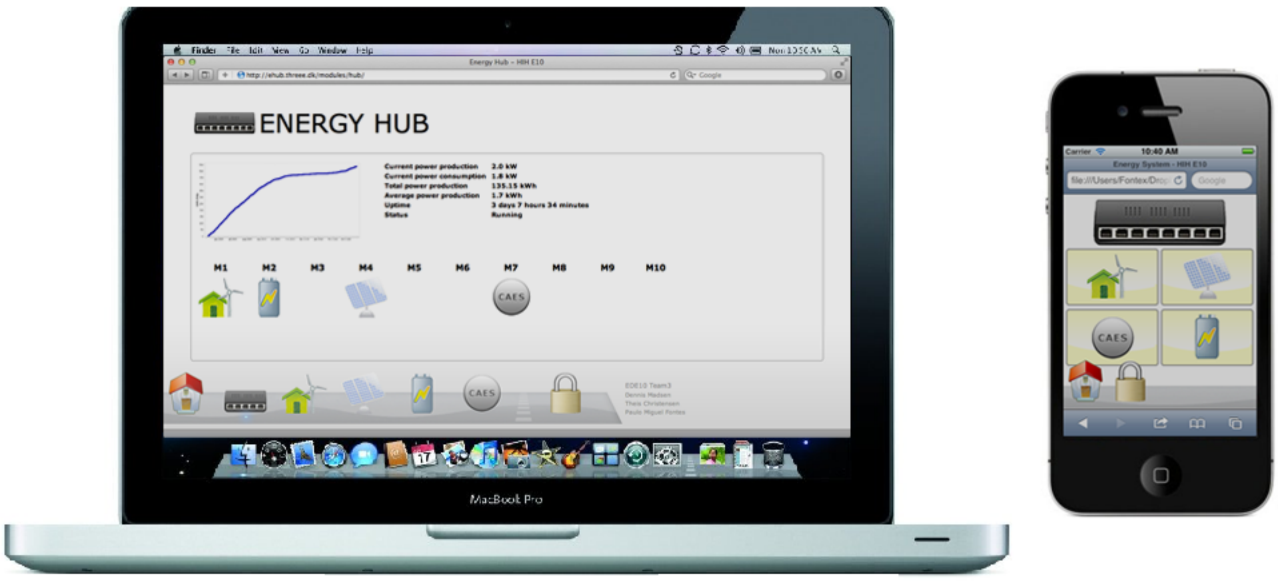
WEB2 - Web Technologies

Project - Energy Hub

Counselor: Martin Olsen



Current power production 2.0 kW
Current power consumption 1.8 kW
Total power production 135.15 kWh
Average power production 1.7 kWh
Uptime 3 days 7 hours 34 minutes
Status Running

BY E10 - Team3:

Theis Christensen (10691)

Dennis Madsen (90248)

Paulo Fontes (10484)

Table of Content

1 Introduction

This report provides an understanding and documentation of a dynamic web interface application part of an energy hub system. An overview off the energy hub project is given in this introduction, for an easy understanding of the functionalities on the web interface. The report is divided into sections starting with an analysis of the system to be, fowling by the design and implementation for a scaled down prototype version of the energy hub project.

For the development and evaluative procedures, a LAMP(Linux Apache MySQL PHP) web server is set up, this process is explained in the implementation section of this report. The web interface can be accessed on the university network at the address http://10.1.18.223/.

The web interface application includes a great amount of different scripting languages such as server side PHP and client side JavaScript, along with markup languages HTML and XML. The code for each will not be explained extensively, instead the most important functionalities will be filtered and explained in great detail.

MySQL is a requirement in this web interface application, being the most used relational database in a open-source project environment. The integration between MySQL and PHP is easy and well documented making the development process faster. The development of the database is made more interactive and intuitive using phpMyAdmin tool, this tool is installed and explained in the web server implementation sections, and can be accessed at: http://10.1.18.223/phpMyAdmin/ , using the user name **eval** and password **ede10eval** with view only permissions.

An extra web application is developed in PHP only for evaluative propose, it allows the user to view the code of all the files on the web server file structure. By this method the last version of the code is always updated for visualization. The application is called SeeIt and can be accessed at the address: http://10.1.18.223/SeeIt/ . This web application is not explained or documented in this report due to its simplicity and off the topic methods.

The web interface layout was built with the help of different interaction design sessions, used in Interaction Design 1 and Web Technologies 1 courses. With the help of this courses, the web page layout was built in HTML and the user experience was improved from the feedback of the interaction design sessions. The web site graphical layout analysis and implementation is not explained or documented in this report, it can be seen on the appendix on the CD-ROM given.

1.1 Energy Hub Overview

An energy hub is a device able to route the energy to the devices connect to it in the most efficient and effective way, the energy hub is part of an green energy system, the devices connected to the it are: photovoltaic panels, wind turbines, batteries, and a dynamic number os different modules that can me available in future.

The user have to be able to analyse stored measurements, the values are retrieved by the modules to the energy hub and this is responsible to store the data so it can be available to the end user (web interface). A scaled down prototype is made for the energy hub, this is made to give and overview over the system to be in is main functionality: the communication between modules and energy hub, communication between the energy hub and the web server and the ability of switch the power between the connected modules.

2 Analysis

In this section an extensive analysis of the web interface is explained, using visual aids for the easy understanding of the system functionalities.

2.1 Database

For a well constructed database, it has to be consistent, flexible and efficient so no data is lost or repeated when data is saved. The analysis of the system information given is fundamental for the overview and development of the database structure.

The following use cases are retrieved from the pre-project of the energy hub project. (Project 3 found in appendix):

Use Cases

- Jan is looking at the web interface for the energy hub. From where he can see the status for each modules connected to it in a graphically way.

- A new energy module is connected to the hub, Jan opens the web interface for the energy hub, he logs in the administrator section of the system to start, stop or see a more detailed overview of each module.

- Jan arrives at the university in the morning and an email was send to him reporting a failure in the green energy system, he login to the administrator web interface, and he can see what the problem might be, and if it's possible to solve it directly on the interface.

From this use cases given by the customer the relevant information for the database design is filtered:

- The status of each module have to be shown.

- A detailed overview of each module such as current production (Voltage, Current, Power, etc.), efficiency of the module, etc.

- Errors have to be reported to the system administrator / maintainer.

In a deeper analysis of the information system for the power energy hub, some technical data is needed to be stored such as:

- Each module have is unique id, this is similar to the MAC address, this way when a module is connected is possible to check if the module have been connected before, so the data can be stored for the same module instead of instantiate a new module which can generate ambiguous data.

- Three types of modules are defined: input, output and bidirectional. This are seen from the power hub point of view where inputs are producers (solar panels, wind turbine, ...), output are consumers (Inverter, 30V output socket) and bidirectional (C.A.E.S, batteries, ...).

Data sets From the system information analysis a basic tabular structure is designed for a better understanding and low level overview of the database, this is called data set structure.

TYPE(ID_TYPE, NAME);

ID_TYPE	Auto increment integer, a new id number is generated when a different type is needed to the system.
NAME	Describe the type name for example: Input, output, bidirectional, etc.

STATUS(ID_STATUS, NAME);

ID_STATUS	Auto increment integer, a new id number is generated when a different status is needed to the system.
NAME	Describe the status name for example: Running, stopped, warning, etc.

MODULES(ID_MODULE, ID_TYPE, NAME, HUB_PORT);

ID_MODULE	Module unique ID, this id as primary key ensures that no module with is repeated in the database.
ID_TYPE	This is a foreigner key for the table type, this way if some other type of module is needed it can be dynamical add.
NAME	The name of the module for example, Solar Panel, wind turbine, battery.
HUB_PORT	Actual connected port for this module

As a requirement, the database have to store the measurement from different sensors. This is stored in a the table MEASUREMENTS.

MEASUREMENTS(ID_MEASURE, ID_SENSOR, DATE_TIME, HUB_PORT, VALUE)

ID_MEASURE	Measurement id, auto increment field.
ID_SENSOR	This is a foreigner key for the table sensor, this associated the value retrieved to the correspondent sensor.
DATE_TIME	Date and time of the measurement is saved so it can be plotted or in case of a lower efficiency a detailed history can analysed.
HUB_PORT	Actual connected port for this module
VALUE	Measurement retrieved by the energy hub.

Logs are a simplified method of keeping track about what is happening in the system, this is one of the most fundamental functionalities of the system.

LOGS(ID_LOG, ID_MODULE, DATE_TIME, ID_STATUS, ID_USER, ID_ERROR);

ID_LOG	Auto increment integer, a new id number is generated every time a measurement is add.
ID_MODULE	This is a foreigner key for the table modules, this allow the system to know from which module correspond the measurement .
DATE_TIME	Date and time of the log is saved, in case of malfunction it will help with a time line.
ID_STATUS	The new status of the module that was changed.
ID_USER	Foreigner key for the table users, this will allow the system to know which user made the change.
ID_ERROR	Foreigner key for the table error, if any error occurs it will be stored.

For error handling situations a table ERRORS is created this way the administrator can have more control and act in case some mall function of the system.

ERRORS(ID_ERROR, NAME);

ID_ERROR	Auto increment integer, a new id number is generated when a different unit is needed to the system.
NAME	Predefined error description.

Measurements are add to the database constantly, which in a non-stop system database size could be a problem, to avoid this situation a MERGES data set is created. This will store an average of values between a time span for each sensor, allowing the customer either to erase the values from the log or export them out from the database.

MERGERS(ID_MERGE, DATE_FROM, DATE_TO, ID_SENSOR, VALUE);

ID_MERGE	Auto increment integer, a new id number is generated every time an wrap is needed.
ID_SENSOR	This is a foreigner key for the table sensors, this allow the system to know from which sensor correspond the values .
DATE_FROM	Date and time of the time span start.
DATE_TO	Date and time of the time span end.
VALUE	Average measurements value.

SENSOR(ID_SENSOR, ID_MODULE, ID_UNITS);

ID_SENSOR	Auto increment integer, a new id number is generated when a different sensor is needed to the system.
ID_MODULE	This is a foreigner key for the table sensors, this allow the system to know from which module correspond the sensor, being a primary key with the id_sensor, this way each sensor correspond to only one module .
ID_UNITS	This is a foreigner key for the table units, this allow the system to know which units the sensor is measuring.

UNITS(ID_UNIT, NAME);

ID_UNIT	Auto increment integer, a new id number is generated when a different unit is needed to the system.
NAME	Describe the unit name for example: A, V, deg ,m/s,etc. (Ampere, Volt, Degrees, Velocity)

To increase security, an USERS table is added associated with different PRIVILEGES for different users. A user root is created leaving the system able to manage multiple users with different privileges if needed.

USERS(ID_USER, NAME, PASS, EMAIL, ID_PRIV);

ID_USER	Auto increment integer, a new id number is generated when a different privilege is needed to the system.
NAME	Username credential.
PASS	User password (Not encrypted in this version).
EMAIL	Email to which warnings are send.
ID_PRIV	Foreigner key for the table privileges.

PRIVILEGES(ID_PRIV, NAME);

ID_UNIT	Auto increment integer, a new id number is generated when a different privilege is needed to the system.
NAME	Describe the user privileges.

For the communication process between the web interface and the energy hub, the ip address of the device need to be always accessible this way a table DEVICES is created in which the last known IP address of the device is stored.

DEVICES(ID_DEVICE, IP);

ID_DEVICE	Auto increment integer, a new id number is generated when a different ip added.
IP	IP address of the device.

At this point, in an abstract way, the database can initialise a new module or identify if the module where connected before, change the status for each module and add new measurements for each sensor.

Data Model Data model is a high-level overview of the database structure. At this stage data sets are translated to a logic structure with the relationship between tables.

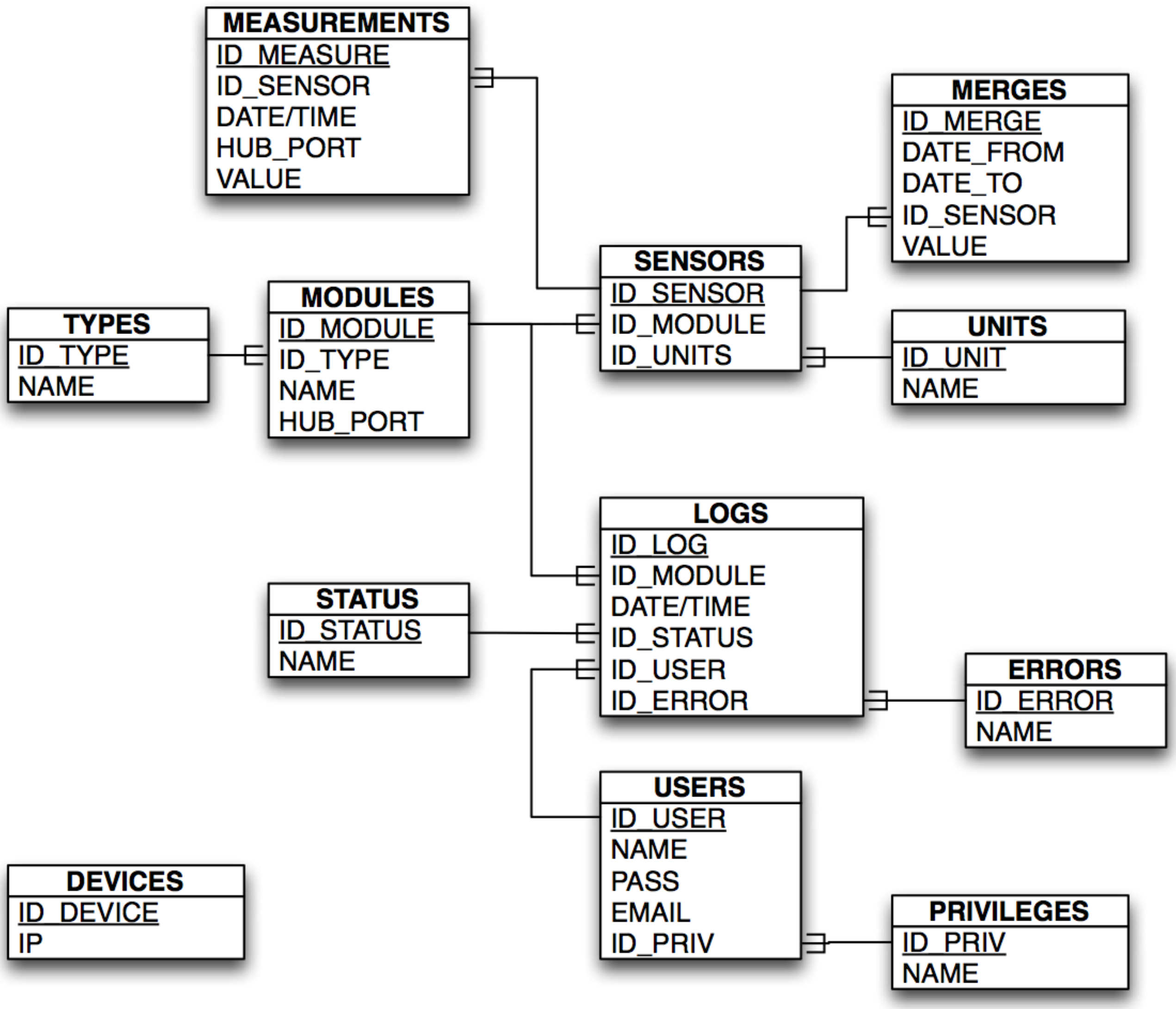


Figure 2.1: Database relational overview (Data Model)

2.2 File Structure

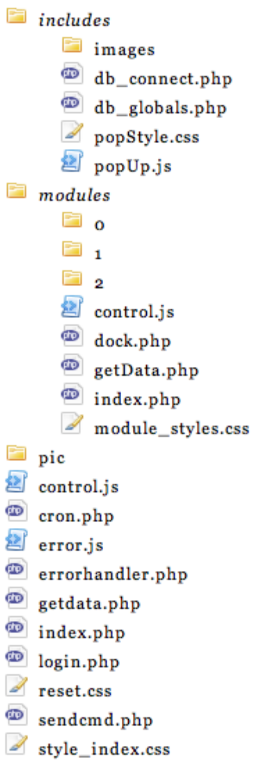
The file structure defines how files are grouped on a system, in a web server architecture the files are stored in the root directory defined by the web server (Apache) configuration. This section describes the analysis of a file structure that allows the scalability of the system.

Overview New modules are added are removed from the energy hub as needed. Some modules have different functionalities that have to be taken in consideration when defining the file structure.

In the root directory of the web server the main page is the entrance to the web interface navigation, from here the user can see the modules connected to the energy hub, and some values about the system such as the efficiency of the green energy system. From the main page the user is able to navigate to the different modules page.

To keep the system dynamic, and flexible the modules page are included inside a predefined layout using PHP. This will keep the graphic layout and navigation system in each page consistent and further improvements for the user experience can be made.

The proposed file structure can be seen bellow:



```
includes
    images
    db_connect.php
    db_globals.php
    popStyle.css
    popUp.js
modules
    0
    1
    2
    control.js
    dock.php
    getData.php
    index.php
    module_styles.css
pic
control.js
cron.php
error.js
errorhandler.php
getdata.php
index.php
login.php
reset.css
sendcmd.php
style_index.css
```

Figure 2.2: Proposed File structure

A short description for each script listed above can be seen in this list:

- includes/images/ - Contains images related with the system functionalities.
- includes/db_connect.php - Handle the connection to the MySQL database.
- includes/db_globals.php - Includes all the global variables with the credentials for the database.

- includes/popStyle.css - Popup login style definition.
- includes/popUp.js - Client side script to handle the user login.
- modules/[0-2] - Modules pages, the numbers are associated with the modules ids, being 0 the hub, 1 the battery and 2 the photovoltaic panels.
- modules/control.js - Client side script, makes background requests to a PHP script, this script is part of the user experience optimization.
- modules/dock.php - PHP script contains the navigation system for the web interface.
- modules/getdata.php - PHP script called in background by the client side script running recursively control.js, this script retrieves the system status and data form database in real time, is part of the user experience optimization.
- modules/index.php - This page includes each module page structure.
- modules/module_styles.css - Graphical layout definition.
- control.js - Client side script, makes background requests to a PHP script and handle the connections of new modules, this script is part of the user experience optimization.
- cron.php - Cron jobs are running by the server in a predefined time, in this case the cron.php at the web server root will point to the cron jobs inside each module folder.
- error.js - Client side script, make background requests to a PHP script part of the error handling system.
- errorhandler.php - PHP script called in background by the client side script running recursively, is part of the error handling system.
- getdata.php - PHP script called in background by the client side script running recursively control.js, this script retrieves the modules connected to the system in real time. Is part of the user experience optimization.
- index.php - First page of the web interface.
- login.php - This PHP script is called in background by the pop up login client side script.
- reset.css - Layout definition of the web interface.
- sendcmd.php - Webservice to send commands to the desired module/enegy hub.
- style_index.css - Layout definition of the web interface.

Due to the size of such system and the few time for the development, the project was scaled down so a high fidelity prototype where the system to be can be implemented and the main functionalities of the system tested.

2.3 Communication

The web interface is the connection between the user and the system, in this interface the user have to be able to see the data retrieved by the modules and at the same time be able to send commands to the energy hub/modules to do predefined tasks.

To retrieve the measurements from the modules, an application running at the energy hub translate the data retrieved from the module through PLC to a URL request at the web server. In the server side a script is implemented to translate this request to a SQL query that saves the measurement to the database. It is important to define which data the server side script will expect from the energy hub request. Since each module can have more than one sensor is important to retrieve the sensor id, the measured value and the energy hub port that this module is connected to.

The communication between the energy hub and the web server isn't exclusive for adding new measurements to the database, this communication is also used to update the database regarding the status of the modules in case of new connection, disconnection, a malfunction or stopped for security reasons. The data is saved in the table logs and shown to the user.

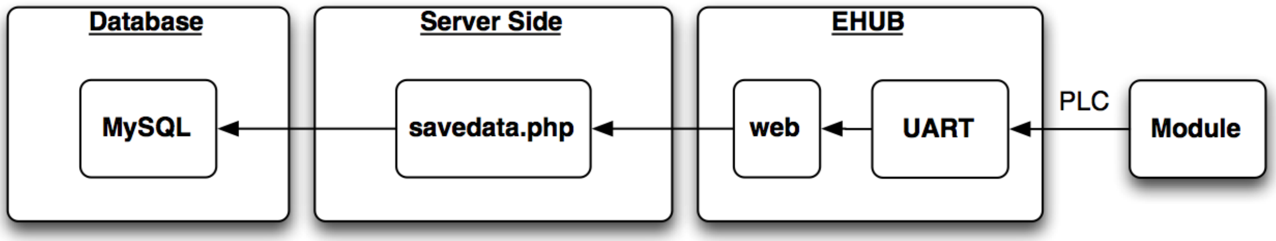


Figure 2.3: Save data to the database

As defined in the Project 3 (in appendix) requirements the system have to give the possibility for the administrator to control the modules and the energy hub. As such the web server have to stablish a connection to the system so commands can be send. A feedback of success or not is always given to the user since the command to be send can be of great importance for the well function of the system.

The flow of communication from the user until the final destination in this case the module or the energy hub.

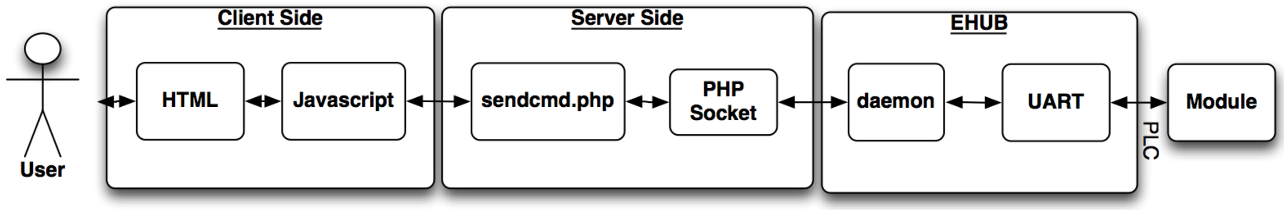


Figure 2.4: Send a command to a module or energy hub

A background application running at the energy hub, ensure that after a reboot, the IP address assigned by DHCP to the energy hub is saved in the database, this way commands can be send through the web interface. The system becomes more flexible in case of internal network changes.

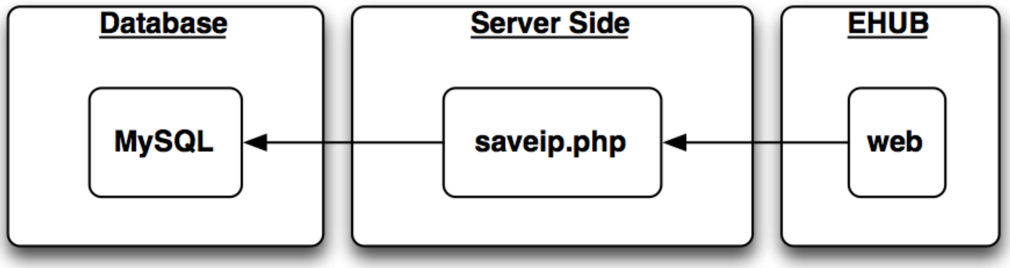


Figure 2.5: Save the IP address of the Embedded device

2.4 Error Handling

The web server have to be able to handle errors such the lost of communication with the energy hub or database connection problems. For a reliable system, a script running on the client side using JavaScript have to be implemented, this way the web interface can recursively test the communication to the energy hub and database.

Is necessary to define the web interface response for both situations database and energy hub connection lost. If the connection is lost to the device the user is able to see the data stored in the database, so the system is not blocked, but a warning is shown to the user/administrator, do the correspondent debug can be done. In case the connection to the database is lost, the user interface is blocked with an warning, the energy hub will continue working, since it doesn't affected the correct work of the system to route the energy, in this situation all the data retrieved by the modules will be lost.

Since a script is running recursively at the client side, when the communications are re-establish to the energy hub or database the user will be able to use the system normally, this is a great improvement in the user experience and functionalities as an web application.

3 Implementation

The development of the web interface is made in a local machine using MAMP (MAC Apache MySQL PHP), and transfer later on to a web server on the university network.

3.1 Web server set up

The approach used to implement a small web server on a Linux environment is described, the web server can be access at the address: http://10.1.18.223, it will be used in the evaluative and testing process

The steps used to install a web server on a machine running the Linux distribution Centos are described bellow:

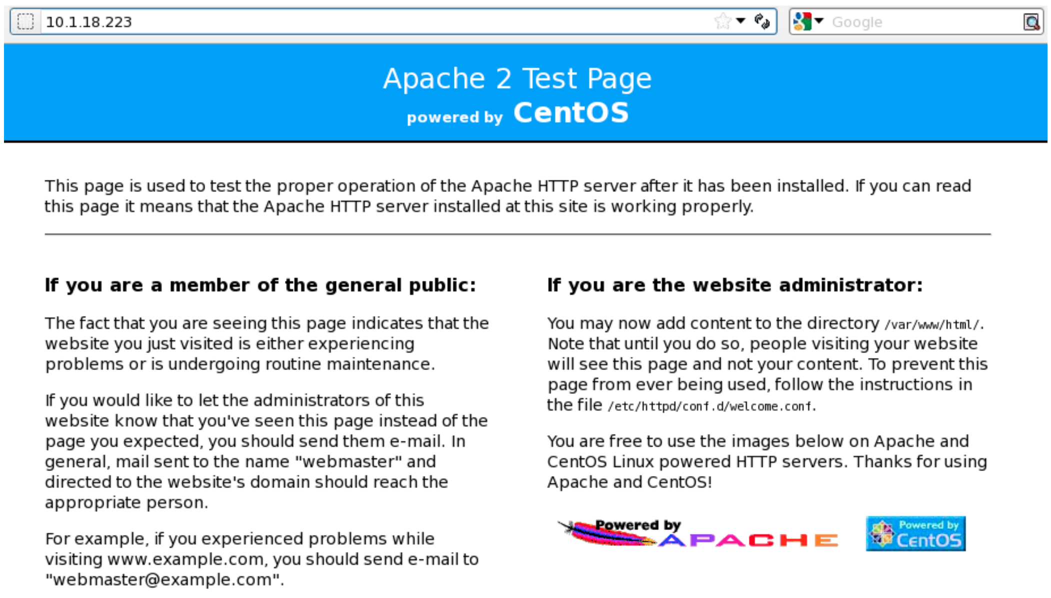
- Installing Apache and bring the server up.
  Install daemon that is going to handle HTTP requests on port 80. (Apache)

  ```
  # yum install httpd
  ```

  Start Apache as background process.

  ```
  # /etc/init.d/httpd start
  ```

  The server is tested by typing in a web browser the ip address of the web server, if the installation is successful the index page is loaded being on a newly installed system the following page.


10.1.18.223

Apache 2 Test Page
powered by CentOS

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting www.example.com, you should send e-mail to "webmaster@example.com".

If you are the website administrator:

You may now add content to the directory /var/www/html/. Note that until you do so, people visiting your website will see this page and not your content. To prevent this page from ever being used, follow the instructions in the file /etc/httpd/conf.d/welcome.conf.

You are free to use the images below on Apache and CentOS Linux powered HTTP servers. Thanks for using Apache and CentOS!

Figure 3.1: First page apache server running

- Installing MySQL and bringing the server up
  Installing MySQL server and client.

```
# yum install mysql-server mysql
```

Start MySQL server as background process.

```
# /etc/init.d/mysqld start
```

Testing the MySQL server.

```
# mysql
```

Server up and running when the follow response is given.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 5.0.95 Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

At this moment the database is set with username root and an empty password. For security reasons a password is set for the MySQL server.

Changing MySQL root password

```
mysql> USE mysql;
mysql> UPDATE user SET Password=PASSWORD('new password') WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

With the instruction **USE mysql** the database directory mysql is open, and with **UPDATE user SET Password=PASSWORD ('new password') WHERE user='root'** the password for the user 'root' is updated to the desired one. **FLUSH PRIVILEGES** erases the privileges in cache and reloads the the new privileges in the MySQL server.

- Installing PHP
  Install PHP scripting language and necessary modules for MySQL server integration.

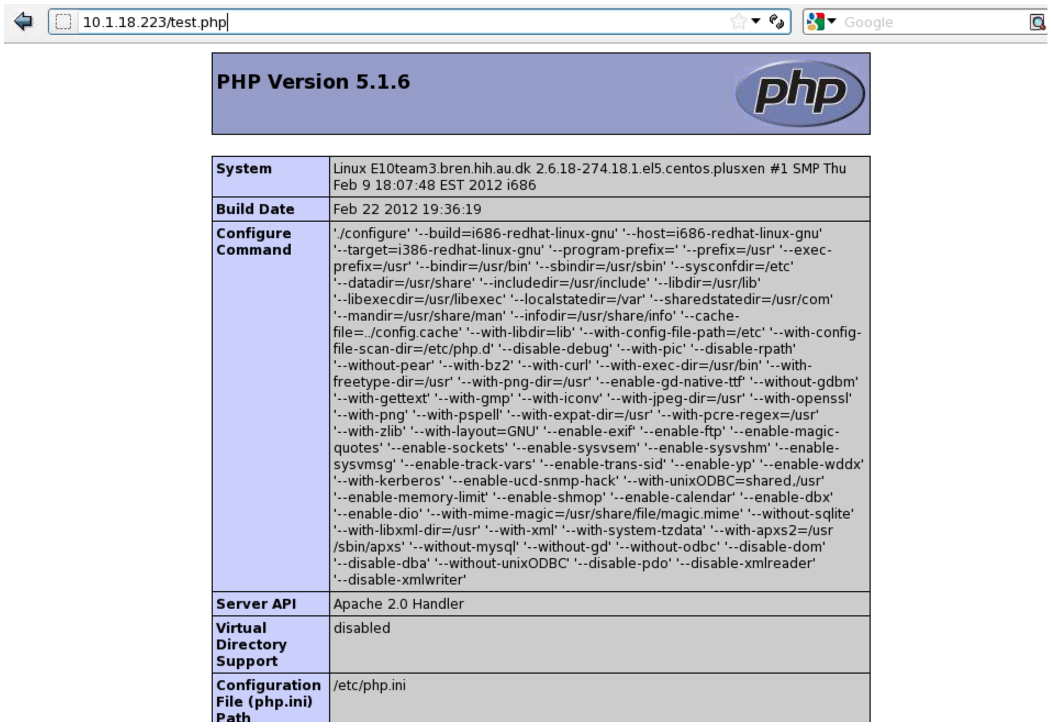
```
# yum install php php-mysql
```

php: scripting language compiler
php-mysql: mysql functions to be called from a PHP script

A file is created and added to the root directory of the web server, this file will retrieve all informations regarding Apache, PHP, MySQL and Modules installed.

```
<?php
 // Shows the in use PHP configuration file.
 phpinfo();
?>
```

Pointing the browser to the address http://10.1.18.223/test.php, will give the following result.



PHP Version 5.1.6

System	Linux E10team3.bren.hih.au.dk 2.6.18-274.18.1.el5.centos.plusxen #1 SMP Thu Feb 9 18:07:48 EST 2012 i686
Build Date	Feb 22 2012 19:36:19
Configure Command	'./configure' '--build=i686-redhat-linux-gnu' '--host=i686-redhat-linux-gnu' '--target=i386-redhat-linux-gnu' '--program-prefix=' '--prefix=/usr' '--exec-prefix=/usr' '--bindir=/usr/bin' '--sbindir=/usr/sbin' '--sysconfdir=/etc' '--datadir=/usr/share' '--includedir=/usr/include' '--libdir=/usr/lib' '--libexecdir=/usr/libexec' '--localstatedir=/var' '--sharedstatedir=/usr/com' '--mandir=/usr/share/man' '--infodir=/usr/share/info' '--cache-file=../config.cache' '--with-libdir=lib' '--with-config-file-path=/etc' '--with-config-file-scan-dir=/etc/php.d' '--disable-debug' '--with-pic' '--disable-rpath' '--without-pear' '--with-bz2' '--with-curl' '--with-exec-dir=/usr/bin' '--with-freetype-dir=/usr' '--with-png-dir=/usr' '--enable-gd-native-ttf' '--without-gdbm' '--with-gettext' '--with-gmp' '--with-iconv' '--with-jpeg-dir=/usr' '--with-openssl' '--with-png' '--with-pspell' '--with-expat-dir=/usr' '--with-pcre-regex=/usr' '--with-zlib' '--with-layout=GNU' '--enable-exif' '--enable-ftp' '--enable-magic-quotes' '--enable-sockets' '--enable-sysvsem' '--enable-sysvshm' '--enable-sysvmsg' '--enable-track-vars' '--enable-trans-sid' '--enable-yp' '--enable-wddx' '--with-kerberos' '--enable-ucd-snmp-hack' '--with-unixODBC=shared,/usr' '--enable-memory-limit' '--enable-shmop' '--enable-calendar' '--enable-dbx' '--enable-dio' '--with-mime-magic=/usr/share/file/magic.mime' '--without-sqlite' '--with-libxml-dir=/usr' '--with-xml' '--with-system-tzdata' '--with-apxs2=/usr /sbin/apxs' '--without-mysql' '--without-gd' '--without-odbc' '--disable-dom' '--disable-dba' '--without-unixODBC' '--disable-pdo' '--disable-xmlreader' '--disable-xmlwriter'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php.ini

Figure 3.2: test.php - PHP configuration file

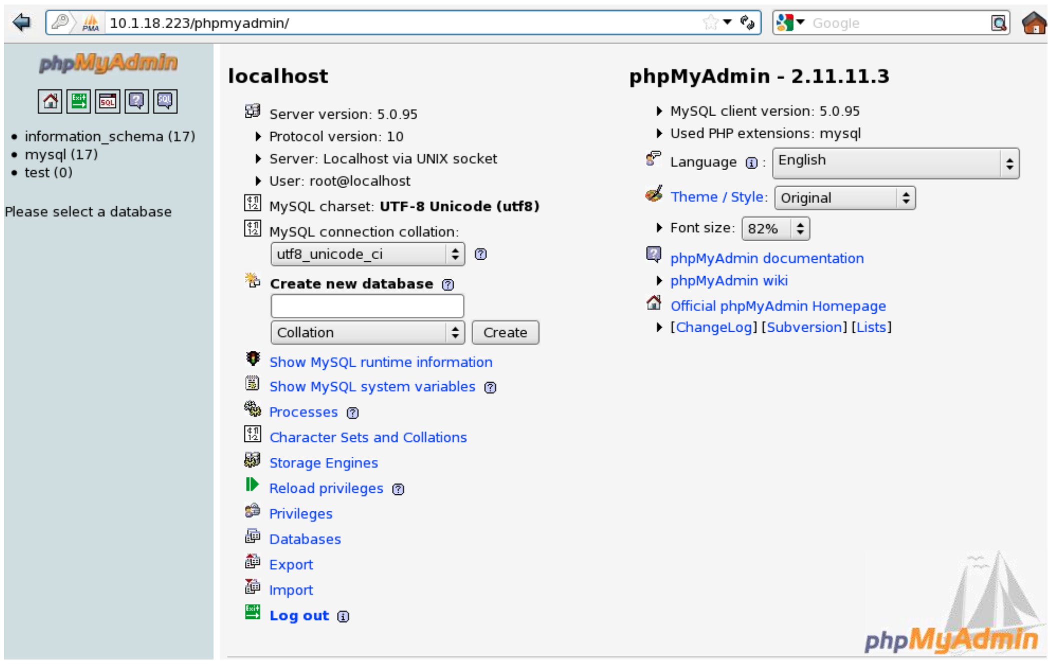
- Installing phpMyAdmin
  phpMyAdmin is a free web based MySQL database administration environment, without this tool the development of a MySQL database have to be made using the command line, it has become a fundamental tool for most web masters, since the development is faster and fewer errors are made.

```
# yum install phpmyadmin
```

The Apache have to be restarted so it can assume the symbolic link to phpMyAdmin tool.

```
# /etc/init.d/httpd restart
```

Testing phpMyAdmin by pointing in the web browser to the address http://10.1.18.223/phpmyadmin. The credential used for evaluation purpose are Username: eval Password: ede10eval


10.1.18.223/phpmyadmin/

- information_schema (17)
- mysql (17)
- test (0)

Please select a database

localhost

- Server version: 5.0.95
- Protocol version: 10
- Server: Localhost via UNIX socket
- User: root@localhost
- MySQL charset: **UTF-8 Unicode (utf8)**
- MySQL connection collation: utf8_unicode_ci
- **Create new database**
- Collation Create
- Show MySQL runtime information
- Show MySQL system variables
- Processes
- Character Sets and Collations
- Storage Engines
- Reload privileges
- Privileges
- Databases
- Export
- Import
- **Log out**

phpMyAdmin - 2.11.11.3

- MySQL client version: 5.0.95
- Used PHP extensions: mysql
- Language: English
- Theme / Style: Original
- Font size: 82%
- phpMyAdmin documentation
- phpMyAdmin wiki
- Official phpMyAdmin Homepage
- [ChangeLog] [Subversion] [Lists]

Figure 3.3: phpMyAdmin front page

3.2 Database

The tool phpMyAdmin is used to accelerate the process of implementation of the database, the SQL statements generated by the tools is shown in this section and explained in detail. The database storage engine by default is set to MyISAM, this is changed to InnoDB so foreigner keys and relationships can be used. To create the tables to store data, a database directory have to be created at first using the statement:

```sql
CREATE DATABASE ‘iEnergy‘;
```

With the database directory created, the tables can now be created into it. Since the syntax to created the tables is the same, an example table that shows a all possible SQL keywords used in this project is used.

```
  CREATE TABLE IF NOT EXISTS `MEASUREMENTS` (
    `ID_MEASURE` int(11) NOT NULL auto_increment,
    `ID_SENSOR` int(11) NOT NULL,
    `DATE_TIME` datetime NOT NULL,
    `HUB_PORT` int(11) NOT NULL,
    `VALUE` float NOT NULL,
    PRIMARY KEY  (`ID_MEASURE`),
    FOREIGN KEY (`ID_SENSOR`) REFERENCES `SENSORS` (`ID_SENSOR`)
    ) ENGINE=InnoDB;
```

CREATE TABLE, as the name indicates, is used to create a new table into the database directory. The keywords IF NOT EXIST are add for consistence measures, so the table is not created twice.

To easy identification of the data, all tables have an ID field, this is set as PRIMARY KEY and AUTO_INCREMENT, it makes sure that new data will have the next ID number and it will be unique. The ID can be as well used to the relationship between tables, in this example the ID_SENSOR is a foreigner key of the table SENSORS. The ID_SENSOR is the primary key of the table SENSORS and is related to this table making a constrain in the data to be inserted. When a new data is inserted, the ID_SENSOR value have to exist in the table sensors, otherwise an error will occur.

At first the name of the field is defined, for example HUB_PORT will be the name of the field followed by the data type, in this case an integer. The keyword NOT NULL makes impossible to add new data without a defining a value to this field.

The ENGINE keyword defines the storage engine to be used in this table, as explained in the beginning of this section, the use of InnoDB is needed for the relationships and foreigner keys.

getData.php This script is described in this section since is main functionality is to retrieve values from the database using SQL queries and construct a XML file with the data retrieved. Only the Ampere measurements are requested and saved into this database since this is prototype of the system the necessary protocol and structure for the sensors in the modules was not defined.

Script parameters:

- unique_id - this is the only parameter expected by this script, since it will retrieve all the measurements from an unique module/sensor, the URL encoding method for this parameter have to be POST.

Several requests to the database are done by this script, each SQL query and is described bellow:

Current power production of the modules:

```
// Current Power Production and Current Power Consuption
  $sql = "SELECT  VALUE FROM `MEASUREMENTS` WHERE `ID_SENSOR`=".$_POST['unique_id']." ORDER BY ID_MEASURE
       DESC LIMIT 1";
  $res = $con->query($sql);
  $row = $res->fetch_row();
  if($row[0]>0){
    $c_pw_pro = ($row[0]*$voltage)."W ";
    $c_pw_con = "0W ";
  } else if ($row[0]<0){
    $c_pw_con = ($row[0]*$voltage*-1)."W ";
    $c_pw_pro = "0W ";
  } else {
    $c_pw_pro = "0W ";
    $c_pw_con = "0W ";
  }
```

The SQL query, uses the keyword **SELECT** to show the value of the column **VALUE**, **FROM** the table **MEASUREMENTS**, **WHERE** the field **ID_SENSOR** match with the **unique_id** send by parameter, the results are **ORDER BY** the column **ID_MEASURE**, from the last inserted value to the first(**DESC**), with a **LIMIT** of **1** result.

The result of this query is the last data entry in the table. If the value is negative it will be multiplied by the voltage and -1, this is the current power consumption. If positive it will be multiplied by the voltage and assigned to the current power production.

Total Power Production:

```
$sql = 'SELECT SUM(VALUE)
    FROM `MEASUREMENTS`
    WHERE `ID_SENSOR`='.$_POST['unique_id'].'
    AND DATE_TIME>(SELECT DATE_TIME FROM LOGS WHERE ID_MODULE='.$_POST['unique_id'].' ORDER BY DATE_TIME
        DESC LIMIT 1)';
$res = $con->query($sql);
$row = $res->fetch_row();
$tt_pw_pro = ($row[0]*$voltage)."W ";
```

The SQL query, uses the keyword **SELECT** to retrieve the addition (**SUM**) of the values from the column **VALUE**, **FROM** the table **MEASUREMENTS**, **WHERE** the field **ID_SENSOR** match with the **unique_id** send by parameter, AND the columns DATE_TIME is greater than the result from the **SELECT** value of the field **DATE_TIME FROM** table LOGS **WHERE** the **ID_SENSOR** match with the **unique_id** send by parameter, this result is **ORDER BY** the column **DATE_TIME**, from the last inserted value to the first(**DESC**), with a **LIMIT** of **1** result.

This query returns the addition of all data in column VALUE from the table measurements after the date and time of the last log entry. This way the total power production seen in the user interface corresponds to the uptime of the module/sensor.

Average power production:

```
// Average Power Production
$sql = 'SELECT AVG(VALUE)
    FROM `MEASUREMENTS`
    WHERE `ID_SENSOR`='.$_POST['unique_id'].'
    AND DATE_TIME>(SELECT DATE_TIME FROM LOGS WHERE ID_MODULE='.$_POST['unique_id'].' ORDER BY DATE_TIME
        DESC LIMIT 1)';
$res = $con->query($sql);
$row = $res->fetch_row();
$avg_pw_pro = ($row[0]*30)."W ";
```

This query is similar to the total power production, instead of the addition of the data in the column VALUES the average is made. This will give the average of values since the last data entry in the table logs.

Module status:

```
// Status
if($_SESSION['error_device']==1){
  $stat = "Connection Failed";
} else {

$sql = 'SELECT `STATUS`.NAME, `STATUS`.ID_STATUS
    FROM STATUS
    INNER JOIN LOGS ON `STATUS`.ID_STATUS = `LOGS`.ID_STATUS
    WHERE `ID_MODULE`='.$_POST['unique_id'].'
    ORDER BY `LOGS`.ID_LOG DESC
    LIMIT 1';

$res = $con->query($sql);
$row = $res->fetch_row();

if($row[0]==""){
  $stat = "Disconnected";
  $id_stat = 2;
} else {
  $stat = ($row[0]);
  $id_stat = $row[1];
}

if($row[1]!=3){
    $uptime = " --- ";
    $c_pw_pro = " --- ";
    $c_pw_con = " --- ";
    $tt_pw_pro = " --- ";
    $avg_pw_pro = " --- ";
} ...
```

The SQL query, uses the keyword **SELECT** to show the values of the columns **NAME** and **ID_STATUS**, **FROM** table **STATUS**, when joined (**INNER JOIN ON**) with table **LOGS** where the ID_STATUS match, and **WHERE** the field **ID_MODULE** match with the **unique_id** send by parameter, the results are **ORDER BY** the column **ID_LOG**, from the last inserted value to the first(**DESC**), with a **LIMIT** of **1** result.

This query returns the status name and id for the desired module, the returned values are assigned to the variable stat and id_stat for later use in this script. In case the status id is different then 3, the module will not be running, so no values are shown to the user.

Create the XML result:

```
  echo '<DATA>'.
       '<C_PW_PRO>'.$c_pw_pro.'</C_PW_PRO>'.
       '<C_PW_CON>'.$c_pw_con.'</C_PW_CON>'.
       '<TT_PW_PRO>'.$tt_pw_pro.'</TT_PW_PRO>'.
       '<AVG_PW_PRO>'.$avg_pw_pro.'</AVG_PW_PRO>'.
       '<UPTIME>'.$uptime.'</UPTIME>'.
       '<STAT id_stat="'.$id_stat.'">'.$stat.'</STAT>'.
       '<M1 id="'.$m_id[1].'" name="'.$m_name[1].'" status="'.$m_status[1].'" id_status="'.$m_id_status[1].'
           "></M1>'.
       '<M2 id="'.$m_id[2].'" name="'.$m_name[2].'" status="'.$m_status[2].'" id_status="'.$m_id_status[2].'
           "></M2>'.
       '<PRIV>'.$priv.'</PRIV>'.
       '<DEV_ERROR>'.$_SESSION['error_device'].'</DEV_ERROR>'.
      '</DATA>';
```

XML as HTML is a markup language, using elements defined by tags. JavaScript is able to translate XML files with built-in objects, this allows the return of a large scale of data at once, reducing the number of scripting needed and requests to the web server.

The construction of the XML file can be seen above, the main element is DATA, having several 'child nodes'. This nodes (like HTML) can have several attributes, for example the element M1 and M2 contain attributes that defines the modules connected to the port 1 and 2 of the energy hub.

3.3 Communication

savedata.php A background application running in the energy hub converts the measurement sent through PLC (UART) to a HTTP request **savedata.php?sensor_id=ID Sensor&value=Sensor Measurement&hub_port=Hu Port**.

Script parameters:

- sensor_id - Id of the sensor to where save data.
- value - the vlue to be saved
- hub_port - the connected port of the module
- op - status, change status of a module. ¡unique_id¿¡new status¿, this is added to the log table.

```
<?php
  function changeStatus($con){
    $date = getdate();
    $today = $date['year'].'-'.$date['mon'].'-'.$date['mday'].' '.$date['hours'].':'.$date['minutes'].':'.
         $date['seconds'];
    $sql = 'INSERT INTO `LOGS` (
        `ID_LOG` ,
        `ID_MODULE` ,
        `DATE_TIME` ,
        `ID_STATUS` ,
        `ID_USER` ,
        `ID_ERROR`
        )
        VALUES (
        NULL , \''.$_GET['id_module'].'\', \''.$today.'\', \''.$_GET['id_status'].'\', \''.$_SESSION['id_user
            '].'\', \'1\'
        )';
    if($con->query($sql)){
      echo "Success";
    }
    $sql = 'UPDATE `MODULES` SET `HUB_PORT` = '.$_GET['hub_port'].' WHERE `MODULES`.`UNIQUE_ID` ='.$_GET['
        id_module'];

    if($con->query($sql)){
      echo "Success";
    }
  }

  if(isset($_GET['op'])){

    switch ($_GET['op']){
      case "status": changeStatus($con); break;
    }

  } else {
    $date = getdate();

    $today = $date['year'].'-'.$date['mon'].'-'.$date['mday'].' '.$date['hours'].':'.$date['minutes'].':'.
        $date['seconds'];

    if(isset($_GET['sensor_id']) && isset($_GET['value']) && isset($_GET['hub_port'])){
      $sql= "INSERT INTO `MEASUREMENTS` (".
          "`ID_MEASURE` ,".
            "`ID_SENSOR` ,".
            "`DATE_TIME` ,".
            "`HUB_PORT` ,".
            "`VALUE`) ".
            "VALUES (NULL , '".$_GET['sensor_id']."', '".$today."', '".$_GET['hub_port']."', '".$_GET['value'
                ]."')";

      $con->query($sql); // Run the query in the MySQL server
      echo $sql;
    } else {
      echo 'Invalide parameters';
    }
  }

?>
```

This script saves the measurement retrieved from a sensor to the database. At first it established the connection to the MySQL server so SQL requests can be made. A PHP function returns an array with the current date and time, this is formatted into YYYY-M-D H:M:S, after it can be saved to the MEASUREMENTS table. The script collects all the parameters send through the URL encoding GET, a SQL code is generated and a request made to the MySQL server, the data is added to the MEASUREMENTS table.

In case of status changes, a parameter **op** is encode with the method GET and if it value is 'status' the function changeStatus() will be called. This function will update the HUB_PORT on the table MODULES and insert a new log to the table LOGS. The request for this functionality is **savedata.php?op=status&id_module=MODULE ID&id_status=STATUS TO CHANDE TO&hub_port=HUB_PORT**.

sendcmd.php The web interface is able to send commands to the energy hub and the modules using the **sendcmd.php?id_module=<module id>&cmd=<command to be send>**. The id_module tells the hub which module the command should be send. No verification is made of the send command by the web server or the energy hub unless the command is specifically send to the hub.

```

<?php

  // SQL request for the device page.
  require_once("includes/db_connect.php");

  if(isset($_GET['cmd']) && isset($_GET['id_module'])){

    $device_port = 5555;

    $sql= "SELECT IP ".
        "FROM `DEVICE` ".
        "ORDER BY ID_DEVICE DESC ".
        "LIMIT 1";

    $res = $con->query($sql); // Run the query in the MySQL server

    $row = $res->fetch_row();

    $device_ip = $row[0];

    if (!$socket=socket_create(AF_INET, SOCK_STREAM, SOL_TCP)){
      exit (socket_strerror(socket_last_error()));
    }

    if (!socket_connect($socket,$device_ip,$device_port)){
      exit (socket_strerror(socket_last_error()));
    }

    $str = $_GET['cmd'].';'.$_GET['id_module'].';'.$_GET['dir'];

    socket_write($socket,$str);

    $msg='';
    $c='';

    while(socket_recv($socket, $c, 256,0)){
        if($c != null) {
          $msg .= $c;
      }
    }

    echo $msg;

    socket_close($socket);

  } else {
    echo 'Command or Module Id not set';
  }

?>
```

At first the script will get the ip address of the energy hub, a SQL query retrieves the last IP address added to the table DEVICES. With the energy hub ip and a predefined port, a connection is created using a TCP socket for the communication between the energy hub and the web server. The parameters are collected from the encode URL and translated to a recognized format in the energy hub. The data is send to the energy hub and a answer of success or not is returned.

This script is called in background by a client side script, this will handle how to warning the user in case a problem occurs.

saveip.php

saveip.php script is called by the background application running on the energy hub, it saves the IP address given by DHCP, this is used for further communication between the web server and the energy hub.

```
  require_once("includes/db_connect.php");

  if(isset($_GET['ip'])){
    $sql= "INSERT INTO `DEVICE` (".
        "`ID_DEVICE` ,".
          "`IP`)".
          "VALUES (NULL , '".$_GET['ip']."')";

    $con->query($sql); // Run the query in the MySQL server

    // No feedback needed since the energy hub will not expect an answer.

  } else {
    echo 'IP not set';
  }
```

The ip address is send by the GET method (saveip.php?ip=127.0.0.1), this is how the data in encoded into a URL, being collected in the variable $_GET['ip']. A $sql variable string is created containing the SQL code to be run at the MySQL server.

db_connect.php

For the communication to the database a driver is used in PHP that provides an interface to the MySQL server. The PHP mysqli extension (MySQL improved) is used in this project, this is recommend for MySQL servers version 4.1.3 or later. This extension provides several benefits as a objective-oriented interface, support for multiple statements, embedded server support and more can be found in the MySQL documentation.

```
require_once("db_globals.php");

$con = new mysqli(DB_HOST,DB_USER,DB_PASS,DB_NAME); // Creates new mysql connection

if($con->connect_error){
  echo "Failed to connect to MySQL: (" . $con->connect_errno . " ) ". $con->connect_error;
}
else { echo "Connection established"; }
```

In this script an object is instantiated with a connection to the MySQL server.
$con = new mysqli<parameters >)
The parameters are included from the db_globals.php, setting the server host, user, password and database to be used.

db_globals.php

Using a script to define the parameters for the MySQL connection, All the scripts that need to use the global parameters for the connection to the MySQL server, should include db_globals.php as shown in the db_connect.php above.

```
// MySQL condifuration
DEFINE ('DB_USER','root');
DEFINE ('DB_PASS','pass');
DEFINE ('DB_HOST','localhost');
DEFINE ('DB_NAME','iEnergy');
```

The global parameters the connection to the MySQL server are define in this script.

- DB_USER - Database user name with read, write and execute permissions.
- DB_PASS - Password for the user
- DB_HOST - Hostname for the MySQL server, if running at the same host as the PHP server, localhost or 127.0.0.1 should be used.
- DB_NAME - Database name to connect to.

3.4 Login

The login script is called in background by the client side script popUp and handles the login and logout of the users.

login.php The server side script login, receives the credentials given by the user and compares with the ones stored in the database. In case of match, the privilege ID is retrieved and a user session is created with the user name and privilege id. In case of logout the session privilege is set to 0 and a message is returned to the client side script to handle.

```
if (isset($_POST['logout'])){ // If logout is set by PORT encoded URL request

  echo "logout";        // Return logout for the javascript
  $_SESSION['priv']=0;    // Set the priv to 0

} else if(isset($_POST['user']) && isset($_POST['pass'])){

  $sql = "SELECT ID_PRIV WHERE NAME='".$_POST['user']."'AND PASS='".$_POST['pass']."'"; // SQL statment
      contruction (No encryption in password)

  $res = $con->query($sql); // MySQL request

  if ($res->num_rows==1){ // See if any value is returned, if true assign privelege to session priv.

    $row = $res->fetch_row();
    $_SESSION['priv']=$row[0];
    echo "ok";
  } else {
    echo "Wrong credentials";
  }
}
```

Passwords are not encrypted in the table USERS since no user management web application was developed, the user names and passwords have to be inserted 'manually' into the database using phpMyAdmin. This system doesn't require high level of security since it cannot be accessible form the outside the university network.

3.5 Error Handling

The implementation of an error handling will keep the web interface more reliable, improve user experience and alert the administrators to a error situation. A error handling PHP script is called in background, this script will test the communication between the web server and the energy hub, and the communication to the MySQL server. As described in the analysis in this report, the user interface is blocked giving the message warning when the communication with the MySQL server is lost and a warning is given to the user when no communication with energy hub is found.

errorhandler.php The error handler PHP script is included in each web page before any other script, this script is included when a page is loaded and is recursively called in background to test the connections. The script gives a XML response, when called as background so the client side JavaScript can handle the real time warnings to the user. This method can be called as active error handle, since it acts in real time, usually error handling is implemented when is need to do some action, for example retrieve data from database. With this system the user is immediately blocked from doing any action until the situation is solved.

Since this script is included before everything else, the HTML, JavaScript and CSS layout is included in the file. Bellow several code blocks were extracted for and easier explanation, the full code can be seen using the application SeeIt and in the appendix of this report.

At first and most important the database connection have to be tested, as such a connection to the database is attempted. If the connection is successful the database error session variable will be 0. In case of connection failure the error variable will be changed to one and kept until the connection is re established, changing the session error variable to 2. This will tell the client side script that the situation was solved and in the next test the variable is changed to 0 again. This can be seen in the code bellow.

```

  if ($_SESSION['error_db']>1) {
    $_SESSION['error_db']=0;
  }

  $_SESSION['error_device']=0;

  // Test DB connection
  require_once("includes/db_globals.php");

  $con = mysql_connect(DB_HOST,DB_USER,DB_PASS,DB_NAME);

  if(!$con){
    $_SESSION['error_db']=1;
    $error = "Failed to connect to database: ". mysql_error();
  } else {
    if ($_SESSION['error_db']==1){
      $_SESSION['error_db']=2;
    } else {
      $_SESSION['error_db']=0;
    }
  }
```

If no error regarding the database connection is acknowledge, the ip address for the energy hub is retrieved and the connection with the energy hub can be established.

```
if($_SESSION['error_db']==0){

    $device_port = 5555;

    $con = new mysqli(DB_HOST,DB_USER,DB_PASS,DB_NAME);

    $sql= "SELECT IP ".
        "FROM `DEVICE` ".
        "ORDER BY ID_DEVICE DESC ".
        "LIMIT 1";

    $res = $con->query($sql);

    $row = $res->fetch_row();

    $device_ip = $row[0];
```

The SQL statement **SELECT IP FROM ‘DEVICE‘ ORDER BY ID_DEVICE DESC LIMIT 1** from the database the last ip address insert in the table. The keywords ORDER BY ID_DEVICE DESC will show the last data added and the keyword LIMIT 1, limits the number of values return to only one. The value is then fetched from the result and assigned to the variable device_ip for later use in the connection.

For the communication to the energy hub a socket have to be created and a connection established. In case of one of this steps don't work (Socket creation and Connection), a error session variable is set to 1, the client side application will catch this error and alert the user for such situation. The socket is closed after testing to leave the connection path open for new tests or commands to be send.

```
    if (!$socket=socket_create(AF_INET, SOCK_STREAM, SOL_TCP)){
      $_SESSION['error_device']=1;
    } else {
      $_SESSION['error_device']=0;
    }

    if (!socket_connect($socket,$device_ip,$device_port)){
      $_SESSION['error_device']=1;
    } else {
      $_SESSION['error_device']=0;
    }

    socket_write($socket,"a");

    socket_close($socket);

```

```
  }
```

The error handle client side (JavaScript) and server side (PHP) are developed at the same time, since the cooperation between both is essential for the proper operation of the error handling system.

This block of code is used only by the client side script, it make a request to the errorhandle.php with the encoded URL variable op=d, this block will return an XML response. The client side script then interpret the answer and reacts according to the errors.

```
  if(isset($_GET['op']) && $_GET['op']=='d'){

    echo '<ERROR>'.
         '<DB>'.$_SESSION['error_db'].'</DB>'.
        '<DEVICE>'.$_SESSION['error_device'].'</DEVICE>'.
        '</ERROR>';

  } else {
```

The else statement in this code indicates that, if the script was not called by the error.js, then it will include the necessary HTML, CSS and JavaScript to the file where it was included.

An example of this situation is found at the index.php, the first lines of this script starts the session and includes errorhandle.php.

```
<?php
  session_start();

  require_once("errorhandler.php");

  require_once("includes/db_connect.php");
?>
```

error.js This script catches and handle errors in the client side, it uses background requests to the PHP script errorhadle.php. By the XML response it acts according the situation. JavaScript is a dynamic language with a lot of potential, and this is used in detail in this script.Meaningful blocks will be extracted from the code and explained here.

Three functions are part of this script, being two of them for animation to improve the user experience, they are not documented in this section. The remaining function is the mains focus of the error handling system, is called makeTests().

This functions initialize the variables req for the URL request, db_error and dev_error that will be assigned with the values returned by the request. At first the browser is tested to define which HTTP request object to use, the main difference is between IE (version 5 and 6) form Microsoft and the rest ot the browsers, since IE (version 5 and 6) uses an ActiveX object to deal with background requests.

```
  var req;
  var db_error;
  var dev_error;

  if(window.XMLHttpRequest){
    req = new XMLHttpRequest();
  } else {
    req = ActiveXObject("Microsoft.XMLHTTP");
  }
```

The open and send methods from the XMLHttpRequest object, are used to send a request to the server. The 'open' method specifies the type of encoding used in the URL (GET or POST) and if the request is synchronous or not. The 'send' method sends the request to the server, parameters are needed in case of POST encoding method.

```
req.open("GET","/errorhandler.php?op=d",false);
req.overrideMimeType('text/xml');
req.send();

var result = req.responseXML;
```

The result variable is initialized and the returned XML data is assigned to it, this data is then extracted and used according to the situation. An example of retrieved XML data can be seen bellow:

```
  <ERROR>
    <DB>0</DB>
    <DEVICE>0</DEVICE>
  </ERROR>
```

Where 0 and 2 stands for no error and 1 for error occurrence.

3.6 User Experience

The user experience is improved by the client side scripts, HTTP requests are make as background using AJAX methods. The diagram above describe this method.

How AJAX Works:

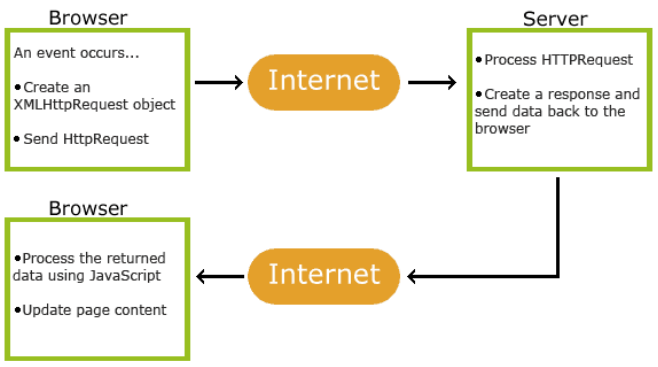

Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

Figure 3.4: http://www.w3schools.com/ajax/ajax_intro.asp

JavaScript is used to handle the returned values and change the layout according to them, this add great functionalities in user experience, since just parts of the page are dynamical changed with now need for refreshing the page with every action.

popUp.js The popUp script, handle the login actions to the web interface. When the locker is clicked, the window is blocked with a dark grey background and the credentials are asked from the user. The user is able to cancel this action, or input the credential to login into the web interface administration. When the user name and password are inserted and the ok button clicked, a background request is made to the script login.php. This will return an 'ok' if the credentials are correct, or an error message to be shown to the user if no match is found on the table USERS. The implementation of the login.php script is explained earlier in this project.

Function sendCred, sends a background request to the server, returns ok if success and error message otherwise. Parameters:

- type - 0 for logout, 1 for login
- str - path for the login.php script, normal path to this script would be the root directory of the server, in this way customized login scripts can be used.

control.js This script update the data displayed to the user without the need of refreshing the all page. It controls all the dynamics of the page such as send commands to the energy hub, get data from the database and displaying it to the users. The code for this script is extensive since it is the controller of the client side interface, as such the code can be seen in appendix or with the application SeeIt.

The script con_control.js in the root directory, is a striped down version of the control script, since its only functionality is to show the modules connected to the energy hub in the main page.

Function getData, makes a background request to the server, returning an XML file. This file is then handle and the correspondent fields on the user interface are updated. (See appendix for code)

Parameters:

- id - module id to retrieve stored values form the database.
- str - path for the getdata.php script, normal path to this script would be the modules/ directory in the server structure, this method allows the modules to use a different script.

Function sendCmd, send a background request to the server, returning the success or not of the command. The result is handle and the user is notified if the command was not successful.

Parameters:

- module_id - ID of the module to who the energy hub is going to transmit the command.
- cmd - command to be send to the module/energy hub.

4 Conclusion

This web interface is part of an green energy system, where modules where distributed by the teams in the class. Each team is responsible for the development of the web page related to them module and the physical equipment, making the cooperation between teams fundamental. The file structure and database structure was approved in team meeting, making the system dynamic and consistent for all modules.

In the analysis phase of this project, it was necessary to understand how can the database an the web interface be verified. As such a scaled down web interface with only two modules and the energy hub is implemented.

With this more realistic approach to the system, it was possible to verify problems regarding the common requirements for the communication between the modules and the energy hub. The database is able to handle several sensors connect to one module, but the energy hub have no knowledge of the sensors connected to the system. For a completely working system, deeper communication and protocol analysis have to be done.

A working high-fidelity prototype is created, ids are assigned to the modules being id:0 to the energy hub, id:1 to the battery and id:2 for the photovoltaic panels. Each module have just one sensor that measures the current (ampere), the id of the sensor is the same as the module id in this working prototype.

The end prototype can handle all functionalities expected for this system, the energy hub is able to send commands to the energy hub, add and retrieve data to and from the database. The error handling and the dynamic update of contents on the client side, improved the user experience for the control and navigation in the system.

5 References

6 Appendixes