

Eetu-Pekka Kouhia

**DEVELOPMENT OF AN ARDUINO-BASED EMBEDDED SYSTEM.
Case: Greenhouse monitoring**

**Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology
May 2016**

ABSTRACT

Unit Kokkola - Pietarsaari	Date May 2016	Author Eetu-Pekka Kouhia
Degree Program Information Technology		
Name of thesis DEVELOPMENT OF AN ARDUINO-BASED EMBEDDED SYSTEM. Case: Greenhouse monitoring		
Instructor Kauko Kolehmainen		Pages 26 + 28
Supervisor Kauko Kolehmainen		
<p>Today embedded systems are replacing various systems that used to be designed with a set of complex electronic circuits. Usually the heart of the embedded system is a microcontroller. One example of a microcontroller is Arduino. Arduino is an open source based prototyping platform used to sense and control physical devices.</p> <p>The purpose of this thesis was to create a microcontroller-based embedded system for monitoring greenhouse environmental variables. The user can control the greenhouse environment through a website. The website displays monitoring data to the user on a 24-hour line chart. Theory explains the use of Arduino microcontroller and how it is used in embedded systems. The practical part of the project introduces which hardware components are used and how they are used to build the system. Theory of the thesis is used as a base for designing the software layer. Software section of the practical part explains how the software was designed and implemented on top of the hardware layer.</p> <p>The finished project was able to meet the predetermined requirements. The design process conducted before assembling the system enabled the implementation to be easy and efficient. The system successfully reduced the power consumption, complexity and the cost of the monitoring project.</p>		

<p>Key words Arduino, embedded systems, sensors, webserver</p>

CONCEPT DEFINITIONS

- AJAX** Asynchronous JavaScript and XML. Used to XMLHttpRequest object to communicate with server-side scripts. AJAX's characteristic is asynchronous, which means it can do all of this without having to refresh the page.
- CSS** Cascading Style Sheets. Separate file, which defines style of HTML elements and how they are to be displayed on screen. The style definitions are saved in external .css files.
- EEPROM** Electronically Erasable PROM. Latest version of ROM memory types. Memory can be erased electronically and rewritten hundred thousand times. Commonly used to store settings on microcontroller included devices.
- HTTP** Hypertext Transfer Protocol. Protocol used to handle connection between web-client and server. Data transferred via protocol does not restrict to HTML documents.
- HTML** Hypertext Markup Language. Standard markup language used to create web pages. HTML code is rendered by the web browsers to visible web pages.
- IDE** Integrated Development Environment. Software environment, which provides tools for programmers, meant for software development.
- IP address** Internet Protocol Address. Unique address identifier that is given to device connected to Internet.
- I²C** Inter-integrated Circuit. Protocol intended to allow multiple "slave" digital integrated circuits to communicate with one or more "master" chips. Each I²C bus consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal.
- LAN** Local Area Network. A group of devices on a small geographic area that share a common communications line or wireless link.
- RELAY** Electromechanical switch. Switch operated with electromagnetic field. Used to isolate high voltage from logical level lower voltage.

- RTC** Real-Time Clock. Integrated circuit keeping track of time while microcontroller is turned off.
- SRAM** Static Random Access Memory. Static RAM-memory is type of semiconductor memory. Random Access Memory is a volatile memory type. This means that the data is eventually lost when the memory is not powered.
- SD** Secure Digital. Non-volatile memory card format developed by the SD Card Association. Compact and small SD-card is commonly used to store data in portable electronic devices.
- SPI** Serial Peripheral Interface. SPI is a synchronous serial communication interface. It is mainly used in embedded systems for short distance communication.
- XML** Extensible Markup Language. Designed to carry data, which is self-describing or self-defining, structure of the data is embedded with the data.

ABSTRACT

CONCEPT DEFINITIONS

CONTENTS

1 INTRODUCTION	1
2 EMBEDDED SYSTEM	2
2.1 Designing Embedded System	3
2.2 Embedded System Architecture	3
2.3 Implementation	5
2.4 Testing	6
3 ARDUINO	8
3.1 Arduino IDE	9
3.2 Arduino Due	10
3.3 Arduino Shields	11
3.4 Sensors	12
4 CASE: GREENHOUSE MONITORING SYSTEM	13
4.1 Requirements	13
4.2 Design	13
4.2.1 Hardware	14
4.2.2 Software	17
4.2.3 Application	20
4.3 Testing and Implementation	21
5 CONCLUSION	24

REFERENCES

APPENDICES

GRAPHS

GRAPH 1. Embedded systems in a car (adapted from Vizcayno 2015).	2
GRAPH 2. Model of the high-level embedded system architecture (adapted from Noergaard 2005).	3
GRAPH 3. Design and Development Lifecycle Model (adapted from Noergaard 2005).	4
GRAPH 4. CAD software for circuit simulation (adapted from FTD Automation Pvt. Ltd 2012).....	5
GRAPH 5. Testing model matrix (adapted from Noergaard 2005).....	8
GRAPH 6. Arduino IDE	9
GRAPH 7. Arduino Due microcontroller (Arduino 2011c)	10
GRAPH 8. Ethernet Shield on top of Arduino Due- microcontroller.....	11
GRAPH 9. DHT22, Digital moisture and temperature sensor (adapted from Robotshop 2016)	12
GRAPH 10. Block Diagram of Greenhouse Control System.	15
GRAPH 11. Soil moisture sensor (adapted from Frueh 2012)	15
GRAPH 12. Moisture sensor circuit diagram (Fritzing 2013).....	16
GRAPH 13. Wiring diagram of the hardware (Fritzing 2013)	16
GRAPH 14. Arduino software flowchart (adapted from Llemos 2015).....	18
GRAPH 15. Arduino IDE with serial monitor.....	19

GRAPH 16. XML response to transfer data between the webserver and the client..... 19
GRAPH 17. Greenhouse user interface website20
GRAPH 18. Frizing circuit diagram.21
GRAPH 19. Electrical connection diagram from DHT22 datasheet22
GRAPH 20. Code snippet of serial printing used for testing.....23

TABLES

TABLE 1. Testing model matrix7

1 INTRODUCTION

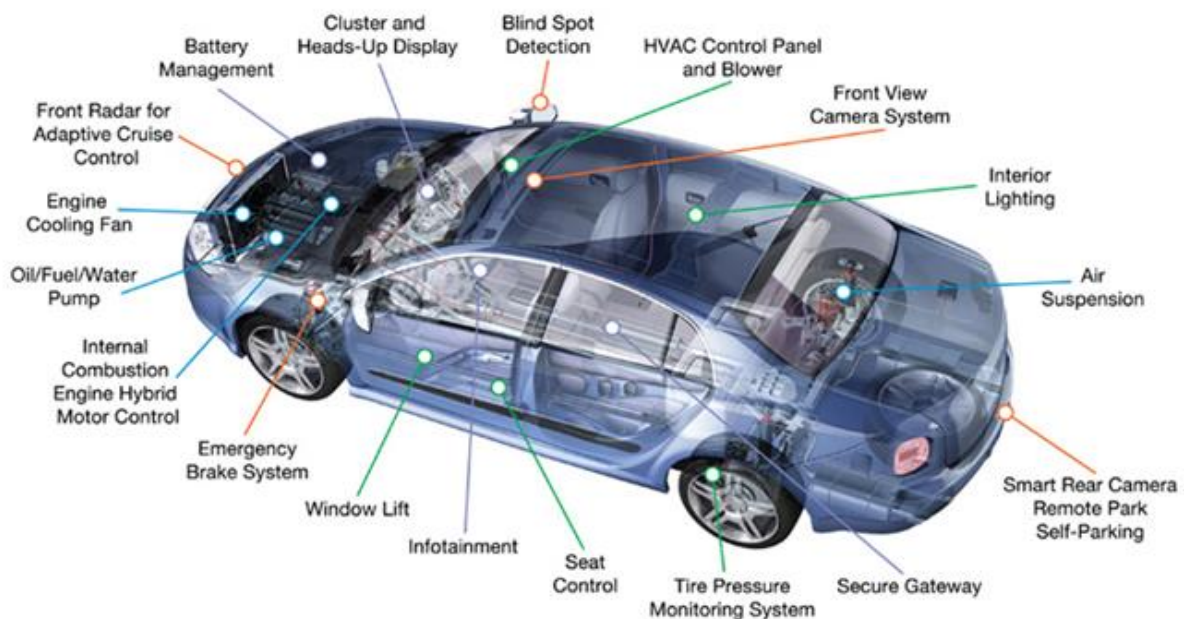
In 2008 the number of devices on the internet exceeded the number of people on the internet. It is estimated in 2020 there would be over 50 billion devices connected. Internet of Things (IOT) is starting to support the process connecting real-world to the Internet. Sensors and microprocessors are recording and transmitting data to the Internet. Rapidly increasing Internet-connected sensors means that new classes of technical capabilities and applications are being created. Constant monitoring is deepening the understanding of the internal and external worlds encountered by humans. High-frequency data processing is developing how humans adapt to the different kinds of data flows enabled by the IOT.

In this thesis, microcontroller-based embedded system is designed to monitor greenhouse environmental variables. In addition to monitoring temperature and moisture, the user can control greenhouse environment through relays. The system was designed using Arduino Due microcontroller and its development environment. Arduino webserver monitoring system was programmed using the C programming language. The sensor data is read and processed by Arduino and it is displayed to the user through the web interface. Website programming is designed with AJAX, HTML and CSS languages. The main aspects of designing the system were the simple usability and the low cost of manufacturing. User interface is designed to be used by people who have no prior computer knowledge.

Theoretical background in chapter 2 describes the methods used behind creating embedded systems. The whole design process is divided to four sections: design model, architecture model, implementation and testing. System design theory is applied when designing the practical part. Theory describes the use of the Arduino microcontroller and how it is utilized in the embedded systems. Practical part of the project is divided into two parts: Hardware and Software. Practical part describes manufacturing of the greenhouse monitoring system. The wiring diagram is explained such a way that someone without practical experience can replicate the process of creating the system. The source code of the user interface and the system are published in appendices.

2 EMBEDDED SYSTEM

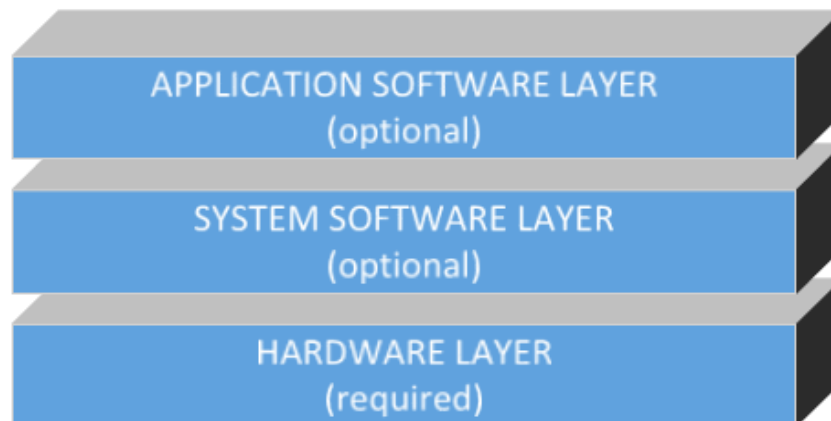
An embedded system is an applied computer system that is built to control a range of functions. Because of rapidly evolving technology the meaning of embedded systems is a vastly fluctuating definition. Advancing technology causes decrease in the cost of manufacturing and allows implementation of various hardware and software components to embedded systems. Embedded system is dedicated to a specific task. Systems normally consists of inputs, outputs and a small processing unit. Most of the devices used in our everyday life are some kind of embedded systems. Devices like mobile phones, watches, and elevators are all embedded systems. Most of the embedded systems are reactive systems, which means that the information received by the system is constantly processed and the system acts based on the information. The information changes according the interaction system and the environment. The example of a reactive embedded system is the car-braking system. Car brakes need to react instantly to a user input. System needs to react in a split second to prevent collision. (Karvinen & Karvinen 2009, 8-11.) Graph 1 shows are different embedded systems inside of a car.



GRAPH 1. Embedded systems in a car (adapted from Vizcayno 2015).

2.1 Embedded System Architecture

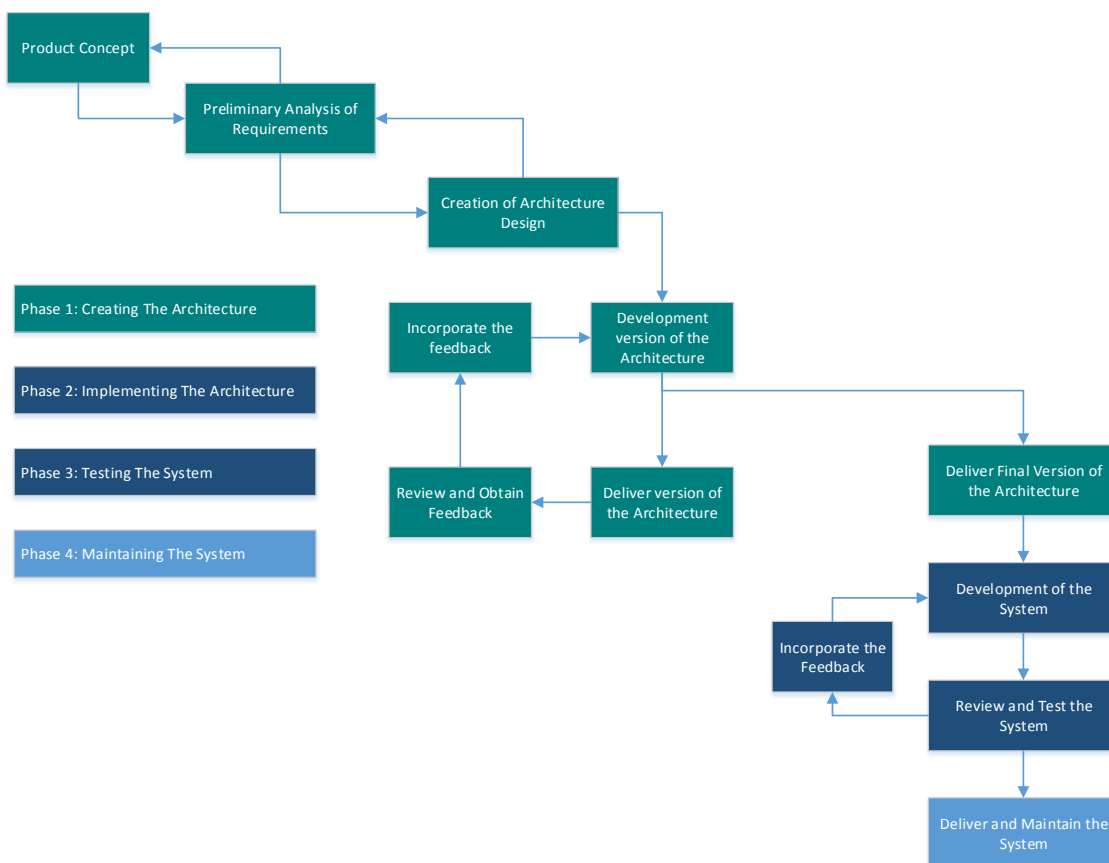
Embedded system architecture is a generalization of the system. Architecture does not show detailed implementation information such as software source code or hardware circuit design. The hardware and software components in an embedded system are presented as part of composition of interacting elements. Elements are representations of hardware and software, leaving only behavioral and inter-relationship information. A structure is one possible representation of the architecture. A structure is a snapshot of the system's hardware and software at design time or at run-time, given a particular environment and a given set of elements. An embedded systems architecture is used to resolve challenges early in a project. Without defining or knowing the internal level of implementation the architecture is the first tool used to analyze the system. The architecture can be used as a high-level blueprint defining the infrastructure of a design. The example of high-level architectural model presented in Graph 2. (Noergaard 2005.)



GRAPH 2. Model of the high-level embedded system architecture (adapted from Noergaard 2005).

2.2 Designing an Embedded System

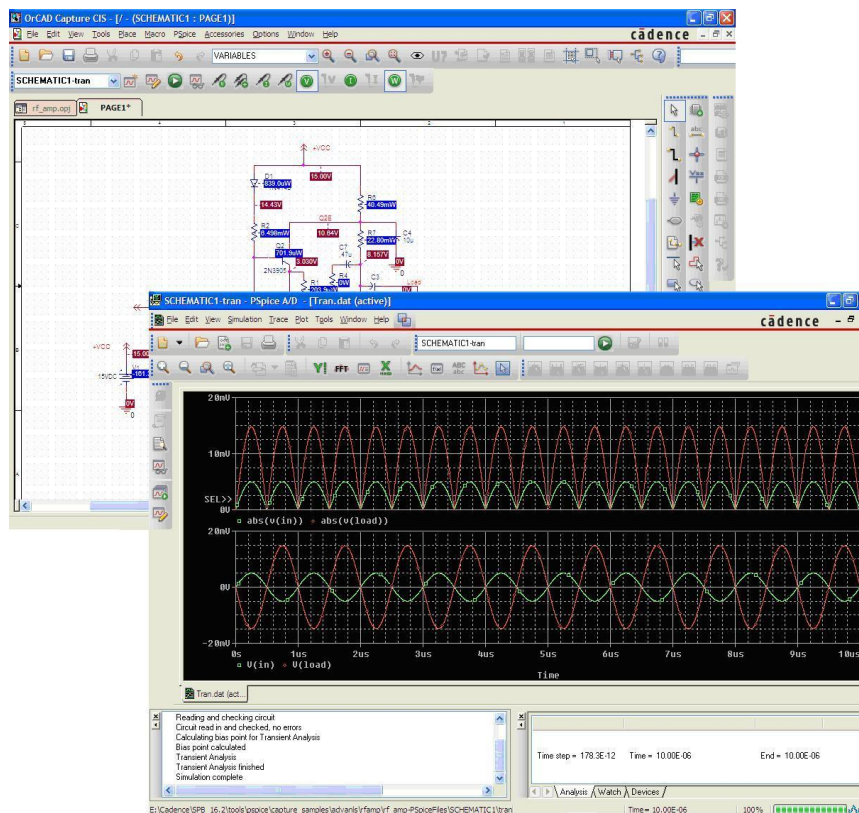
When designing an embedded system multiple different models can be used to approach the design. The four cornerstones of embedded system modelling are big-bang, code-and-fix, waterfall and spiral model. Most of the many models used in system design are based on single model, but combination of the cornerstone models can be sometimes applied. In big-bang model, no planning is executed before developing the system. The code-and-fix model requirements are defined but no processes are prepared before the start of development. The waterfall model uses strict process for developing a system in steps, each part of the system is developed step by step. The spiral model is a similar model to waterfall model. The development process is concluded systematically and between steps feedback is obtained and incorporated back to the process. Between step feedback and systematic progress model is shown in Graph 3. In the Graph 3 waterfall and spiral models have been combined into one system design model. (Noergaard 2005.)



GRAPH 3. Design and Development Lifecycle Model (adapted from Noergaard 2005).

2.3 Implementation

Implementing the design is one of the final phases of embedded system design. Traditionally, the design and implementation of control systems are often separated, which causes the development of embedded systems to be highly time consuming and costly. Having accurate design of architecture and system model helps to save money and time in the implementation phase. There are several tools built to ease the implementation of the system. The implementation and development process of the embedded system's hardware and software layer is made possible with development tools. One of the tools used on the hardware side is Computer-Aided Design (CAD). CAD is used to simulate circuits at the electrical level. In order to study a circuit's behavior before the final circuit and software is implemented simulation is used to test the hardware. Screen capture of the software is shown in Graph 4. Integrated Development Environment (IDE) is used to aid the implementation of the software side in embedded systems. More information about Integrated Development Environment is found in chapter 3 under Arduino IDE. After implementation, functional testing selects tests that assess how well the implementation meets the requirements of the product. (Noergaard 2005.)



GRAPH 4. CAD software for circuit simulation (adapted from FTD Automation Pvt. Ltd 2012).

2.4 Testing

The goals of testing is to assure the quality of a system. The tester is trying to determine if the system is operating according to its design. In other words, the tester is trying to determine if the error also known as a bug is found from the system. Testing can be also used to track whether bugs have been fixed. (Noergaard 2005.) Noergaard describes in the quotation what bugs are and how they behave in a system.

Under testing, bugs usually stem from either the system not adhering to the architectural specifications— i.e., behaving in a way it shouldn't according to documentation, not behaving in a way it should according to the documentation, behaving in a way not mentioned in documentation— or the inability to test the system. (Noergaard 2005, 563).

The types of bugs encountered in testing depend on the type of testing performed to the system. There are four generally used models to test the system: static black box testing, static white box testing, dynamic black box testing, or dynamic white box testing. Table 1 shows techniques used for testing. Black box testing means that a tester has no access to schematics or source code. Black box testing is based on general product information given to the tester. White box testing also known as clear box or glass box testing is where the tester has access to the source code and the schematics of the system. Static testing is used when the system is not running, whereas dynamic testing is applied when the system is running. (Noergaard 2005.)

TABLE 1. Testing model matrix (adapted from Noergaard 2005).

	Black Box Testing	White Box Testing
Static Testing	<p>Testing the product specifications by:</p> <ol style="list-style-type: none"> 1. Looking for high-level fundamental problems, oversights, omissions (i.e., pretending to be customer, research existing guidelines/standards, review and test similar software, etc.). 2. Low-level specification testing by insuring completeness, accuracy, preciseness, consistency, relevance, feasibility, etc. 	<p>Process of methodically reviewing hardware and code for bugs without executing it.</p>
Dynamic Testing	<p>Requires definition of what software and hardware does, includes:</p> <ul style="list-style-type: none"> • <i>data testing</i>, which is checking info of user inputs and outputs • <i>boundary condition testing</i>, which is testing situations at edge of planned operational limits of software • <i>internal boundary testing</i>, which is testing powers-of-two, ASCII table • <i>input testing</i>, which is testing null, invalid data • <i>state testing</i>, which is testing modes and transitions between modes software is in with state variables <p>i.e., race conditions, repetition testing (main reason is to discover memory leaks), stress (starving software = low memory, slow CPU, slow network), load (feed software = connect many peripherals, process large amount of data, web server have many clients accessing it, etc.), and so on.</p>	<p>Testing running system while looking at code, schematics, etc.</p> <p>Directly testing low-level and high-level based on detailed operational knowledge, accessing variables and memory dumps. Looking for data reference errors, data declaration errors, computation errors, comparison errors, control flow errors, subroutine parameter errors, I/O errors, etc.</p>

3 ARDUINO

Arduino is an open source tool for developing computers that can sense and control more of the physical world than desktop computer. It is an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. The software is written in C or C++ programming language. The Arduino development board is an implementation of wiring, a similar physical computing platform, which is based on the processing multimedia programming environment. (Arduino 2011a.)

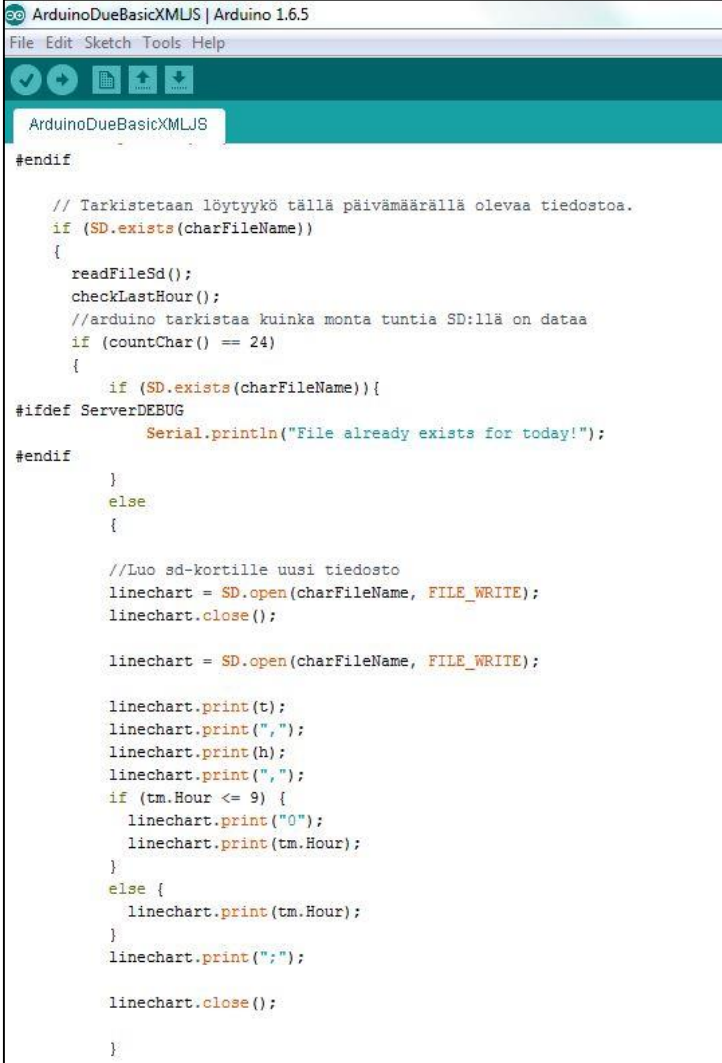
This single chip microcontroller has a microprocessor, which comes from a company called Atmel. The chip is known as an AVR. The AVR chip is running at only 16 MHz with an 8-bit core, and has a very limited amount of available memory, with 32 kilobytes of storage and 2 kilobytes of random access memory. Basic model of Arduino is shown in Graph 5. Arduino setup build around Atmel microprocessor causes it to be easy and popular to be used in all different kinds of DIY projects. (Evans 2011, 2-3; Banzi 2011, 17-18.)



GRAPH 5. Arduino Uno microcontroller (Arduino 2011b).

3.1 Arduino IDE

Arduino IDE is programming environment that allows the user to draft different kind of programs and load them into the Arduino microcontroller. Arduino uses user-friendly programming language, which is based on programming language called Processing. After the user has written his code, IDE compiles and translates the code to the assembler language. After translating the code, the IDE uploads the program to the Arduino microcontroller. Arduino IDE has a built-in code parser that will check the user written code before sending it to the Arduino. IDE software includes the set of different kind of programs that are ready to be tested on the device. After testing the program it can be uploaded to the Arduino by USB cable that vary in different models (Banzi 2011, 20-21). Graph 6 shows a screen capture of java-based Arduino IDE.



```

ArduinoDueBasicXMLJS | Arduino 1.6.5
File Edit Sketch Tools Help

ArduinoDueBasicXMLJS

#endif

// Tarkistetaan löytyykö tällä päivämäärällä olevaa tiedostoa.
if (SD.exists(charFileName))
{
  readFileSd();
  checkLastHour();
  //arduino tarkistaa kuinka monta tuntia SD:llä on dataa
  if (countChar() == 24)
  {
    if (SD.exists(charFileName)){
#ifdef ServerDEBUG
      Serial.println("File already exists for today!");
#endif
    }
    else
    {
      //Luo sd-kortille uusi tiedosto
      linechart = SD.open(charFileName, FILE_WRITE);
      linechart.close();

      linechart = SD.open(charFileName, FILE_WRITE);

      linechart.print(t);
      linechart.print(",");
      linechart.print(h);
      linechart.print(",");
      if (tm.Hour <= 9) {
        linechart.print("0");
        linechart.print(tm.Hour);
      }
      else {
        linechart.print(tm.Hour);
      }
      linechart.print(";");

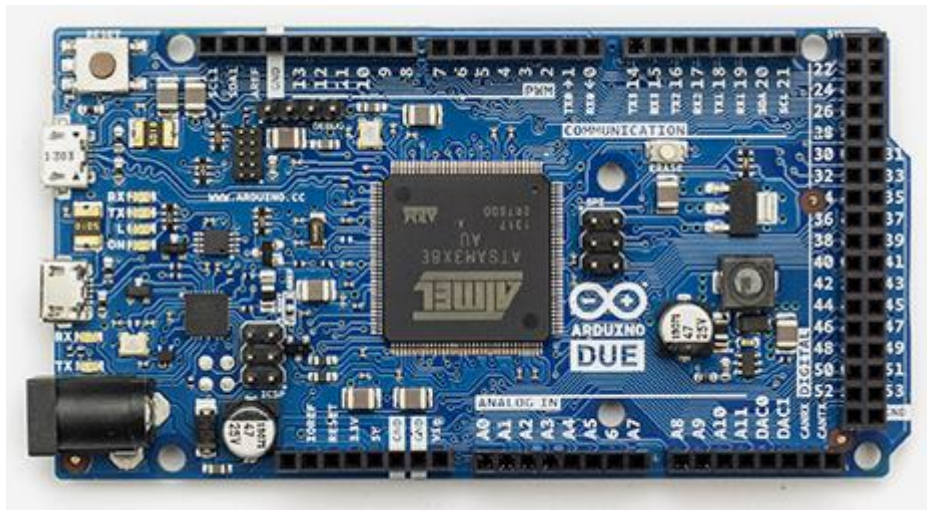
      linechart.close();
    }
  }
}

```

GRAPH 6. Arduino IDE

3.2 Arduino Due

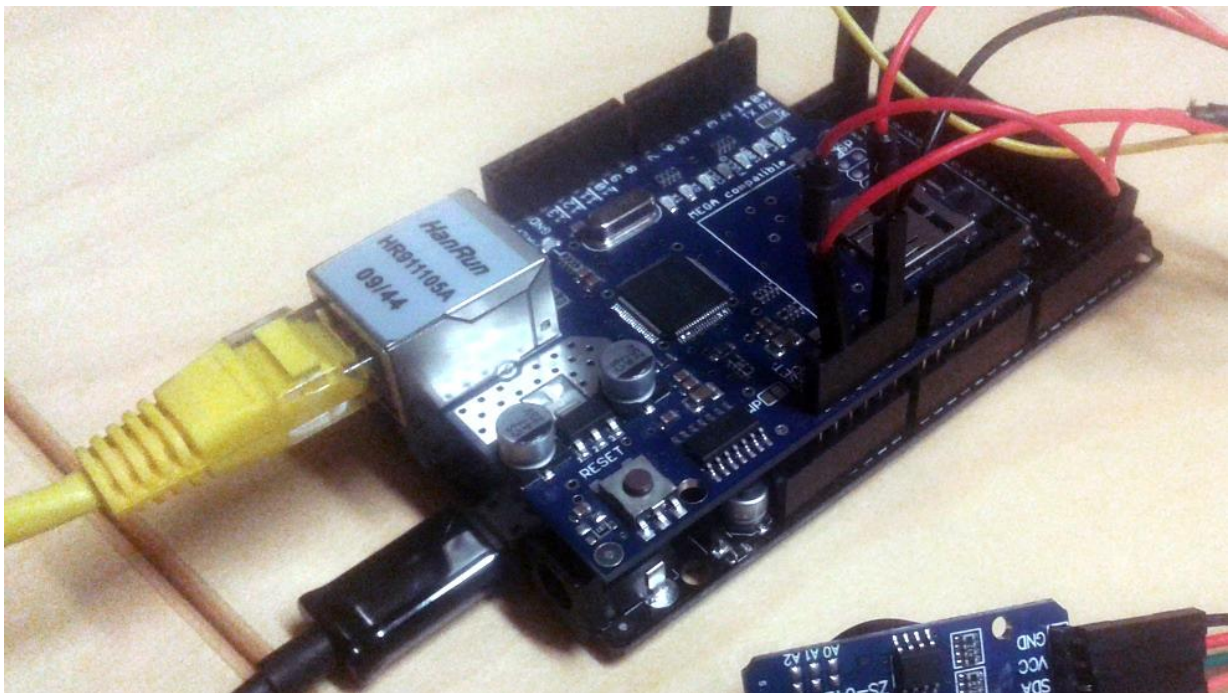
Arduino Due is the Arduino Microcontroller family's first development board based on the Atmel SAM3X8E ARM Cortex-M3 CPU that is shown in GRAPH 7. It has 54 digital input/output pins, 12 analog inputs, an 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button. Arduino Due has extended memory capabilities with 512kb of FLASH memory and 96kb of SRAM. The difference to other Arduino family boards is that the logical level voltage is 3.3v, the most of the Arduino boards run 5v on logical level. Arduino Due is the extended version of the Arduino family and it has all the basic functionalities of an Arduino. The microcontroller does not lack its usability because it has good compatibility with different module boards (shields). (Arduino 2011c.)



GRAPH 7. Arduino Due microcontroller (Arduino 2011c).

3.3 Arduino Shields

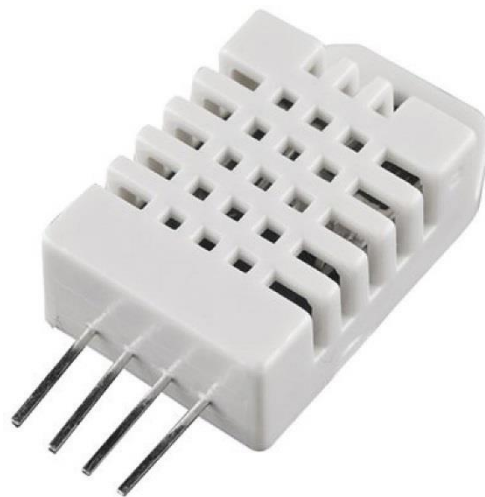
Shields are boards that can be stacked on top of the Arduino circuit board extending its capabilities. The picture of Arduino Ethernet shield presented in Graph 8. The different shields follow the same philosophy as the original toolkit: they are easy to mount, and cheap to produce. (Arduino, 2011d). Arduino Shields are designed to improve the versatility of the simple board. Almost every model of Arduino is compatible with shields designed to it. Shields do not only improve Arduino by giving it more connected sensors or circuits. They also contain code libraries made for the specific usage of the shield. Most common reason to buy shield for Arduino is that the project requires more input or output devices, which default port amount cannot provide. After the community have started developing different shields by themselves, Arduino manufacturers have started to embed shields directly into Arduino circuit boards. Easy install and removal of the shield gives opportunity to use Arduino in all different type of projects. (Banzi 2011.)



GRAPH 8. Ethernet Shield on top of Arduino Due microcontroller.

3.4 Sensors

The purpose of a sensor is to respond an input physical property and to convert it into an electrical signal that is compatible with electronic circuits (Fraden 2010, 2). Sensors are electronic devices that measure a physical quality such as light or temperature and convert it to a voltage. Example of digital temperature and moisture sensor is presented in Graph 9. There are two types of sensors: digital and analog. Digital sensor output varies between one and zero, which translates to sensors voltage range. Analog sensor can output any value between its voltage ranges. Its voltage output changes according to the reading from the sensor. Digital sensor output is ON (1) often 5v, or OFF (0), 0v. Analog sensor is used to measure precise numerical information like temperature or speed. Analog sensors can output almost an infinite range of values. Sensors are used to expand the capabilities of the Arduino. Sensor output is connected to input pin of Arduino and the data is converted to digital form. Some sensors have analog to digital converter embedded to the sensor so the data is outputted as digital data. Those sensors which don't have onboard analog to digital converter, data is sent analog to Arduino which then uses its onboard converter to convert data to digital. After data is processed to digital form, it can be processed on the microcontroller. (Karvinen & Karvinen, 2014.)



GRAPH 9. DHT22, Digital moisture and temperature sensor (adapted from Robotshop 2016).

4 CASE: GREENHOUSE MONITORING SYSTEM

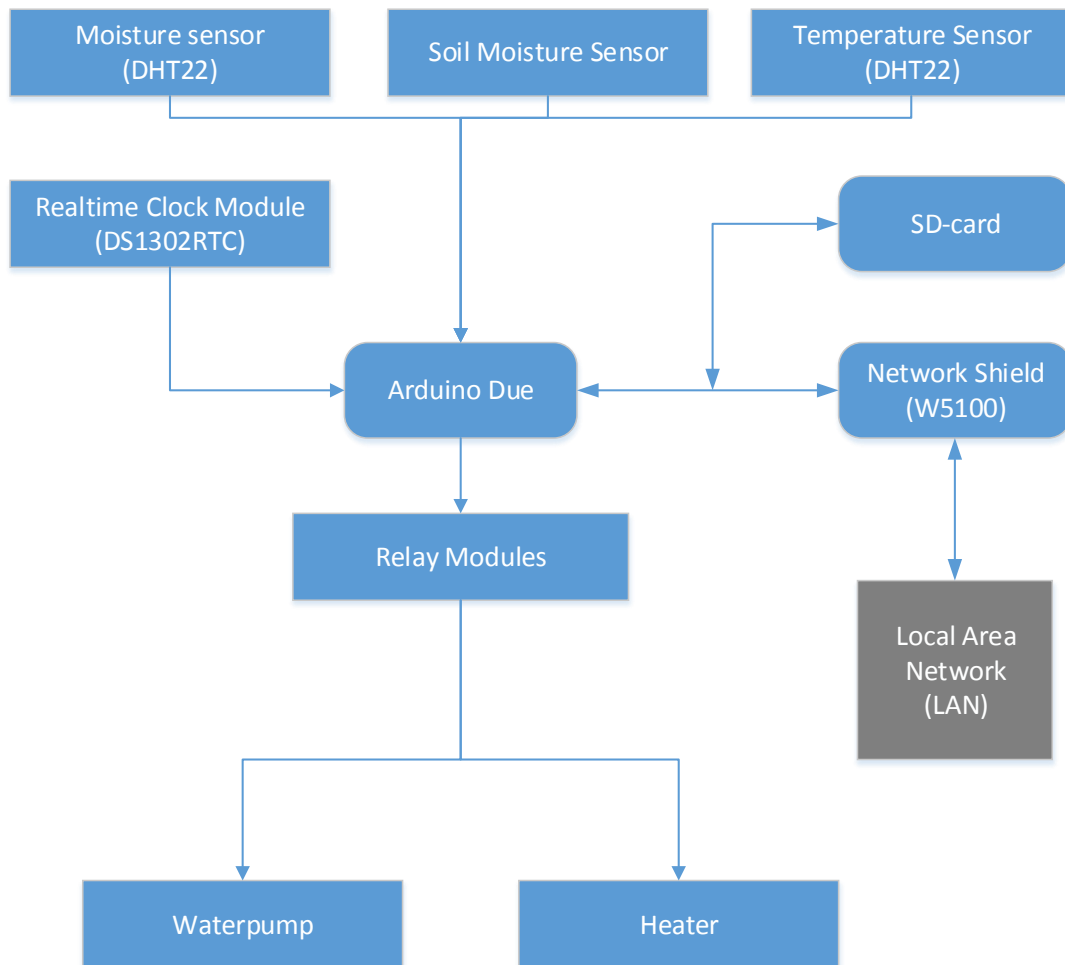
Important factors for the quality and productivity of plant growth are temperature, humidity, light and the level of the carbon dioxide. Monitoring of these environmental variables gives better understanding on how efficiently plants are growing and how to achieve maximal plant growth. The optimal greenhouse climate adjustment can enable us to improve productivity and to achieve remarkable energy savings - especially during the winter in northern countries. (Aarons Creek Farms 2012.) Difference between consumer and corporate level greenhouse monitoring system is that the accuracy of sensing environment is on a small area. Consumer level greenhouses are smaller so total cost of the system can be kept low. It used to be in the past that greenhouses had one cabled measurement point in the middle to measure information from the greenhouse automation system. Modern greenhouses are larger and more adjustable. Lights, ventilation, heating and other support systems can be more precisely controlled, which requires increased amount of sensors and better accuracy of locations. Increased number of measurement points should not dramatically increase the automation system cost. (Timmerman & Kamp 2003.)

4.1 Requirements

Before beginning to design the monitoring system for the greenhouse, certain requirements were set. The system is needed to be easy to use and the user could remotely monitor environmental changes inside the greenhouse. Sensor data required to be collected and stored for showing long period changes in the environment variables. Hardware requirements were set so that the cost of the system would be low as possible. To narrow down the cost of the system only three environment variables were chosen to be tracked: air humidity, soil moisture and temperature.

4.2 Design

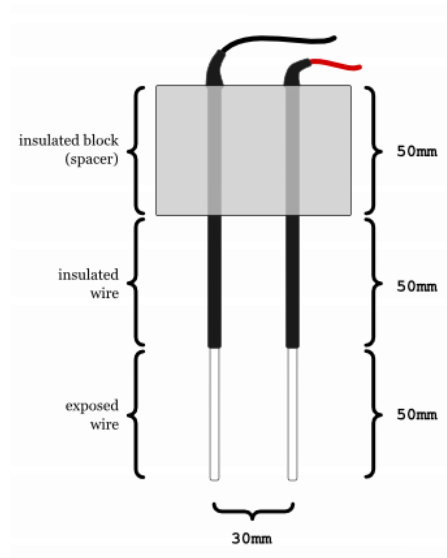
Embedded system design is divided into three layers: Hardware, Software and Application layer. See chapter 2.2. The architecture model of the system is shown in Graph 3. Hardware layer consists of electrical specifications of the design. Layer describes wiring of sensors, shield, RTC and Relays to Arduino Due. Software layer has the programming design of the system. Software describes functions to control relays and technique used to read sensor data. Application layer introduces the design of web-based user interface and the way data is transferred between software and the application layer.



GRAPH 10. Block Diagram of Greenhouse Control System.

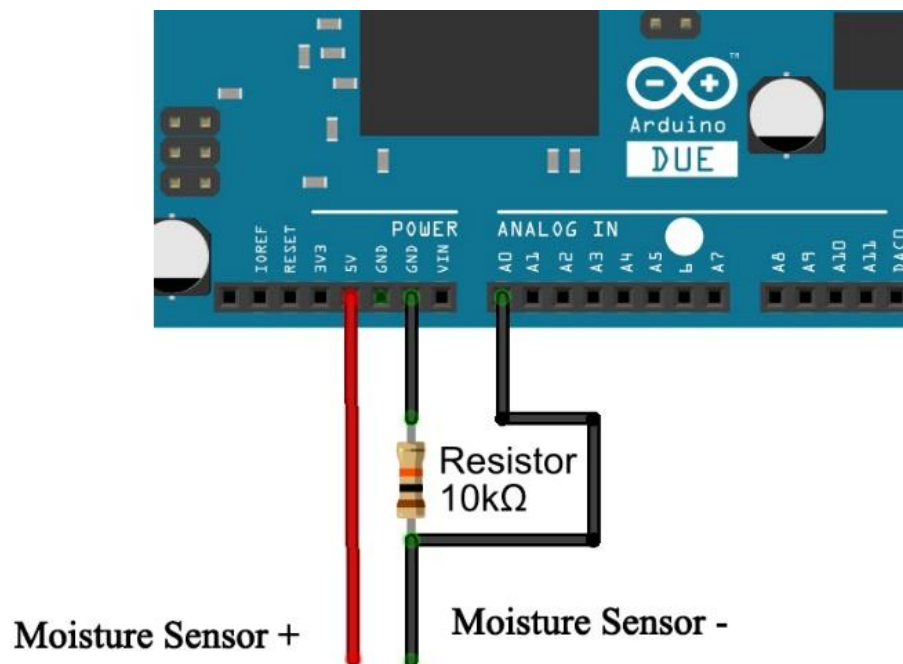
4.2.1 Hardware

Arduino Due microcontroller is the heart of the greenhouse monitoring system. Due provides enough processing power and memory to run the webserver and it can read multiple sensors simultaneously. To add network connectivity to the project, network shield W5100 is added on top of the main board. Arduino network shield enables website hosting to local area network (LAN). W5100 network shield included SD card slot, which was used for long term data storage. Network shield is visible on top of Arduino in Graph 8. Real-time clock module is introduced to the system to keep up time and date, even on power loss. The real time clock module DS1302 is connected to Due through I²C bus. RTC module's data line (SDA) is connected to pin 20 and clock line (SCL) is connected to pin 21 on the Arduino. Digital humidity and temperature sensor DHT22 is connected to digital pin 12 of the microcontroller.



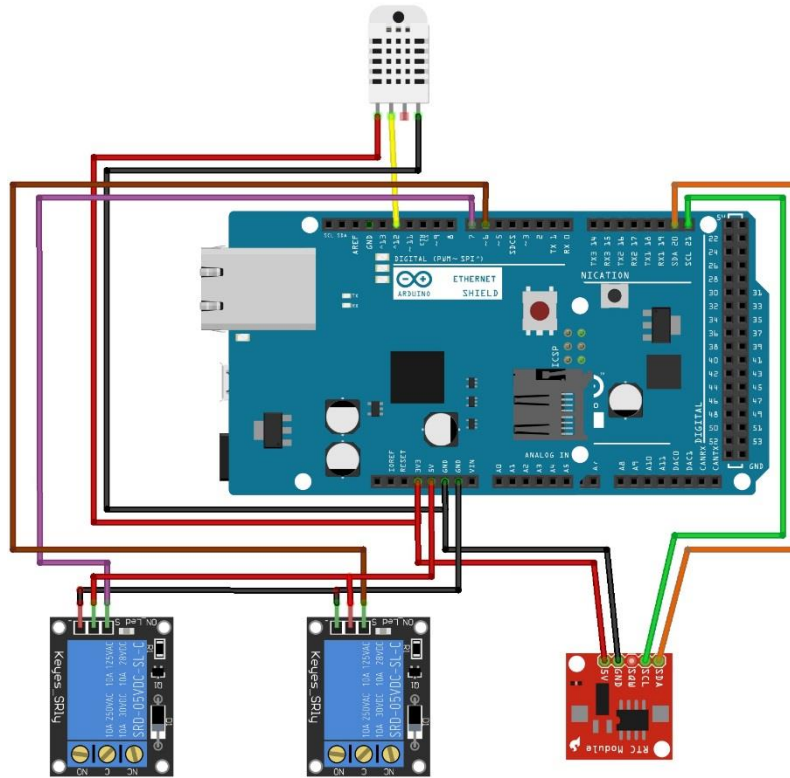
GRAPH 11. Soil moisture sensor (adapted from Frueh 2012).

Two soil moisture sensors shown in Graph 11 are connected to pins A0 and A1. Current is driven from Arduino to one of the poles in the soil moisture sensor (Graph 11). The second pole on the sensor Graph 11 is connected to Arduino analog pin. Analog pin is used to sense amount of conductivity of the soil. Moisture in a ground increases the conductivity of the soil. Soil moisture sensor connection circuit is shown in Graph 12.



GRAPH 12. Moisture sensor circuit diagram (Fritzing 2013).

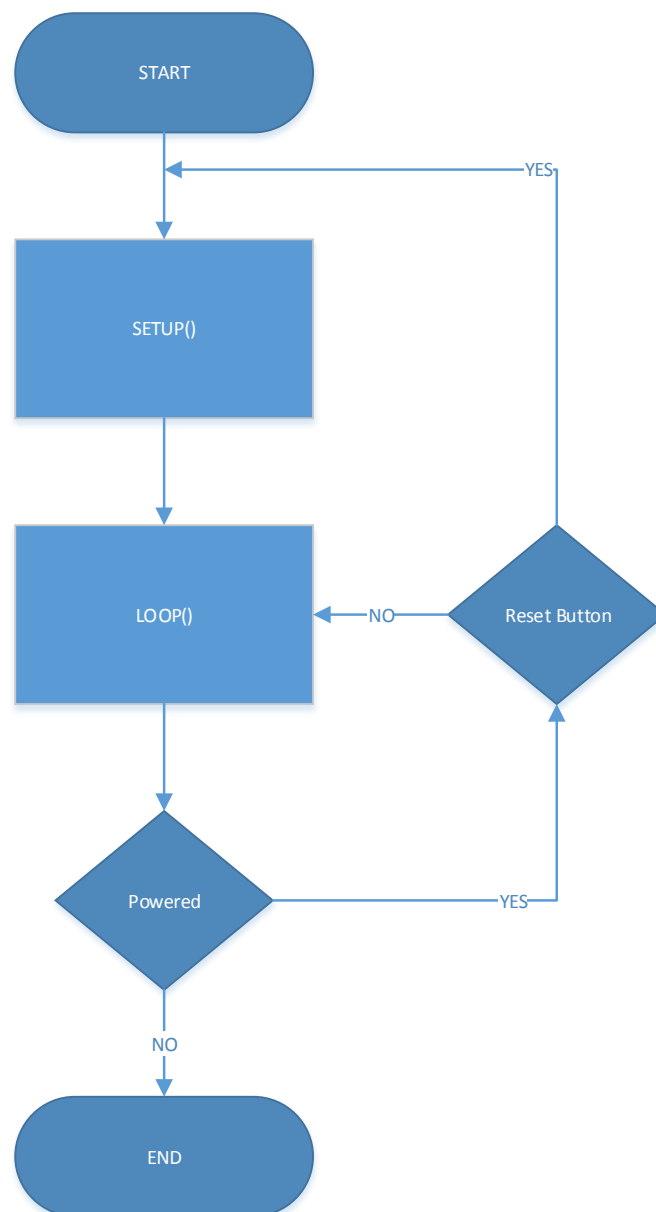
Relay modules are required to separate high current and high voltage devices from logical level devices. The water pump and heater are high voltage and high current devices which are controlled through relay modules. Modules are connected to digital pins 6 and 7. Block diagram (Graph 10.) displays the connection between Arduino Due, sensors and modules. Graphical version of the circuit diagram is shown in Graph 13.



GRAPH 13. Wiring diagram of the hardware (Fritzing 2013).

4.2.2 Software

In Arduino programming there are two main functions. Main functions are `setup()` and `loop()`. `Setup()` function is only operated once when device is booted up, it is mostly used to setup initiation settings. `Loop()` is ran after the `setup()` function has finished, `loop()` function will run repeatedly until power off or reset button is pushed (GRAPH 14). Arduino programming is supported by wide amount of libraries. Large amount of open-source libraries are available from Arduino community. `Setup()` function flowchart describes the setup process of the system (APPENDIX 1).



GRAPH 14. Arduino software flowchart (adapted from Llemos 2015).

Programming of the software was first started from the sensors and the real time clock module. Arduino IDE provided libraries to help reading data from DHT22 and RTC modules. Soil moisture sensor (Graph 11.) data was read directly from analog pins. Data from sensors and RTC was printed out to IDE's serial monitor for testing purposes (Graph 15). One of the requirements of the system was that there would be long-term data saved for charting purposes. Arduino provides a SD card library, which was used to create a function sdCardDatalog () (APPENDIX 4/2). The function saves sensor data and the time to the SD card on the W5100 network shield. To provide user remotely monitor their greenhouse through webpage, webserver needed to be established. Webserver libraries were created for Arduino but they did not meet the requirements of the system. Better web script support was needed to the project, so new webserver was designed to fit precisely the system requirements. Live charting was designed to display data for the user on the website. Without JavaScript support on the webserver, internet connection would have been needed. The new webserver enables the system to be used offline in local area network without internet connection.

The screenshot shows the Arduino IDE interface with the 'ArduinoDueBasicXMLJS' sketch loaded. The serial monitor on the right displays the following output:

```

Starting DHT-sensor
Starting SD..ok
Ready
Time: 16:26 Date: 20.10.2015
Humidity: 35.30 % Temperature: 21.30 *C
sdCardDatalog()Datalog
20102015.txt
charFileName
Success! Read
File exists
Success! Read
END
Inside timer Datalog
Time: 16:27 Date: 20.10.2015
Humidity: 35.70 % Temperature: 21.30 *C

Client request #1: GET / HTTP/1.1
file = /
file type =
method = GET
params =
protocol = HTTP/1.1
Home page SD file
filename format ok
SRAM = -404
file found..opened..send..closed
disconnected

Client request #2: GET /Chart.js HTTP/1.1
file = /CHART.JS
file type = JS
method = GET
params =
protocol = HTTP/1.1
SD file
filename format ok
SRAM = -404
file found..opened..send..closed
disconnected

Client request #3: GET /inputs&nocache=334815.5724816
XML UpdateSite
bad character
disconnected

```

GRAPH 15. Arduino IDE with serial monitor.

Asynchronous JavaScript and XML (AJAX) was used to transfer sensor data from webserver to a client. The client sends a request of XMLHttpRequest object to communicate with server-side scripts. The server responds to the request by sending the XML file containing sensor data in its HTTP header. The request can send as well as receive information in different formats, including XML, HTML, and even text files. The client request received by the server can be seen from Graph 15. and server respond is shown in client debugging mode in Graph 17. If the webserver cannot access the RTC module data, the system does not save the sensor data to SD card or send it to the web client. Logic of the webserver function is explained in the flowchart in Appendix 2. Full source-code of the system is shown in Appendix 4.



```

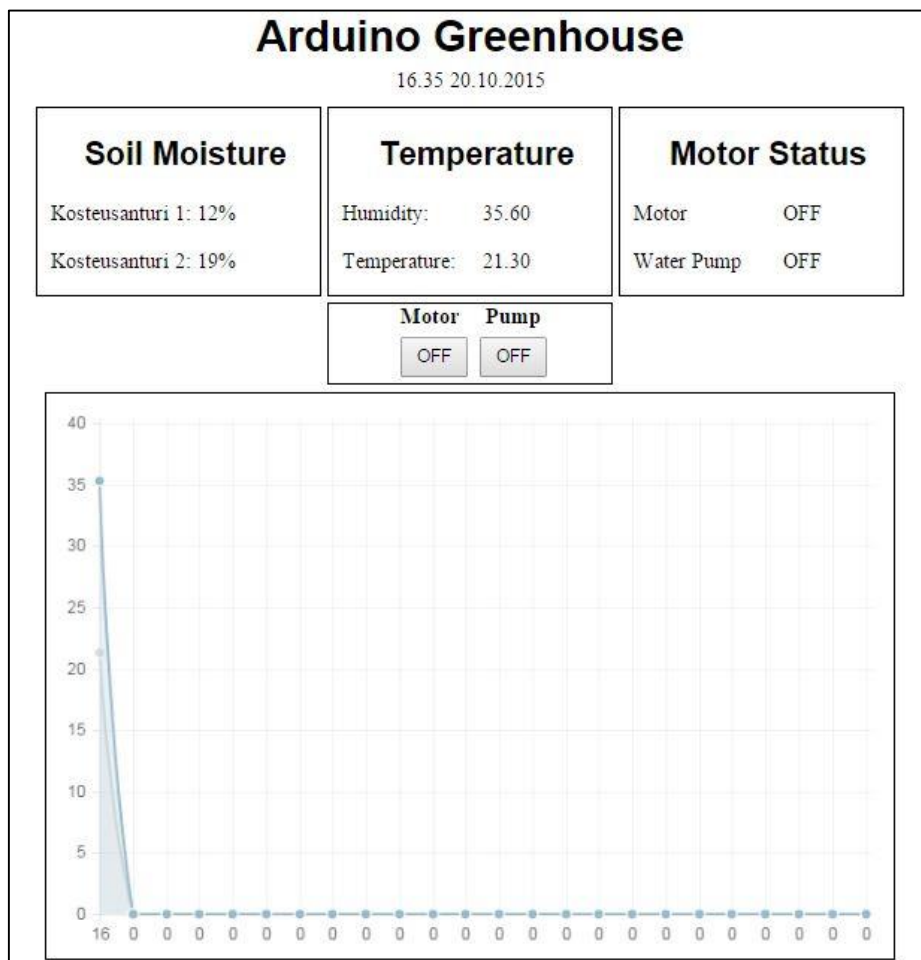
XMLHttpRequest {} (index):68
  onabort: null
  onerror: null
  onload: null
  onloadend: null
  onloadstart: null
  onprogress: null
  ▶ onreadystatechange: function ()
  ontimeout: null
  readyState: 4
  response: "<inputs><linechart>21.30,35.30,16;</linechart></inputs>"
  responseText: "<inputs><linechart>21.30,35.30,16;</linechart></inputs>"
  responseType: ""
  responseURL: "http://192.168.1.10/linechart&nocache=316682.99180455506"
  ▶ responseXML: document
  status: 200
  statusText: "OK"
  timeout: 0
  ▶ upload: XMLHttpRequestUpload

```

GRAPH 16. XML response to transfer data between the webserver and the client.

4.2.3 Application

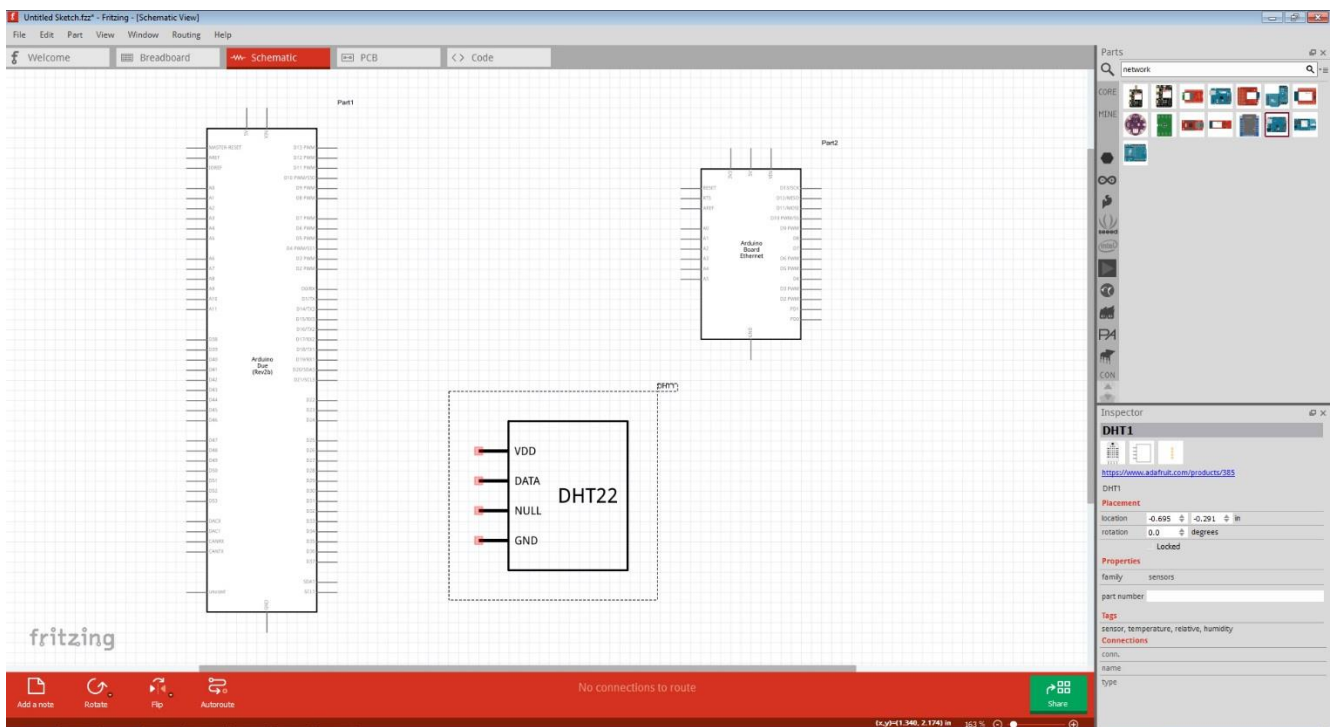
Application layer of the system is the user interface webpage. Web interface can be accessed by hard coded IP address. By default, the IP address is 192.168.1.10. On the webpage, the user can access relays and view the live data of the sensors. JavaScript library Chart.js is used to render 24-hour line chart data of temperature and humidity. Line chart data is stored on the SD card on the server. Before the data is sent to the client, the latest timecoded sensor data is verified to make sure duplicate values are not saved on the SD card. In occurrence of sudden power outage system will load latest data from the SD card and send it to be displayed. The user interface webpage accessed in LAN is shown in Graph 17. Real-time clock module time and date is displayed under the main heading. Time is updated every minute together with sensor data and line chart. If webserver does not respond to client data request line chart is shown empty. The full source code of the website is shown in Appendix 5.



GRAPH 17. Greenhouse user interface website.

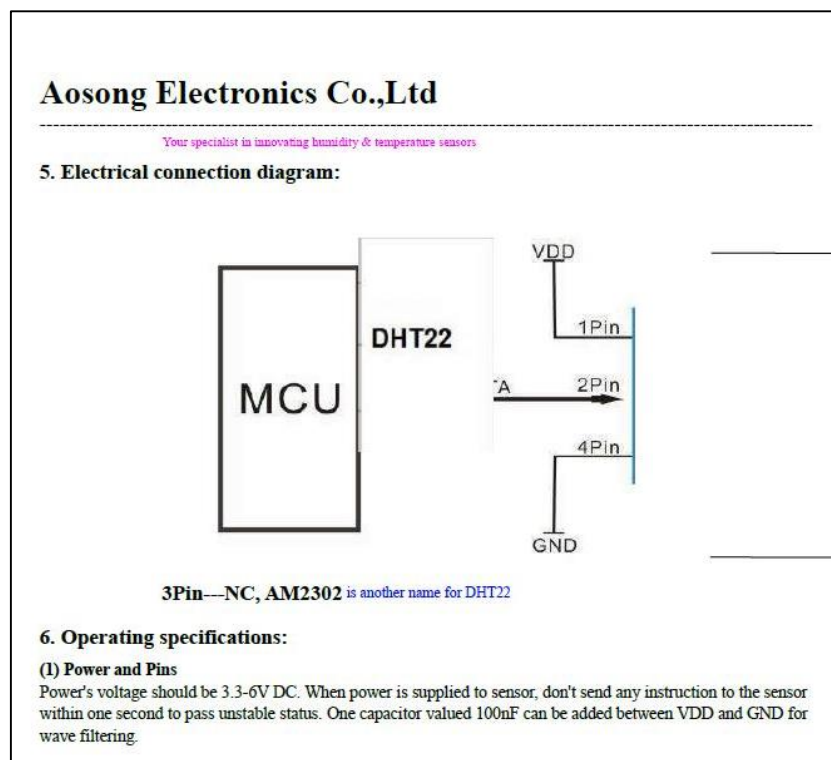
4.3 Implementation and Testing

Arduino provides tools to assist the implementation of a system. Arduino IDE enables serial communication with Arduino microcontroller through USB. Printing to serial console in the IDE helps to track function events between the hardware and the software layer. The serial console can be seen on the right side of Graph 15. Arduino community open source libraries assist the implementation of the sensors and network shield to the Arduino Due. In this implementation, libraries, which are used, are meant to simplify sensor data fetching and network connectivity. Libraries give access to live sensor data with single function. Another implementing tool used for the design is Fritzing. It is an open-source circuit design tool for prototyping. It is used to create wiring diagrams of the project. Graphical presentation of modules, sensors and microcontrollers give the designer free hands to design the prototype. With Fritzing it is easy to develop a product from schematic to prototype.



GRAPH 18. Frizing circuit diagram.

Testing of the system mainly focuses on the software side testing. During the building process of the monitoring system, dynamic testing was done constantly together while programming the system itself. Dynamic testing is discussed in previous chapter 2.4. Software testing of the system was broken into two different parts. In the first part, each subsystem of the software was tested for idle run without network load and then with multiple computers connecting to webserver causing network load. Second part was that each subsystem was put together to the whole system and was tested for network load by multiple computers connecting simultaneously to the device. Each subsystem consisted of one functionality of the system. For example, each sensor data fetching was divided and tested separately. Hardware side testing was mainly checking the wiring diagrams of the sensors. Manufacturers of sensors provide comprehensive datasheet of electronic properties and diagrams. Electrical connection diagram of DHT22 datasheet is described in Graph 19.



GRAPH 19. Electrical connection diagram from DHT22 datasheet.

After the sensor, webserver, SD card and real time clock functions were tested and diagnosed to be fully operational, software of the system was put together. Arduino being able to print serial data during development process helped debugging of the system. To test the data transfer between webserver and client some dummy variables were created to populate the line chart on the website. The data used to test the XML data transfer can be seen in APPENDIX 4/1 on line 44-46. To keep track of the whole system while testing some serial data was printed between each functionality of the software. Example of serial printing used during testing seen in Graph 20.

```
86
87 #ifdef ServerDEBUG
88     Serial.print(F("Starting SD.."));
89 #endif
90
91     if(!SD.begin(4)) {
92 #ifdef ServerDEBUG
93     Serial.println(F("failed"));
94 #endif
95     }
96     else {
97 #ifdef ServerDEBUG
98     Serial.println(F("ok"));
99     sdStatus = 1;
100 #endif
101     }
102
```

GRAPH 20. Code snippet of serial printing used for testing

5 CONCLUSION

A systematic approach in designing the microcontroller based system for measurement and control of the three essential parameters for plant growth, temperature, humidity and soil moisture, has been followed. The system has successfully overcome of the existing systems by reducing the power consumption, complexity and the cost at the same time providing a precise form of maintaining the environment. The results obtained from the measurement have shown that the system performance is quite reliable and accurate.

Arduino microcontrollers are constantly evolving development platform. Vastly advancing technology can easily bypass technology used in Arduino Due and make the technology outdated. Growing open-source community is constantly developing. More advanced software is programmed to work similarly with monitoring environment variables.

Environment variable monitoring DIY projects are common in open-source communities. Multiple greenhouse or household plant monitoring projects can be found online. The key factor that sets this greenhouse-monitoring project apart from other DIY monitoring systems is that the user can easily access the data through the web interface and the user can affect the environment inside the greenhouse through the interface. Usually web interfacing monitoring systems require multiple hardware to handle hosting services. To narrow down the cost of hosting and monitoring system was combined to one device.

For further research, the system could be designed to operate in a wireless environment. Wi-Fi technology would be considerable option to provide free access to user interface. Alternatively, even further taken option can be 3G or 4G cellular connectivity. Accessing the monitoring data regardless of distance to the location of the greenhouse would make monitoring more efficient and less time consuming. Data could be accessed via internet with the handheld device.

REFERENCES

- Aarons Creek Farms. 2012. Greenhouse Buying Guide. Available: <http://www.littlegreenhouse.com/guide.shtml>. Accessed: 26 April 2016.
- Arduino. 2011a. Introduction. Available: <http://www.arduino.cc/en/Guide/Introduction>. Accessed: 13 April 2016.
- Arduino. 2011b. Arduino Uno. Available: <http://arduino.cc/en/Main/arduinoBoardUno>. Accessed: 14 March 2016.
- Arduino. 2011c. Arduino Due. Available: <http://www.arduino.cc/en/Main/ArduinoBoardDue>. Accessed: 19 March 2016.
- Arduino. 2011d. Arduino Shields. Available: <http://arduino.cc/en/Main/ArduinoShields>. Accessed: 15 April 2016.
- Banzi, M. 2011. Arduino – Getting Started with Arduino. Maker Media, Inc.
- Banzi, M. 2012. Available: http://www.ted.com/talks/massimo_banzi_how_arduino_is_open_sourcing_imagination#. Accessed: 14 March 2016.
- Evans, B. 2011. Beginning Arduino Programming. 2-3. New York: Apress.
- Fraden, J. 2010. Handbook of Modern Sensors: Physics, Designs, and Applications. Pringer Science & Business Media.
- Fritzing. 2013. Fritzing: a tool for advancing electronic prototyping for designers. Available: <http://fritzing.org/home/>. Accessed: 26 March 2016.
- Frueh, A. 2012. Soil Moisture Sensor. Available: http://gardenbot.org/howTo/soilMoisture/how-to_moisture-sensor_big.png. Accessed: 21.4.2016.
- FTD Automation Pvt. Ltd. 2012. OrCAD PSpice A/D. Available: http://www.ftdautomation.com/upload/orcad_ospice/PSpice.jpg. Accessed: 24 March 2016.

Karvinen, T. & Karvinen, K. 2014. Getting Started with Sensors: Measure the World with Electronics, Arduino, and Raspberry Pi. Maker Media, Inc.

Karvinen, T. & Karvinen, K. 2009. Sulautetut. Helsinki: Readme.fi.

Llemos, J.A. 2015. How does an Arduino sketch work? Available: <http://japaalekhin.llemos.com/wp-content/uploads/2015/10/arduino-software.jpg>. Accessed: 26 March 2016.

Mäenpää, Y. 2011. Arduino – Perusteista hallintaan. Hämeenlinna: Robomaa.com.

Noergaard, T. 2005. Embedded Technology : Embedded Systems Architecture : A Comprehensive Guide for Engineers and Programmers. Burlington, MA, USA: Newnes. ProQuest ebrary.

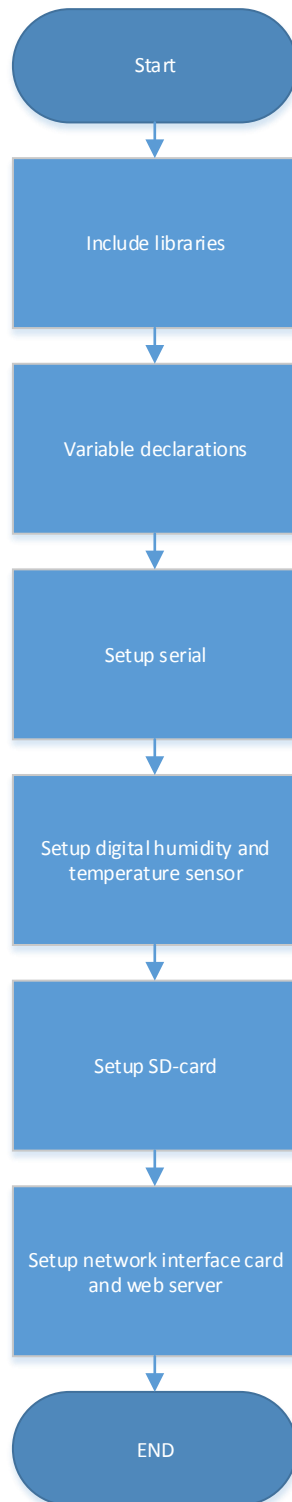
Robotshop. 2016. Humidity and Temperature Sensor - DHT22. Available: <http://www.robotshop.com/media/catalog/product/cache/1/image/800x800/9df78eab33525d08d6e5fb8d27136e95/h/u/humidity-temperature-sensor-dht22.jpg>. Accessed: 20 March 2016.

SparkFun Electronics – DHT22. 2010. Digital-output relative humidity & temperature sensor/module DHT22 (DHT22 also named as AM2302). Available: <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>. Accessed 29 March 2016.

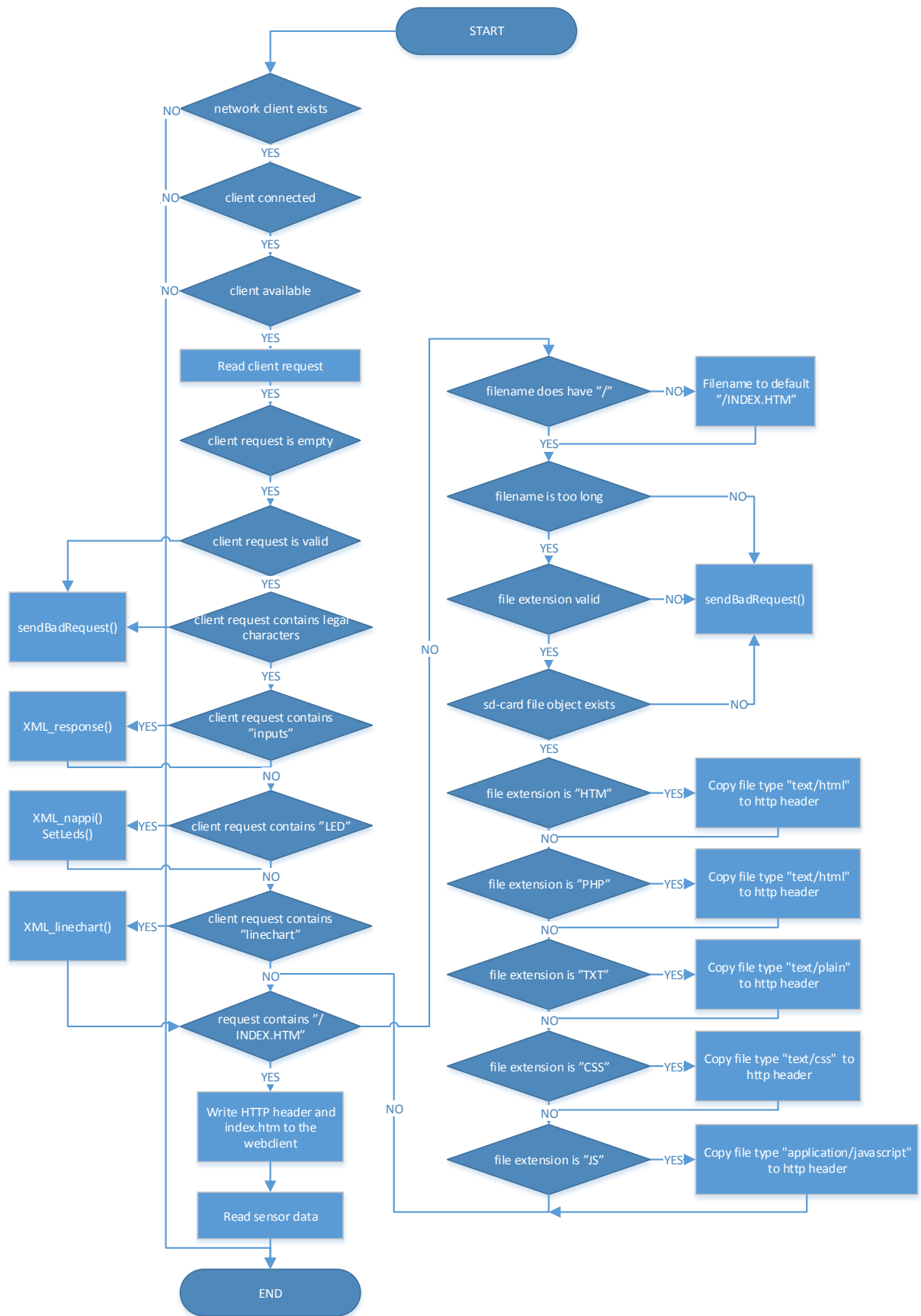
Timmerman, G. J. & Kamp, P. G. H. 2003. Computerized Environmental Control in Greenhouses, PTC.

Vizcayno, D. 2015. The Future of Technology, Privacy, Security and Risks (part 3 of 5). Available: <https://dcvizcayno.files.wordpress.com/2015/08/as1.png>. Accessed: 19 March 2016.

System setup() flowchart



CheckServer function flowchart



Function Declarations

Char StringContains()

Function is used to find if string contains specific string or set of characters.

By finding, certain string from http-request enables user to control motors through web interface.

Int countChar()

In the file saved on the SD-card every reading from sensor is saved and separated by semicolon.

Function is used to determine how many readings have been already saved on the SD-card by finding semicolons from string containing set of readings.

Void strtoupper(char* sBuf)

Characters in string are capitalized with this function.

Void checkLastHour()

Reading file and determining which was the last hour saved on the SD-card. In case of power loss running memory is erased. To determine which hour is already been saved to the SD-card it is needed to determine number of data saved for certain date.

Void checkServer()

Main function for updating website content. Function is loading website from SD-card and updating line chart graph from the sensors.

Void datalogTimer()

Timing sensor data saved on the SD-card every hour. Checking if the SD-card is enabled and is accessible.

Void loop()

One of two the main functions of Arduino. Function is called continuously while Arduino is powered.

Void myStuff()

Showing network socket status by reading serial console command 'r'.

Void readFileSd()

Reading SD-card data for further use in program.

Void sdCardDatalog()

Saving sensor data to SD-card.

Void sendBadRequest(Ethernet thisClient)

Checking for HTTP-request format to be correct.

Void sendFileNotFound(Ethernet thisClient)

Searching if the file exists on the SD-card.

Void serialSensors()

Displaying data to serial monitor on Arduino IDE. Used mainly for debugging.

Void setLEDs()

Used to control relay according website controls.

Void setup()

One of two the main functions of Arduino. Function is called every time at the start of powering Arduino board.

Void ShowSockStatus()

Printing socket status to serial monitor.

Void XML_linechart(Ethernet thisClient)

Generating XML string containing line chart values, what is send back to web client.

Void XML_nappi(Ethernet thisClient)

Generating XML string containing button status, which is send back to web client.

Void XML_response(Ethernet thisClient)

Generating XML string containing sensor values and time code, which is then send back to web client.

Arduino webserver code

```

1  /*
2  Web server sketch for IDE v1.0.5 and w5100/w5200
3  Original web server example by SurferTim
4  Last modified 6 June 2015
5  http://playground.arduino.cc/Code/WebServerST
6
7  Arduino Watering System:
8  Webserver hosting air moisture, soil moisture and temperature to a website on user
   defined address in local network.
9  Every hour reading is saved to a SD-card. Data from SD-card is read to 24-hour
   linechart which is then hosted on the website.
10 Every day new file is created and each file is then containing 24 values of soil
   moisture, air moisture and air temperature.
11
12 Written by Eetu-Pekka Kouhia
13 26.10.2015
14 */
15
16 //Additional Libraries written for certain sensors
17 #include <DHT.h>
18 #include <Wire.h>
19 #include <Time.h>
20 #include <DS1307RTC.h>
21 #include <SPI.h>
22 #include <Ethernet.h>
23 #include <SD.h>
24 #include <utility/w5100.h>
25 #include <utility/socket.h>
26
27 //Temperature & Humidity Sensor setup
28 #define DHTPIN 2 // Pin connected to Temperature & Humidity sensor
29 #define DHTTYPE DHT22 // DHT 22 (AM2302), Model of the Humidity and Temperature
   module
30
31 #define ServerDEBUG // Used for debugging the server, if defined serial print enabled
32
33 //Network settings
34 byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEC }; //Mac-address of the LAN-shield
35 IPAddress ip( 192,168,1,10 ); //Ip-address of the lan-shield
36 IPAddress gateway( 192,168,1,1 ); //Gateway of the lan-shield
37 IPAddress subnet( 255,255,255,0 ); //Subnet of the lan-shield
38
39 tmElements_t tm; //Declaring time object to store data read from realtime clockmodule
40
41 //Setting up temporary buffer to store data from SD-card
42 char temp_Chart_Buf[362];
43
44 //Pre-filled buffer to test XML-send and Chart.JS populating
45 //char temp_Chart_Buf[362] =
   "23.12,23.12,00;25.11,25.11,01;15.15,15.15,02;23.12,23.12,03;23.12,23.12,04;15.15,23.1
   2,05;23.12,23.12,06;25.11,23.12,07;15.15,23.12,08;23.12,23.12,09;25.11,23.12,10;15.15,
   23.12,11;23.12,23.12,12;25.11,23.12,13;15.15,23.12,14;23.12,23.12,15;25.11,23.12,16;15
   .15,23.12,17;23.12,23.12,18;25.11,23.12,19;15.15,23.12,20;23.12,23.12,21;25.11,23.12,2
   2;15.15,23.12,23;";
46
47 File linechart; //Declaring for File object for SD-card
   file.
48 char charFileName[] = "00000000.TXT"; //Declaring variable to store filename
   populated from date and year

```

```

49  int runOnce = 1; //Declaring variable to store function run
    counter
50  int sdStatus = 0; //Declaring variable to store SD-card read
    status
51  int lastHour = 99; //Declaring variable to check what was the
    previous hour, used to time linechart data save on Sd-card every hour.
52  int a1,b1,c1,a2,b2,c2,c; //Declaring variables to help populating
    CharFileName[]
53  int j = 0; //Declaring variable to count loop run
54  char requestBuffer[64]; //Declaring buffer to store GET request
55  int LED_state[2]; //Declaring array to store I/O status
56  int HourInt; //Declaring variable to store last hour
    data from SD-card
57  int loopCount = 0;
58  int timerCounter = 0;
59
60  const char legalChars[] = "ABCDEFGHJKLMNPQRSTUVWXYZ1234567890/._~";
61  unsigned int requestNumber = 0;
62  unsigned long connectTime[MAX_SOCKET_NUM];
63
64  //Declaring variables to store Temperature & Humidity sensor data
65  float h = 0;
66  float t = 0;
67
68  //Declaring use of the Temperature & Humidity sensor (library DHT)
69  DHT dht(DHTPIN, DHTTYPE,30);
70
71  //Declaring port where the webserver is hosted
72  EthernetServer server(80);
73
74  //Function running once at the beginning
75  void setup()
76  {
77      Serial.begin(9600);
78
79      // disable w5100 SPI while starting SD
80      digitalWrite(10,HIGH);
81      digitalWrite(3,HIGH);
82
83      Serial.println(F("Starting DHT-sensor"));
84      dht.begin();
85      delay(200);
86
87      #ifdef ServerDEBUG
88          Serial.print(F("Starting SD.."));
89      #endif
90
91      if(!SD.begin(4)) {
92          #ifdef ServerDEBUG
93              Serial.println(F("Failed"));
94          #endif
95          }
96      else {
97          #ifdef ServerDEBUG
98              Serial.println(F("ok"));
99              sdStatus = 1;
100         #endif
101     }
102

```

```

103 Ethernet.begin(mac, ip, gateway, gateway, subnet);
104
105 delay(2000);
106 server.begin();
107
108 unsigned long thisTime = millis();
109
110 for(int i=0;i<MAX_SOCK_NUM;i++) {
111     connectTime[i] = thisTime;
112 }
113
114 #ifdef ServerDEBUG
115     Serial.println(F("Ready"));
116 #endif
117 }
118
119 //Main loop running constantly
120 void loop()
121 {
122     checkServer();
123     myStuff();
124     datalogTimer();
125 }
126
127 void myStuff() {
128     if(Serial.available()) {
129         if(Serial.read() == 'r') ShowSockStatus();
130     }
131     checkSockStatus();
132 }
133
134 void datalogTimer(){
135     if (sdStatus){
136         if (tm.Hour != lastHour)
137         {
138             Serial.print("sdCardDatalog()");
139             sdCardDatalog();
140             lastHour = tm.Hour;
141             runOnce = 1;
142         }
143         else
144         {
145             if(runOnce)
146             {
147                 Serial.println("Inside timer Datalog");
148                 runOnce = 0;
149             }
150         }
151     }
152     else
153     {
154         if(runOnce)
155         {
156             Serial.println("Cannot Datalog SD failure");
157             runOnce = 0;
158         }
159     }
160 }
161

```



```

162 void checkServer() {
163     EthernetClient client = server.available();
164     if(client) {
165         boolean currentLineIsBlank = true;
166         boolean currentLineIsGet = true;
167         int tCount = 0;
168         char tBuf[128];
169         int r,t;
170         char *pch;
171         char methodBuffer[8];
172         char pageBuffer[64];
173         char paramBuffer[64];
174         char protocolBuffer[9];
175         char fileName[32];
176         char fileType[4];
177         int clientCount = 0;
178         requestNumber++;
179
180 #ifdef ServerDEBUG
181     Serial.print(F("\r\nClient request #"));
182     Serial.print(requestNumber);
183     Serial.print(F(": "));
184 #endif
185     // this controls the timeout
186     int loopCount = 0;
187
188     while (client.connected()) {
189         while(client.available()) {
190             // if packet, reset loopCount
191             // loopCount = 0;
192             char c = client.read();
193
194             if(currentLineIsGet && tCount < 63)
195             {
196                 tBuf[tCount] = c;
197                 tCount++;
198                 tBuf[tCount] = 0;
199             }
200
201             if (c == '\n' && currentLineIsBlank) {
202 #ifdef ServerDEBUG
203                 Serial.print(tBuf);
204 #endif
205                 while(client.available()) client.read();
206
207                 int scanCount = sscanf(tBuf,"%7s %47s
208 %8s",methodBuffer,requestBuffer,protocolBuffer);
209
210                 if(scanCount != 3) {
211 #ifdef ServerDEBUG
212                     Serial.println(F("bad request"));
213 #endif
214                     sendBadRequest(client);
215                     return;
216                 }
217                 else{
218                     char* pch = strtok(requestBuffer,"?");
219                     if(pch != NULL) {

```

```

220         strncpy(fileName,pch,31);
221         strncpy(tBuf,pch,31);
222
223         pch = strtok(NULL,"?");
224         if(pch != NULL) {
225             strcpy(paramBuffer,pch);
226         }
227         else paramBuffer[0] = 0;
228     }
229
230     if (StrContains(requestBuffer, "inputs")) {
231         XML_response(client);
232     }
233
234     if (StrContains(requestBuffer, "LED")) {
235         XML_nappi(client);
236         SetLEDs();
237     }
238
239     if (StrContains(requestBuffer, "linechart")) {
240         XML_linechart(client);
241     }
242
243     strtoupper(requestBuffer);
244     strtoupper(tBuf);
245
246     for(int x = 0; x < strlen(requestBuffer); x++) {
247         if(strchr(legalChars,requestBuffer[x]) == NULL) {
248             Serial.println(F("bad character"));
249             sendBadRequest(client);
250             return;
251         }
252     }
253 #ifdef ServerDEBUG
254     Serial.print(F("file = "));
255     Serial.println(requestBuffer);
256 #endif
257     pch = strtok(tBuf, ".");
258
259     if(pch != NULL) {
260         pch = strtok(NULL, ".");
261
262         if(pch != NULL) strncpy(fileType,pch,4);
263         else fileType[0] = 0;
264
265 #ifdef ServerDEBUG
266     Serial.print(F("file type = "));
267     Serial.println(fileType);
268 #endif
269     }
270
271 #ifdef ServerDEBUG
272     Serial.print(F("method = "));
273     Serial.println(methodBuffer);
274 #endif
275     if(strcmp(methodBuffer,"GET") != 0 && strcmp(methodBuffer,"HEAD") != 0) {
276         sendBadRequest(client);
277         return;
278     }

```

```

279
280 #ifdef ServerDEBUG
281     Serial.print(F("params = "));
282     Serial.println(paramBuffer);
283
284     Serial.print(F("protocol = "));
285     Serial.println(protocolBuffer);
286 #endif
287     if(strcmp(requestBuffer, "/INDEX.HTM") == 0) {
288 #ifdef ServerDEBUG
289     Serial.println(F("dynamic page"));
290 #endif
291     }
292     else {
293     if(strcmp(fileName, "/") == 0) {
294     strcpy(fileName, "/INDEX.HTM");
295     strcpy(fileType, "HTM");
296
297 #ifdef ServerDEBUG
298     Serial.print(F("Home page "));
299 #endif
300     }
301 #ifdef ServerDEBUG
302     Serial.println(F("SD file"));
303 #endif
304     if(strlen(fileName) > 30) {
305 #ifdef ServerDEBUG
306     Serial.println(F("filename too long"));
307 #endif
308     sendBadRequest(client);
309
310     return;
311     }
312     else if(strlen(fileType) > 3 || strlen(fileType) < 1) {
313
314 #ifdef ServerDEBUG
315     Serial.println(F("file type invalid size"));
316 #endif
317     sendBadRequest(client);
318     return;
319     }
320     else {
321 #ifdef ServerDEBUG
322     Serial.println(F("filename format ok"));
323 #endif
324     if(SD.exists(fileName)) {
325
326 #ifdef ServerDEBUG
327     Serial.print(F("file found.."));
328 #endif
329     File myFile = SD.open(fileName);
330
331     if(!myFile) {
332 #ifdef ServerDEBUG
333     Serial.println(F("open error"));
334 #endif
335     sendFileNotFound(client);
336     return;
337     }

```

```

338 #ifdef ServerDEBUG
339     Serial.print(F("opened.."));
340 #endif
341     strcpy_P(tBuf,PSTR("HTTP/1.0 200 OK\r\nContent-Type: "));
342
343     // send Content-Type
344     if(strcmp(fileType,"HTML") == 0) strcat_P(tBuf,PSTR("text/html"));
345     else if(strcmp(fileType,"PHP") == 0) strcat_P(tBuf,PSTR("text/html"));
346     else if(strcmp(fileType,"TXT") == 0)
347         strcat_P(tBuf,PSTR("text/plain"));
348     else if(strcmp(fileType,"CSS") == 0) strcat_P(tBuf,PSTR("text/css"));
349     else if(strcmp(fileType,"JS") == 0)
350         strcat_P(tBuf,PSTR("application/javascript"));
351     else strcat_P(tBuf,PSTR("text/plain"));
352
353     strcat_P(tBuf,PSTR("\r\nConnection: close\r\n\r\n"));
354     client.write(tBuf);
355
356     if(strcmp(methodBuffer,"GET") == 0) {
357 #ifdef ServerDEBUG
358     Serial.print(F("send.."));
359 #endif
360     while(myFile.available()) {
361         clientCount = myFile.read(tBuf,64);
362         client.write((byte*)tBuf,clientCount);
363     }
364     myFile.close();
365 #ifdef ServerDEBUG
366     Serial.println(F("closed"));
367 #endif
368     client.stop();
369 #ifdef ServerDEBUG
370     Serial.println(F("disconnected"));
371 #endif
372     return;
373 }
374 #ifdef ServerDEBUG
375     Serial.println(F("File not found"));
376 #endif
377     sendFileNotFound(client);
378     return;
379 }
380 }
381 }
382
383 #ifdef ServerDEBUG
384     Serial.println(F("Sending response"));
385 #endif
386     strcpy_P(tBuf,PSTR("HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n"));
387     client.write(tBuf);
388     client.stop();
389 }
390 }
391 else if (c == '\n') {
392     currentLineIsBlank = true;
393     currentLineIsGet = false;
394 }

```

```

395     else if (c != '\r') {
396         currentLineIsBlank = false;
397     }
398 }
399 loopCount++;
400
401 // if 1 second has passed since last packet
402 if(loopCount > 1000) {
403     // close connection
404     client.stop();
405 #ifdef ServerDEBUG
406     Serial.println("\r\nTimeout");
407 #endif
408 }
409 // delay 1ms for timeout timing
410 delay(1);
411 }
412 #ifdef ServerDEBUG
413     Serial.println(F("disconnected"));
414 #endif
415 }
416 #ifdef ServerDEBUG
417     serialSensors();
418 #endif
419 }
420
421 // Print data to serial
422 void serialSensors(){
423     if (timerCounter > 120000 || timerCounter == 0)
424     {
425         RTC.read(tm);
426         Serial.print("Time: ");
427         if (tm.Hour <= 9)
428         {
429             Serial.print("0");
430             Serial.print(tm.Hour);
431         }
432         else
433         {
434             Serial.print(tm.Hour);
435         }
436         Serial.print(":");
437         if (tm.Minute <= 9)
438         {
439             Serial.print("0");
440             Serial.print(tm.Minute);
441         }
442         else
443         {
444             Serial.print(tm.Minute);
445         }
446         Serial.print(" ");
447         Serial.print("Date: ");
448         Serial.print(tm.Day);
449         Serial.print(".");
450         Serial.print(tm.Month);
451         Serial.print(".");
452         Serial.println(tmYearToCalendar(tm.Year));
453         timerCounter = 1;

```

```

454     h = dht.readHumidity();
455     t = dht.readTemperature();
456     if (isnan(h) || isnan(t)) {
457         Serial.println("Failed to read from DHT sensor!");
458         return;
459     }
460     else
461     {
462         Serial.print("Humidity: ");
463         Serial.print(h);
464         Serial.print(" % ");
465         Serial.print("Temperature: ");
466         Serial.print(t);
467         Serial.println(" *C ");
468     }
469     }
470     timerCounter++;
471 }
472
473 //File not found
474 void sendFileNotFound(EthernetClient thisClient) {
475     char tBuf[64];
476     strcpy_P(tBuf,PSTR("HTTP/1.0 404 File Not Found\r\n"));
477     thisClient.write(tBuf);
478     strcpy_P(tBuf,PSTR("Content-Type: text/html\r\nConnection: close\r\n\r\n"));
479     thisClient.write(tBuf);
480     strcpy_P(tBuf,PSTR("<html><body><H1>FILE NOT FOUND</H1></body></html>"));
481     thisClient.write(tBuf);
482     thisClient.stop();
483 #ifdef ServerDEBUG
484     Serial.println(F("disconnected"));
485 #endif
486 }
487
488 //Bad http request
489 void sendBadRequest(EthernetClient thisClient) {
490     char tBuf[64];
491     strcpy_P(tBuf,PSTR("HTTP/1.0 400 Bad Request\r\n"));
492     thisClient.write(tBuf);
493     strcpy_P(tBuf,PSTR("Content-Type: text/html\r\nConnection: close\r\n\r\n"));
494     thisClient.write(tBuf);
495     strcpy_P(tBuf,PSTR("<html><body><H1>BAD REQUEST</H1></body></html>"));
496     thisClient.write(tBuf);
497     thisClient.stop();
498 #ifdef ServerDEBUG
499     Serial.println(F("disconnected"));
500 #endif
501 }
502
503 void strtoupper(char* aBuf) {
504
505     for(int x = 0; x<strlen(aBuf);x++) {
506         aBuf[x] = toupper(aBuf[x]);
507     }
508 }
509
510 byte socketStat[MAX_SOCKET_NUM];
511
512 void ShowSockStatus()

```



```

513 {
514   for (int i = 0; i < MAX SOCK_NUM; i++) {
515     Serial.print(F("Socket#"));
516     Serial.print(i);
517     uint8_t s = W5100.readSnSR(i);
518     socketStat[i] = s;
519     Serial.print(F(":0x"));
520     Serial.print(s,16);
521     Serial.print(F(" "));
522     Serial.print(W5100.readSnPORT(i));
523     Serial.print(F(" D:"));
524     uint8_t dip[4];
525     W5100.readSnDIPR(i, dip);
526     for (int j=0; j<4; j++) {
527       Serial.print(dip[j],10);
528       if (j<3) Serial.print(".");
529     }
530     Serial.print(F("("));
531     Serial.print(W5100.readSnDPORT(i));
532     Serial.println(F(")"));
533   }
534 }
535
536 void checkSockStatus()
537 {
538   unsigned long thisTime = millis();
539
540   for (int i = 0; i < MAX SOCK_NUM; i++) {
541     uint8_t s = W5100.readSnSR(i);
542
543     if((s == 0x17) || (s == 0x1C)) {
544       if(thisTime - connectTime[i] > 30000UL) {
545         Serial.print(F("\r\nSocket frozen: "));
546         Serial.println(i);
547         close(i);
548       }
549     }
550     else connectTime[i] = thisTime;
551
552     socketStat[i] = W5100.readSnSR(i);
553   }
554 }
555
556 void XML_response(EthernetClient thisClient){
557   char tunti, minuutti, sekunti;
558   #ifdef ServerDEBUG
559     Serial.print("XML UpdateSite\n");
560   #endif
561   // HTTP Headerin luominen XML tiedostolle
562   thisClient.print("HTTP/1.1 200 OK\r\n");
563   thisClient.print("Content-Type: text/xml\n");
564   thisClient.print("Connection: keep-alive\n");
565   thisClient.print("<?xml version = \"1.0\" ?>\n");
566   thisClient.print("\n");
567
568   //////////// XML ////////////
569   thisClient.print("\n<inputs>\n");
570
571   h = dht.readHumidity();

```

```

572     t = dht.readTemperature();
573
574 #ifndef ServerDEBUG
575     if (isnan(h) || isnan(t)) {
576         Serial.println("Failed to read from DHT sensor!");
577     }
578 #endif
579
580 // Sensors Temperature & Humidity XML creation
581 thisClient.print("<lampo>");
582 thisClient.print(h);
583 thisClient.print("</lampo>\n");
584 thisClient.print("<kosteus1>");
585 thisClient.print("12");
586 thisClient.print("</kosteus1>\n");
587 thisClient.print("<kosteus2>");
588 thisClient.print("19");
589 thisClient.print("</kosteus2>\n");
590 thisClient.print("<kosteus>");
591 thisClient.print(t);
592 thisClient.print("</kosteus>\n");
593
594 // Kellonaika XML luominen
595 thisClient.print("<time>");
596
597 if (tm.Hour <= 9) {
598     thisClient.print("0");
599     thisClient.print(tm.Hour);
600 }
601 else {
602     thisClient.print(tm.Hour);
603 }
604
605 thisClient.print(".");
606
607 if (tm.Minute <= 9) {
608     thisClient.print("0");
609     thisClient.print(tm.Minute);
610 }
611 else {
612     thisClient.print(tm.Minute);
613 }
614 thisClient.println("</time>");
615
616 // Date XML creation
617 thisClient.print("<date>");
618 thisClient.print(tm.Day);
619 thisClient.print(".");
620 thisClient.print(tm.Month);
621 thisClient.print(".");
622 thisClient.print(tmYearToCalendar(tm.Year));
623 thisClient.println("</date>");
624
625 thisClient.print("<motor>");
626 thisClient.print(LED_state[0]);
627 thisClient.println("</motor>");
628 thisClient.print("<pump>");
629 thisClient.print(LED_state[1]);
630 thisClient.println("</pump>");

```



```

631
632     thisClient.print("</inputs>\n");
633     thisClient.stop();
634 }
635
636 // Linechart XML creation
637 void XML_linechart(EthernetClient thisClient)
638 {
639     // HTTP Headerin luominen XML tiedostolle
640     thisClient.print("HTTP/1.1 200 OK\r\n");
641     thisClient.print("Content-Type: text/xml\n");
642     thisClient.print("Connection: keep-alive\n");
643     thisClient.print("<?xml version = \"1.0\" ?>\n");
644     thisClient.print("\n");
645
646     //////////// XML ////////////
647     thisClient.println("\n<inputs>");
648     thisClient.print("<linechart>");
649     for (int k = 0; k < (countChar()*15);k++){
650         thisClient.print(temp_Chart_Buf[k]);
651         Serial.print(temp_Chart_Buf[k]);
652     }
653     thisClient.println("</linechart>");
654     thisClient.println("</inputs>");
655     thisClient.stop();
656 }
657
658 //Button XML creation
659 void XML_nappi(EthernetClient thisClient)
660 {
661     // HTTP Header creation for the XML file
662     thisClient.print("HTTP/1.1 200 OK\r\n");
663     thisClient.print("Content-Type: text/xml\n");
664     thisClient.print("Connection: keep-alive\n");
665     thisClient.print("<?xml version = \"1.0\" ?>\n");
666     thisClient.print("\n");
667
668     //////////// XML ////////////
669     thisClient.print("<inputs>\n");
670     // Button status XML
671     thisClient.print("<LED1>");
672     if (LED_state[0]) {
673         thisClient.print("ON");
674     }
675     else {
676         thisClient.print("OFF");
677     }
678     thisClient.println("</LED1>");
679     // LED4
680     thisClient.print("<LED2>");
681     if (LED_state[1]) {
682         thisClient.print("ON");
683     }
684     else {
685         thisClient.print("OFF");
686     }
687     thisClient.println("</LED2>");
688     thisClient.print("</inputs>\n");
689     thisClient.stop();

```

```

690 }
691
692 //Led port controlling by status read from XML
693 void SetLEDs(void)
694 {
695     // LED 1 (pin 6)
696     if (StrContains(requestBuffer, "LED1=1")) {
697         LED_state[0] = 1; // save LED state
698         digitalWrite(6, HIGH);
699     }
700     else if (StrContains(requestBuffer, "LED1=0")) {
701         LED_state[0] = 0; // save LED state
702         digitalWrite(6, LOW);
703     }
704     // LED 2 (pin 7)
705     if (StrContains(requestBuffer, "LED2=1")) {
706         LED_state[1] = 1; // save LED state
707         digitalWrite(7, HIGH);
708     }
709     else if (StrContains(requestBuffer, "LED2=0")) {
710         LED_state[1] = 0; // save LED state
711         digitalWrite(7, LOW);
712     }
713 }
714 }
715
716 //Checking string content
717 char StrContains(char *str, char *sfind)
718 {
719     char found = 0;
720     char index = 0;
721     char len;
722
723     len = strlen(str);
724
725     if (strlen(sfind) > len) {
726         return 0;
727     }
728     while (index < len) {
729         if (str[index] == sfind[found]) {
730             found++;
731             if (strlen(sfind) == found) {
732                 return 1;
733             }
734         }
735         else {
736             found = 0;
737         }
738         index++;
739     }
740
741     return 0;
742 }
743
744 void checkLastHour(){
745     int numberOfData = countChar();
746     int hourIndexStart = (numberOfData*15)-3;
747     int hourIndexEnd = hourIndexStart + 1;
748     char HourData[2];

```

```

749   HourData[0] = temp_Chart_Buf[hourIndexStart];
750   HourData[1] = temp_Chart_Buf[hourIndexEnd];
751   HourInt = atoi(HourData);
752   }
753
754   void sdCardDatalog(){
755       String str;
756       if (tmYearToCalendar(tm.Year) >= 2000){
757
758   #ifndef ServerDEBUG
759       Serial.println("Datalog");
760   #endif
761
762       a1 = (tm.Day/10);
763       a2 = (tm.Day%10);
764       b1 = (tm.Month/10);
765       b2 = (tm.Month%10);
766       c = (tmYearToCalendar(tm.Year)%2000);
767       c1 = (c/10);
768       c2 = (c%10);
769
770       str = String(a1);
771       str = str + String(a2);
772       str = str + String(b1);
773       str = str + String(b2);
774       str = str + "20";
775       str = str + String(c1);
776       str = str + String(c2);
777       str = str + ".txt";
778
779       str.toCharArray(charFileName,13);
780
781       for (int i = 0; i <= 12;i++){
782   #ifndef ServerDEBUG
783           Serial.print(charFileName[i]);
784   #endif
785       }
786   #ifndef ServerDEBUG
787       Serial.println(" ");
788   #endif
789
790       // Checking if current date file is already on the sd-card
791       if (SD.exists(charFileName))
792       {
793           readFileSd();
794           checkLastHour();
795           //arduino tarkistaa kuinka monta tuntia SD:llä on dataa
796           if (countChar() == 24)
797           {
798               if (SD.exists(charFileName)){
799   #ifndef ServerDEBUG
800                   Serial.println("File already exists for today!");
801   #endif
802               }
803               else
804               {
805
806                   //Luo sd-kortille uusi tiedosto
807                   linechart = SD.open(charFileName, FILE_WRITE);

```

```

808         linechart.close();
809
810         linechart = SD.open(charFileName, FILE_WRITE);
811
812         linechart.print(t);
813         linechart.print(", ");
814         linechart.print(h);
815         linechart.print(", ");
816         if (tm.Hour <= 9) {
817             linechart.print("0");
818             linechart.print(tm.Hour);
819         }
820         else {
821             linechart.print(tm.Hour);
822         }
823         linechart.print(";");
824
825         linechart.close();
826
827     }
828 }
829 else
830 {
831     if (HourInt != tm.Hour){
832         linechart = SD.open(charFileName, FILE_WRITE);
833
834         linechart.print(t);
835         linechart.print(", ");
836         linechart.print(h);
837         linechart.print(", ");
838         if (tm.Hour <= 9) {
839             linechart.print("0");
840             linechart.print(tm.Hour);
841         }
842         else {
843             linechart.print(tm.Hour);
844         }
845         linechart.print(";");
846         linechart.close();
847     }
848     else {
849 #ifdef ServerDEBUG
850         Serial.println("Data for the hour exists");
851 #endif
852     }
853     readFileSd();
854 }
855
856 }
857 else
858 {
859     //Creating new file to sd card by picking up a date from rtc-sensor
860     //Or if the file by current date exists save new sensor data to sd-card
861     Serial.println(charFileName);
862     linechart = SD.open(charFileName, FILE_WRITE);
863     linechart.close();
864
865     readFileSd();
866     checkLastHour();

```

```

867
868     if (SD.exists(charFileName)) {
869         Serial.println("File exists");
870         linechart = SD.open(charFileName, FILE_WRITE);
871
872         linechart.print(t);
873         linechart.print(",");
874         linechart.print(h);
875         linechart.print(",");
876         if (tm.Hour <= 9) {
877             linechart.print("0");
878             linechart.print(tm.Hour);
879         }
880         else {
881             linechart.print(tm.Hour);
882         }
883         linechart.print(";");
884
885         linechart.close();
886
887         readFileSd();
888     }
889     else {
890 #ifdef ServerDEBUG
891         Serial.println("File doesn't exist.");
892 #endif
893     }
894 }
895 #ifdef ServerDEBUG
896     Serial.println("END");
897 #endif
898 }
899 else
900 {
901 #ifdef ServerDEBUG
902     Serial.println("Year is Incorrect, Datalogging is OFF");
903 #endif
904 }
905 }
906
907 //counting how many specific characters are found in char array (string)
908 int countChar()
909 {
910     int koko = sizeof(temp_Chart_Buf);
911     int count = 0;
912     for (int z = 0; z < koko; z++){
913         if (temp_Chart_Buf[z] == ','){
914             count++;
915         }
916     }
917     return count;
918 }
919
920 //Reading sensor data from sd-card to temporary variable on running memory.
921 void readFileSd(){
922
923     linechart = SD.open(charFileName);
924
925     if (linechart) {

```

```
926 #ifdef ServerDEBUG
927     Serial.println("Success! Read");
928 #endif
929     // read from the file until there's nothing else in it:
930     while (linechart.available()) {
931         temp_Chart_Buf[j] = (linechart.read());
932         j++;
933 #ifdef ServerDEBUG
934         Serial.print(temp_Chart_Buf[j]);
935 #endif
936     }
937     j = 0;
938     // close the file:
939     linechart.close();
940 } else {
941 #ifdef ServerDEBUG
942     // if the file didn't open, print an error:
943     Serial.println("Error Reading the File");
944 #endif
945 }
946 }
947
```

Client website

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>Kastelujärjestelmä</title>
6
7     <script language="Javascript" src="/Chart.js"></script>
8
9   <script>
10    var dataSet = [];
11    var tempt = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
12    var tempH = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
13    var tempC = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
14    var strLED1 = "";
15    var strLED2 = "";
16    var LED1_state = 0;
17    var LED2_state = 0;
18
19    function GetArduinoInputs()
20    {
21      nocache = "&nocache=" + Math.random() * 1000000;
22      var request = new XMLHttpRequest();
23      console.log(request);
24      request.onreadystatechange = function()
25      {
26        console.log(this);
27        if (this.readyState == 4) {
28          if (this.status == 200) {
29            if (this.responseXML != null) {
30              // extract XML data from XML file (containing switch states
31              and analog value)
32              document.getElementsByClassName("lamppu")[0].innerHTML =
this.responseXML.getElementsByTagName("lamppu")[0].innerHTML;
33              document.getElementsByClassName("kosteus")[0].innerHTML =
this.responseXML.getElementsByTagName("kosteus")[0].innerHTML;
34              document.getElementsByClassName("kosteus1")[0].innerHTML =
this.responseXML.getElementsByTagName("kosteus1")[0].innerHTML;
35              document.getElementsByClassName("kosteus2")[0].innerHTML =
this.responseXML.getElementsByTagName("kosteus2")[0].innerHTML;
36              document.getElementsByClassName("time")[0].innerHTML =
this.responseXML.getElementsByTagName("time")[0].innerHTML;
37              document.getElementsByClassName("date")[0].innerHTML =
this.responseXML.getElementsByTagName("date")[0].innerHTML;
38              document.getElementsByClassName("motor")[0].innerHTML =
this.responseXML.getElementsByTagName("motor")[0].innerHTML;
39              document.getElementsByClassName("pump")[0].innerHTML =
this.responseXML.getElementsByTagName("pump")[0].innerHTML;
40
41              if
42              (parseFloat(this.responseXML.getElementsByTagName("motor")[0].innerHTML) == 1)
43              {
44                document.getElementsByClassName("motor")[0].innerHTML =
45                "ON"
46              }
47              else{
48                document.getElementsByClassName("motor")[0].innerHTML =
49                "OFF"
50              }
51              if (parseFloat(this.responseXML.getElementsByTagName("pump")
[0].innerHTML) == 1)
52              {
53                document.getElementsByClassName("pump")[0].innerHTML =
54                "ON"
55              }

```



```

52         else{
53             document.getElementsByClassName("pump")[0].innerHTML =
"OFF"
54         }
55     }
56 }
57 }
58 }
59     request.open("GET", "inputs" + nocache, true);
60     request.send(null);
61     setTimeout('GetArduinoInputs()', 10000);
62 }
63
64 setInterval(function getLinechart()
65 {
66     nocache = "&nocache=" + Math.random() * 1000000;
67     var request = new XMLHttpRequest();
68     console.log(request);
69     request.onreadystatechange = function()
70     {
71         if (this.readyState == 4) {
72             if (this.status == 200) {
73                 if (this.responseXML != null) {
74
75                     dataSet =
this.responseXML.getElementsByTagName("linechart")[0].innerHTML;
76
77                     var res = dataSet.split(";");
78
79                     console.log(res);
80
81                     for (var i = 0; i < (res.length-
82 1); ++i)
83                     {
84                         var temp = res[i].split(",");
85
86                         tempt[i] = parseFloat(temp[0]);
87                         temph[i] = parseFloat(temp[1]);
88                         tempc[i] = temp[2];
89                     }
90
91                     console.log(tempt);
92                     console.log(temph);
93                     console.log(tempc);
94                 }
95             }
96         }
97     }
98     request.open("GET", "linechart" + nocache, true);
99     request.send(null);
100 },5000);
101
102 function GetIO()
103 {
104     nocache = "&nocache=" + Math.random() * 1000000;
105     var request = new XMLHttpRequest();
106     console.log(request);
107     request.onreadystatechange = function()
108     {
109         console.log(this);
110         if (this.readyState == 4) {

```



```

111         if (this.status == 200) {
112             if (this.responseXML != null) {
113                 document.getElementById("LED1").value =
this.responseXML.getElementsByTagName("LED1")[0].innerHTML;
114                 document.getElementById("LED2").value =
this.responseXML.getElementsByTagName("LED2")[0].innerHTML;
115             }
116         }
117     }
118     request.open("GET", strLED1 + strLED2 + nocache, true);
119     request.send(null);
120 }
121
122 function GetButton1()
123 {
124     if (LED1_state == 1) {
125         LED1_state = 0;
126         strLED1 = "&LED1=0";
127     }
128     else {
129         LED1_state = 1;
130         strLED1 = "&LED1=1";
131     }
132     GetIO();
133 }
134
135 function GetButton2()
136 {
137     if (LED2_state == 1) {
138         LED2_state = 0;
139         strLED2 = "&LED2=0";
140     }
141     else {
142         LED2_state = 1;
143         strLED2 = "&LED2=1";
144     }
145     GetIO();
146 }
147
148
149
150 </script>
151
152 <style>
153     body {
154         margin: auto;
155         padding: 0px;
156         text-align: center;
157         width: 100%;
158         float: right;
159     }
160     h1 {
161         text-align: center;
162         font-family: Arial, "Trebuchet MS", Helvetica, sans-serif;
163     }
164     h2 {
165         text-align: center;
166         font-family: Arial, "Trebuchet MS", Helvetica, sans-serif;
167     }
168
169     .box {
170         margin: 5px auto 5px auto;
171         padding: 0 10px 0 10px;
172

```

```
173         text-align:center;
174     }
175 }
176
177     .IO_box {
178         margin: 5px auto auto auto;
179         border: 1px solid black;
180         padding: 0px 0px 0px 10px;
181         text-align:left;
182         display: inline-block;
183 width: 200px;
184     }
185
186     .IO_box1 {
187         margin: 5px auto auto auto;
188         border: 1px solid black;
189         padding: 0 10px 0 10px;
190 width: 190px;
191     }
192
193     .box_button {
194         margin: 20px auto 5px auto;
195         border: 1px solid black;
196         padding: 0 10px 0 10px;
197         width: 150px;
198     }
199
200     .custom-button-style1{
201         margin: 5px 5px 5px 5px;
202         padding: 10px 15px;
203         color: #FFDFDF;
204         background-color: #555;
205         border: 0 none;
206         -webkit-border-radius: 3px;
207         -moz-border-radius: 3px;
208         border-radius: 3px;
209         cursor: pointer;
210     }
211
212     .small_text {
213         font-size: 70%;
214         color: #737373;
215     }
216
217     #chart_div {
218 margin: 5px auto 5px auto;
219 height: 400px;
220 width: 600px;
221 padding: 10px 15px;
222 border: 1px solid black;
223     }
224
225     input[type="button"] {
226         margin: 5px auto 5px 5px;
227         padding: 0 10px 0 10px;
228         width: 50px;
229         height: 30px;
230         display:inline;
231     }
232
233     h4 {
234         margin: auto auto 20px;
235         padding: 0 10px 0 10px;
236         width: 50px;
237         height: 30px;
```

```

238         display:inline;
239     }
240
241     .motor{
242         float:right;
243         margin: auto 30% auto auto;
244     }
245
246     .pump{
247         float:right;
248         margin: auto 30% auto auto;
249     }
250
251     .lampe
252     {
253         float:right;
254         margin: auto 30% auto auto;
255     }
256
257     .kosteus
258     {
259         float:right;
260         margin: auto 30% auto auto;
261     }
262
263     h1
264     {
265         margin: 2% auto auto auto;
266         padding: 0 10px 0 10px;
267     }
268
269
270     </style>
271 </head>
272
273 <body onload="GetArduinoInputs(); lineChartBuild()">
274     <h1>Arduino Greenhouse</h1>
275     <div class="box">
276         <span class="time">...</span>
277         <span class="date">...</span>
278     </div>
279
280     <div class="IO_box">
281         <h2>Soil Moisture</h2>
282         <p>Kosteusanturi 1: <span class="kosteus1">...</span>%</p>
283         <p>Kosteusanturi 2: <span class="kosteus2">...</span>%</p>
284     </div>
285
286     <div class="IO_box">
287         <h2>Temperature</h2>
288         <p>Humidity: <span class="lampe">...</span></p>
289         <p>Temperature: <span class="kosteus">...</span></p>
290     </div>
291
292     <div class="IO_box">
293         <h2>Motor Status</h2>
294         <p>Motor <span class="motor">...</span></p>
295         <p>Water Pump <span class="pump">...</span></p>
296     </div>
297
298     <div class="IO_box1">
299         <h4>Motor</h4><h4>Pump</h4><br>
300         <input type="button" id="LED1" value="OFF"
onclick="GetButton1()" />
301

```

```

302         <input type="button" id="LED2" value="OFF"
onclick="GetButton2()" />
303     </div>
304     <div id="chart_div">
305         <canvas id="canvas" height="400" width="600"></canvas>
306     </div>
307
308
309 <script>
310     var startingData = {
311         labels: [00,01,02,03,04,05,06,07,08,09,10,11,12,13,14,15,16,17,18,19,20,21,22,23],
312         datasets: [
313             {
314                 fillColor: "rgba(220,220,220,0.2)",
315                 strokeColor: "rgba(220,220,220,1)",
316                 pointColor: "rgba(220,220,220,1)",
317                 pointStrokeColor: "#fff",
318                 data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
319             },
320             {
321                 fillColor: "rgba(151,187,205,0.2)",
322                 strokeColor: "rgba(151,187,205,1)",
323                 pointColor: "rgba(151,187,205,1)",
324                 pointStrokeColor: "#fff",
325                 data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
326             }
327         ]
328     };
329
330     var canvas = document.getElementById("canvas")
331     var ctx = canvas.getContext('2d')
332     var myLineChart = new Chart(ctx).Line(startingData, {scaleShowLabels: true});
333
334     setInterval(function(){
335         for(var i = 0; i<24;i++){
336             myLineChart.scale.xLabels[i] = tempt[i];
337             myLineChart.datasets[0].points[i].value = tempt[i];
338             myLineChart.datasets[1].points[i].value = temph[i];
339         }
340         myLineChart.update();
341     }, 5000);
342
343
344
345     </script>
346 </body>
347 </html>
348

```