Eetu-Pekka Kouhia

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

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology
May 2016
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.

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.

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.

Key words
Arduino, embedded systems, sensors, webserver
# 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 PC 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 (Fritzng 2013) ................................................16
GRAPH 13. Wiring diagram of the hardware (Fritzng 2013) ................................................16
GRAPH 14. Arduino software flowchart (adapted from Llamos 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 website .........................................................................................20
GRAPH 18. Frizing circuit diagram ........................................................................................................21
GRAPH 19. Electrical connection diagram from DHT22 datasheet .........................................................22
GRAPH 20. Code snippet of serial printing used for testing .................................................................23

TABLES
TABLE 1. Testing model matrix ..............................................................................................................7
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).](image-url)
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).

<table>
<thead>
<tr>
<th>Static Testing</th>
<th>Black Box Testing</th>
<th>White Box Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Testing the product specifications by:</td>
<td>Process of methodically reviewing hardware and code for bugs without executing it.</td>
</tr>
<tr>
<td></td>
<td>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.).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Low-level specification testing by insure completeness, accuracy, preciseness, consistency, relevance, feasibility, etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Testing</th>
<th>Requires definition of what software and hardware does, includes:</th>
<th>Testing running system while looking at code, schematics, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- <strong>data testing</strong>, which is checking info of user inputs and outputs</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>- <strong>boundary condition testing</strong>, which is testing situations at edge of planned operational limits of software</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>internal boundary testing</strong>, which is testing powers-of-two, ASCII table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>input testing</strong>, which is testing null, invalid data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>state testing</strong>, which is testing modes and transitions between modes software is in with state variables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
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.)

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.

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 or 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.)

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 buys 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.
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 I^C 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.
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.
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).

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.

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.

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.

<table>
<thead>
<tr>
<th>Soil Moisture</th>
<th>Temperature</th>
<th>Motor Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kosteusautti 1: 12%</td>
<td>Humidity: 35.60</td>
<td>Motor: OFF</td>
</tr>
<tr>
<td>Kosteusautti 2: 19%</td>
<td>Temperature: 21.30</td>
<td>Water Pump: OFF</td>
</tr>
</tbody>
</table>

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. Fritzing 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.

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

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


System setup() flowchart

Start

Include libraries

Variable declarations

Setup serial

Setup digital humidity and temperature sensor

Setup SD-card

Setup network interface card and web server

END
CheckServer function flowchart

START

network client exists

- network client exists
  - NO
  - YES

client connected

- client connected
  - NO
  - YES

client available

- client available
  - NO
  - YES

Read client request

- Read client request
  - YES
  - NO

client request is valid

- client request is valid
  - NO
  - YES

client request contains legal characters

- client request contains legal characters
  - NO
  - YES

client request contains "inputs"

- client request contains "inputs"
  - NO
  - YES

client request contains "LED"

- client request contains "LED"
  - NO
  - YES

client request contains "linechart"

- client request contains "linechart"
  - NO
  - YES

client request contains "inputs" "linechart"

- client request contains "inputs" "linechart"
  - NO
  - YES

Read sensor data

- Read sensor data
  - YES
  - NO

SEND_BUG_REQUEST()
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

*/
Web server sketch for IDE v1.0.5 and w5100/w5200
Original web server example by SurferTim
Last modified 6 June 2015
http://playgroundarduino.com/Code/WebServerST

Arduino Watering System:
Webserver hosting air moisture, soil moisture and temperature to a website on user
defined address in local network.
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.
Every day new file is created and each file is then containing 24 values of soil
moisture, air moisture and air temperature.

Written by Eetu-Pekka Kouhia
26.10.2015
*/

//Additional Libraries written for certain sensors
#include <DHT.h>
#include <Wire.h>
#include <Time.h>
#include <DS1307RTC.h>
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>
#include <utility/w5100.h>
#include <utility/socket.h>

//Temperature & Humidity Sensor setup
#define DMTPIN 2  // Pin connected to Temperature & Humidity sensor
#define DMTTYPE DMT22  // DMT22 (AM2302), Model of the Humidity and Temperature
module

#define ServerDEBU // Used for debugging the server, if defined serial print enabled

//Network settings
byte mac[] = { 0x0D, 0x4A, 0x0B, 0x0F, 0x0F, 0x0F };  // Mac-address of the LAN-shield
IPAddress ip(192.168,1,10);  // Ip-address of the lan-shield
IPAddress gateway(192.168,1,1);  // Gateway of the lan-shield
IPAddress subnet(255,255,255,0);  // Subnet of the lan-shield

tmElements_t tm;  // Declaring time object to store data read from realtime clockmodule

//Setting up temporary buffer to store data from SD-card
char temp_Chat_Buf[362];

//Pre-filled buffer to test XML-send and Chart.JS populating
char temp_Chat_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;25.11,23.12,22
2;15.15,23.12,23;";

File linechart;  // Declaring for File object for SD-card
file.
char charFileName[] = "00000000.TXT";  // Declaring variable to store filename
populated from date and year
int runOnce = 1; //Declaring variable to store function run
counter
int sdStatus = 0; //Declaring variable to store SD-card read
status
int lastHour = 0; //Declaring variable to check what was the
previous hour, used to time linechart data save on SD-card every hour.
int a1,b1,c1,a2,b2,c2,c3;
//Declaring variables to help populating CharFile瞠[]
int j = 0; //Declaring variable to count loop run
char requestBuffer[64]; //Declaring buffer to store GET request
int LED_state[{}]; //Declaring array to store I/O status
int HourInt;
data from SD-card
int loopCount = 0;
timerCounter = 0;
const char legalChars[] = "/ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234567890/_.-";
unsigned int requestNumber = 0;
unsigned long connectTime[MAX.SOCK_NUM];

//Declaring variables to store Temperature & Humidity sensor data
float h = 0;
float t = 0;

//Declaring use of the Temperature & Humidity sensor (library DHT)
DHT dht(DHTPIN, DHTTYPE,30);

//Declaring port where the webserver is hosted
EthernetServer server(80);

//Function running once at the beginning
void setup(){
  Serial.begin(9600);
  // disable w5100 SPI while starting SD
digitalWrite(10,HIGH);
digitalWrite(3,HIGH);
Serial.println(F("Starting DHT-sensor"));
dht.begin();
delay(2000);
#endif

#if ServerDEBUG
  Serial.print(F("Starting SD..."));
#endif

if(!SD.begin(4)){
  #ifdef ServerDEBUG
    Serial.println(F("failed"));
  #endif
  else {
    #ifdef ServerDEBUG
      Serial.println(F("ok"));
      sdStatus = 1;
    #endif
  }
Ethernet.begin(mac, ip, gateway, gateway, subnet);
delay(2000);
server.begin();

unsigned long thisTime = millis();
for(int i=0;i<MAX.SOCK.NUM;i++) {
    connectTime[i] = thisTime;
}

#ifdef ServerDEBUG
    Serial.println(F("Ready"));
#endif

// Main loop running constantly
void loop()
{
    checkServer();
    myStuff();
    datalogTimer();
}

void myStuff()
{
    if(Serial.available()) {
        if(Serial.read() == 'r') showSockStatus();
    }
    checkSockStatus();
}

void datalogTimer()
{
    if(!sdStatus())
    {
        if(tm.Hour != lastHour)
        {
            Serial.print("sdCardDatalog()\n");
            sdCardDatalog();
            lastHour = tm.Hour;
            runOnce = 1;
        }
    }

    else
    {
        if(runOnce)
        {
            Serial.println("Inside timer Datalog\n");
            runOnce = 0;
        }
    }
}

else
{
    if(runOnce)
    {
        Serial.println("Cannot Datalog SD failure\n");
        runOnce = 0;
    }
}
void checkServer()
{
    EthernetClient client = server.available();
    if (client) {
        boolean currentLineIsBlank = true;
        boolean currentLineIsGet = true;
        int tCount = 0;
        char tBuf[128];
        int r, t;
        char *pch;
        char methodBuffer[64];
        char pageBuffer[64];
        char paramBuffer[64];
        char protocolBuffer[50];
        char fileName[32];
        char fileType[1];
        int httpClientCount = 0;
        requestNumber++;
        #ifdef ServerDEBUG
        Serial.print(F("\r\nClient request ");
        Serial.print(requestNumber);
        Serial.print(F(" "););
        #endif
        // this controls the timeout
        int loopCount = 0;
        while (client.connected()) {
            while (client.available()) {
                // if packet, reset loopCount
                loopCount = 0;
                char c = client.read();
                if (currentLineIsGet && tCount < 65)
                {
                    tBuf[tCount] = c;
                    tCount++;
                    tBuf[tCount] = 0;
                }
                if (c == '\n' && currentLineIsBlank) {
                    #ifdef ServerDEBUG
                    Serial.print(tBuf);
                    #endif
                    while (client.available()) client.read();
                    int scanCount = sscanf(tBuf, "%7s %47s
%s", methodBuffer, requestBuffer, protocolBuffer);
                    if (scanCount != 3) {
                        #ifdef ServerDEBUG
                        Serial.println(F("bad request"));
                        #endif
                        sendBadRequest(client);
                        return;
                    } else {
                        char * pch = strtok(requestBuffer,"?");
                        if (pch != NULL) {
                    
                
            


```c
strcpy(fileName, pch, 31);
strcpy(tBuf, pch, 31);
pch = strtok(NULL, ",?");
if (pch != NULL) {
    strcpy(paramBuffer, pch);
} else paramBuffer[0] = 0;
if (StrContains(requestBuffer, "inputs") && XML_response(client));
if (StrContains(requestBuffer, "LED") && XML_nappi(client);
    SetLEDs();
}
if (StrContains(requestBuffer, "linechart") && XML_linechart(client);
strtoupper(requestBuffer);
strtoupper(tBuf);
for (int x = 0; x < strlen(requestBuffer); x++) {
    if (strchr(legalChars, requestBuffer[x]) == NULL) {
        Serial.println(F("BAD character");
        sendBadRequest(client);
        return;
    }
}
#ifdef ServerDEBUG
Serial.print(F("file = "));
Serial.println(requestBuffer);
#endif
pch = strtok(tBuf, ",.");
if (pch != NULL) {
    pch = strtok(NULL, ",.");
    if (pch != NULL) strcpy(fileType, pch, 4);
    else fileType[0] = 0;
#endif
#ifdef ServerDEBUG
Serial.print(F("file type = "));
Serial.println(fileType);
#endif
#ifdef ServerDEBUG
Serial.print(F("Method = "));
Serial.println(methodBuffer);
#endif
if (strcmp(methodBuffer, "GET") != 0 && strcmp(methodBuffer, "HEAD") != 0) { 
    sendBadRequest(client);
    return;
}
```
```cpp
    #ifdef ServerDEBUG
        Serial.println(F("params = "));
        Serial.println(paramBuffer);
        Serial.println(F("protocol = "));
        Serial.println(protocolBuffer);
    #endif
    
    if(strcmp(requestBuffer,"/INDEX.HTM") == 0) {
        #ifdef ServerDEBUG
            Serial.println(F("dynamic page"));
        #endif
        
        } else {
            if(strcmp(fileName,"/")) {
                strcpy(fileName,"/INDEX.HTM");
                strcpy(fileType,".HTM");
            }
    #ifdef ServerDEBUG
            Serial.println(F("Home page "));
        #endif
    }
    #ifdef ServerDEBUG
        Serial.println(F("SD file"));
    #endif
    
    if(strlen(fileName) > 30) {
    #ifdef ServerDEBUG
        Serial.println(F("filename too long"));
    #endif
        sendBadRequest(client);
        return;
    }
    
    else if(strlen(fileType) > 3 || strlen(fileType) < 1) {
        #ifdef ServerDEBUG
        Serial.println(F("file type invalid size"));
    #endif
        sendBadRequest(client);
        return;
    }
    
    else {
    #ifdef ServerDEBUG
        Serial.println(F("filename format ok"));
    #endif
        if(SD.exists(fileName)) {
            #ifdef ServerDEBUG
            Serial.println(F("File found.
```

```c
#ifdef ServerDEBUG
    else Serial.print(F("opened.");
#endif
strcpy_P(tBuf,PSTR("HTTP/1.0 200 OK\r\n\nContent-Type: ");

// send Content-Type
if(strncmp(fileType,"HTM") == 0) strcat_P(tBuf,PSTR("text/html"));
else if(strncmp(fileType,"PHP") == 0) strcat_P(tBuf,PSTR("text/html"));
else if(strncmp(fileType,"TXT") == 0) strcat_P(tBuf,PSTR("text/plain"));
else if(strncmp(fileType,"CSS") == 0) strcat_P(tBuf,PSTR("text/css"));
else if(strncmp(fileType,"JS") == 0) strcat_P(tBuf,PSTR("application/javascript"));
else strcat_P(tBuf,PSTR("text/plain"));

strcat_P(tBuf,PSTR("\r\n\n\r\n\nConnection: close\r\n\n\n"));
client.write(tBuf);

if(strncmp(methodBuffer,"GET") == 0) {
    #ifdef ServerDEBUG
        Serial.print(F("send...");
    #endif
    while(myFile.available()) {
        clientCount = myFile.read(tBuf,64);
        client.write((Byte*)tBuf,clientCount);
    }
    myFile.close();
#endif ServerDEBUG
    Serial.println(F("closed");
#endif ServerDEBUG
    client.stop();
#endif ServerDEBUG
    Serial.println(F("disconnected");
    return;
}
#else {
#endif ServerDEBUG
    Serial.println(F("File not found");
    #endif
    sendFileNotFound(client);
    return;
}
#endif ServerDEBUG
Serial.println(F("Sending response");
strcpy_P(tBuf,PSTR("HTTP/1.1 200 OK\r\n\nContent-Type: text/html\r\n\n\n"));
client.write(tBuf);
client.stop();
}
else if (c == '\n') {
    currentLineIsBlank = true;
    currentLineIsGet = false;
```
else if (c != '\t') {
    if (currentLineIsBlank)
    {
        currentLineIsBlank = false;
    }
    loopCount++;
}
// if 1 second has passed since last packet
if(loopCount > 1000) {
    // close connection
    client.stop();
    #ifdef ServerDEBUG
    Serial.println("\nTimeout");
    #endif
    // delay 1ms for timeout timing
    delay(1);
    #ifdef ServerDEBUG
    Serial.println(F("Disconnected"));
    #endif
    #ifdef ServerDEBUG
    serialSensors();
    #endif
}
// Print data to serial
void serialSensors(){
    if (timerCounter > 120000 || timerCounter == 0)
    {
        RTC.read(tm);
        Serial.print("Time: ");
        if (tm.Hour <= 9)
        {
            Serial.print("0");
            Serial.print(tm.Hour);
        }
        else
        {
            Serial.print(tm.Hour);
        }
        Serial.print(":");
        if (tm.Minute <= 9)
        {
            Serial.print("0");
            Serial.print(tm.Minute);
        }
        else
        {
            Serial.print(tm.Minute);
        }
        Serial.print(" ");
        Serial.print("Date: ");
        Serial.print(tm.Day);
        Serial.print(".");
        Serial.print(tm.Month);
        Serial.print(".");
        Serial.println(tmYearToCalendar(tm.Year));
        timerCounter = 1;
h = dht.readHumidity();
t = dht.readTemperature();
if (isnan(h) || isnan(t)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
} else {
  Serial.println("Humidity: ");
  Serial.print(h);
  Serial.println("% ");
  Serial.println("Temperature: ");
  Serial.print(t);
  Serial.println("°C ");
  timerCounter++;
}

// File not found
void sendFileNotFound(EthernetClient thisClient) {
  char tBuf[64];
  strcpy_P(tBuf, PSTR("HTTP/1.0 404 File Not Found\r\n"));
  thisClient.write(tBuf);
  strcpy_P(tBuf, PSTR("Content-Type: text/html\r\nConnection: close\r\n\r\n"));
  thisClient.write(tBuf);
  strcpy_P(tBuf, PSTR("<html><body><h1>FILE NOT FOUND</h1></body></html>"));
  thisClient.write(tBuf);
  thisClient.stop();
  #ifdef ServerDEBUG
  Serial.println(F("disconnected"));
  #endif
}

// Bad http request
void sendBadRequest(EthernetClient thisClient) {
  char tBuf[64];
  strcpy_P(tBuf, PSTR("HTTP/1.0 400 Bad Request\r\n"));
  thisClient.write(tBuf);
  strcpy_P(tBuf, PSTR("Content-Type: text/html\r\nConnection: close\r\n\r\n"));
  thisClient.write(tBuf);
  strcpy_P(tBuf, PSTR("<html><body><h1>BAD REQUEST</h1></body></html>"));
  thisClient.write(tBuf);
  thisClient.stop();
  #ifdef ServerDEBUG
  Serial.println(F("disconnected"));
  #endif
}

void strtcupper(char* aBuf) {
  for(int x = 0; x<strlen(aBuf);x++) {
    aBuf[x] = toupper(aBuf[x]);
  }
}

byte socketStat[MAX_SOCKET_NUM];

void ShowSockStatus()
```c
{ 
for (int i = 0; i < MAX.SOCK.NUM; i++) {
    Serial.print(F("Socket "))); 
    Serial.print(i); 
    uint8_t s = W5100.readSnSR(i); 
    socketStat[i] = s; 
    Serial.print(F(" 0x");
    Serial.print(s, 16); 
    Serial.print(F(" ");
    Serial.print(W5100.readSnPORT(i)); 
    Serial.print(F(" D: ");
    uint8_t dip[i]; 
    W5100.readSnDIPR(i, dip); 
    for (int j=0; j<i; j++) {
        Serial.print(dip[j], 10); 
        if (j<i) Serial.print(" .");
    }
    Serial.print(F(" ");
    Serial.print(W5100.readSnDPORT(i));
    Serial.println(F(" "));
}
void checkSocketStatus() 
{ 
    unsigned long thisTime = millis();
    for (int i = 0; i < MAX.SOCK.NUM; i++) {
        uint8_t s = W5100.readSnSR(i); 
        if(s == OX1F || (s == OX1C)) {
            if(thisTime - connectTime[i] > 30000UL) {
                Serial.print(F(" \nSocket frozen: ");
                Serial.println(i);
                close(i);
            }
        }
    }
 else connectTime[i] = thisTime;
    socketStat[i] = W5100.readSnSR(i); 
}
void XML_response(EthernetClient thisClient){
char tunti, minuutti, sekunti;
#define ServerDEBUG
    Serial.print("XML UpdateSite \n");
#endif
    // HTTP Headerin luominen XML tiedostolle
    thisClient.print("HTTP/1.1 200 OK\r\n");
    thisClient.print("Content-Type: text/xml\n");
    thisClient.print("Connection: keep-alive\n");
    thisClient.print("<xml version = "1.0" ?>\n");
    thisClient.print("\n");
    //////////////////////////////////////////////////////
    thisClient.print("\n<ingute>\n");
    h = dht.readHumidity();
    ```
```c
    t = dht.readTemperature();

    if (isnan(h) || isnan(t)) {
        Serial.println("Failed to read from DHT sensor");
    }

    // Sensors Temperature & Humidity XML creation
    thisClient.print("<lampo>");
    thisClient.print(h);
    thisClient.print("</lampo>
");
    thisClient.print("<kosteus1>");
    thisClient.print(h);
    thisClient.print("</kosteus1>
");
    thisClient.print("<kosteus2>");
    thisClient.print(t);
    thisClient.print("</kosteus2>");
    thisClient.print("<kosteus>");
    thisClient.print(t);
    thisClient.print("</kosteus>");

    // Kellonaiksi XML luominen
    thisClient.print("<time>");

    if (tm.Hour <= 9) {
        thisClient.print("0");
    />
    thisClient.print(tm.Hour);
    }
    else {
        thisClient.print(tm.Hour);
    }

    thisClient.print(".");

    if (tm.Minute <= 9) {
        thisClient.print("0");
    }
    thisClient.print(tm.Minute);
    }
    else {
        thisClient.print(tm.Minute);
    }
    thisClient.println("</time>");

    // Date XML creation
    thisClient.print("<date>");
    thisClient.print(tm.Day);
    thisClient.print(".");
    thisClient.print(tm.Month);
    thisClient.print(".");
    thisClient.print(tmYearToCalendar(tm.Year));
    thisClient.println("</date>");

    thisClient.print("<motor>");
    thisClient.print(LED_state[1]);
    thisClient.print("</motor>");
    thisClient.print("<pump>");
    thisClient.print(LED_state[1]);
    thisClient.println("</pump>");
```
```cpp
thisClient.println("</inputs>");
thisClient.stop();
}

// Linechart XML creation
void XML_linechart(EthernetClient thisClient)
{
    // HTTP Header
    thisClient.print("HTTP/1.1 200 OK
\r\n");
    thisClient.println("Content-Type: text/xml\n");
    thisClient.print("Connection: keep-alive\n");
    thisClient.println("<?xml version = "1.0" encoding = "\n");
    thisClient.println("\n");

    // XMLContent
    thisClient.println("<inputs>\n");
    thisClient.println("<linechart>\n");
    for (int k = 0; k < (countChar() + 15); k++)
    {
        thisClient.println(temp_Out1[k]);
        Serial.print(temp_Out1[k]);
    }
    thisClient.println("</linechart>\n");
    thisClient.println("</inputs>\n");
    thisClient.stop();
}

// Button XML creation
void XML_nappi(EthernetClient thisClient)
{
    // HTTP Header creation for the XML file
    thisClient.print("HTTP/1.1 200 OK
\r\n");
    thisClient.println("Content-Type: text/xml\n");
    thisClient.print("Connection: keep-alive\n");
    thisClient.println("<?xml version = "1.0" encoding = "\n");
    thisClient.println("\n");

    // XMLContent
    thisClient.println("<inputs>\n");
    // Button status XML
    thisClient.println("<nappi>");
    if (LED_state1[])
    {
        thisClient.println("<ON>\n");
    }
    else
    {
        thisClient.println("<OFF>\n");
    }
    thisClient.println("</nappi>");
    // LED4
    thisClient.println("<LED4>\n");
    if (LED_state4[])
    {
        thisClient.println("<ON>\n");
    }
    else
    {
        thisClient.println("<OFF>\n");
    }
    thisClient.println("</LED4>\n");
    thisClient.println("</inputs>\n");
    thisClient.stop();
```
APPENDIX 4/13

690 }
691 // Led port controlling by status read from XML
692 void SetLEDs(void)
693 {
694     // LED 1 (pin 6)
695     if (StrContains(requestBuffer, "LED1-1")) {
696         LED_state[0] = 1; // save LED state
697         digitalWrite(6, HIGH);
698     }
699     else if (StrContains(requestBuffer, "LED1-0")) {
700         LED_state[0] = 0; // save LED state
701         digitalWrite(6, LOW);
702     }
703     // LED 2 (pin 7)
704     if (StrContains(requestBuffer, "LED2-1")) {
705         LED_state[1] = 1; // save LED state
706         digitalWrite(7, HIGH);
707     }
708     else if (StrContains(requestBuffer, "LED2-0")) {
709         LED_state[1] = 0; // save LED state
710         digitalWrite(7, LOW);
711     }
712 }
713
714 // Checking string content
715 char StrContains(char *str, char *sfind)
716 {
717     char found = 0;
718     char index = 0;
719     char len;
720     len = strlen(str);
721     if (strlen(sfind) > len) {
722         return 0;
723     }
724     while (index < len) {
725         if (str[len - 1] == sfind[found]) {
726             found++;
727             if (strlen(sfind) == found) {
728                 return 1;
729             }
730         } else {
731             found = 0;
732             index++;
733         }
734     }
735     return 0;
736 }
737
738 void checkLastHour(){
739     int numberOfData = countChar();
740     int hourIndexStart = (numberOfData*15)/3;
741     int hourIndexEnd = hourIndexStart + 1;
742     char HourData[4];
void sdCardDatelog(){
    String str;
    if (tmYearToCalendar(tm.Year) >= 2000){
        #ifdef ServerDEBUG
        Serial.println("Datalog");
        #endif
        a1 = (tm.Day/10);
        a2 = (tm.Day%10);
        b1 = (tm.Month/10);
        b2 = (tm.Month%10);
        c = (tmYearToCalendar(tm.Year)%2000);
        c1 = (c/10);
        c2 = (c%10);
        str = String(a1);
        str = str + String(a2);
        str = str + String(b1);
        str = str + String(b2);
        str = str + "20";
        str = str + String(c1);
        str = str + String(c2);
        str = str + ".txt";
        str.toCharArray(charFileName,13);
        for (int i = 0; i <= 12; i++){
            #ifdef ServerDEBUG
            Serial.println(charFileName[i]);
            #endif
        }
        #ifdef ServerDEBUG
        Serial.println(".");
        #endif
        // Checking if current date file is already on the sd-card
        if (SD.exists(charFileName))
        {
            readFileId();
            checkLastHour();
            //arduino tarkistaa kuinka monta tuntia SD:llä on dataa
            if (countChar() == 24)
            {
                #ifdef ServerDEBUG
                Serial.println("File already exists for today!");
                #endif
            } else
            {
                //Luku SD-kortille uusi tiedosto
                linechart = SD.open(charFileName, FILE_WRITE);
            }
        } else
        {
            //Luku SD-kortille uusi tiedosto
            linechart = SD.open(charFileName, FILE_WRITE);
        }
    } else
    {
        //Luku 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 <= 5) {
    linechart.print("0");
    linechart.print(tm.Hour);
}
else {
    linechart.print(tm.Hour);
}
linechart.print(";");

linechart.close();

}
else {
    if (HourInt != tm.Hour){
        linechart = SD.open(charFileName, FILE_WRITE);
        linechart.print(t);
        linechart.print(",");
        linechart.print(h);
        linechart.print(",");
        if (tm.Hour <= 5) {
            linechart.print("0");
            linechart.print(tm.Hour);
        }
        else {
            linechart.print(tm.Hour);
        }
        linechart.print(";");
        linechart.close();
    }
    else {
        #ifdef ServerDEBUG
        Serial.println("Data for the hour exists");
        #endif
    }
    readFileSd();
}

else {
    //Creating new file to sd card by picking up a date from rtc-sensor
    //Or if the file by current date exists save new sensor data to sd-card
    Serial.println("charFileName");
    linechart = SD.open(charFileName, FILE_WRITE);
    linechart.close();
    readFileSd();
    checklistHour();
}
if (SD.exists(charFileName)) {
    Serial.println("File exists");
    linechart = SD.open(charFileName, FILE_WRITE);
    linechart.print(t);
    linechart.print("");
    linechart.print(h);
    linechart.print("");
    if (tm.Hour <= 0) {
        linechart.print("0");
        linechart.print(tm.Hour);
    }
    else {
        linechart.print(tm.Hour);
    }
    linechart.print(";");
    linechart.close();
    readFileSd();
} 
else {
    #ifdef ServerDEBUG
    Serial.println("File doesn't exist.");
    #endif
}
#endif ServerDEBUG
#ifdef ServerDEBUG
Serial.println("END");
#endif
else {
    #ifdef ServerDEBUG
    Serial.println("Year is Incorrect, Datalogging is OFF");
    #endif
}
}

//counting how many specific characters are found in char array (string)
int countChar() {
    int koko = sizeof(temp_Chart_Buf);
    int count = 0;
    for (int z = 0; z < koko; z++){
        if (temp_Chart_Buf[z] ==");
        count++;
    }
    return count;
}

//Reading sensor data from sd-card to temporary variable on running memory.
void readFileSc() {
    linechart = SD.open(charFileName);
    if (linechart) {
        // Code snippet for reading data from SD card and saving it into temporary variables
#ifdef ServerDEBUG
    Serial.println("Success! Read");
#endif

    // read from the file until there's nothing else in it:
    while (linechart.available()) {
        temp_Chat_Buf[j] = (linechart.read());
        j++;
    }
#endif
    Serial.println(temp_Chat_Buf[j]);
#endif

    j = 0;
    // close the file:
    linechart.close();
} else {
    // if the file didn't open, print an error:
    Serial.println("Error Reading the File");
#endif
}

}
function GetArduinoInputs()
{
    nocache = "&nocache=" + Math.random() * 1000000;
    var request = new XMLHttpRequest();
    console.log(request);
    request.onreadystatechange = function()
    {
        console.log(this);
        if (this.readyState == 4) {
            if (this.status == 200) {
                if (this.responseXML != null) {
                    // extract XML data from XML file (containing switch states
                    document.getElementsByTagName("lamp")[0].innerHTML =
                        this.responseXML.getElementsByTagName("lamp")[0].innerHTML;
                    document.getElementsByTagName("kosteus")[0].innerHTML =
                        this.responseXML.getElementsByTagName("kosteus")[0].innerHTML;
                    document.getElementsByTagName("kosteus1")[0].innerHTML =
                        this.responseXML.getElementsByTagName("kosteus1")[0].innerHTML;
                    document.getElementsByTagName("kosteus2")[0].innerHTML =
                        this.responseXML.getElementsByTagName("kosteus2")[0].innerHTML;
                    document.getElementsByTagName("time")[0].innerHTML =
                        this.responseXML.getElementsByTagName("time")[0].innerHTML;
                    document.getElementsByTagName("date")[0].innerHTML =
                        this.responseXML.getElementsByTagName("date")[0].innerHTML;
                    document.getElementsByTagName("motor")[0].innerHTML =
                        this.responseXML.getElementsByTagName("motor")[0].innerHTML;
                    document.getElementsByTagName("pump")[0].innerHTML =
                        this.responseXML.getElementsByTagName("pump")[0].innerHTML;
                    if (parseFloat(this.responseXML.getElementsByTagName("motor")[0].innerHTML) == 1)
                    {
                        document.getElementsByTagName("motor")[0].innerHTML =
                            "ON"
                    }
                    else{
                        document.getElementsByTagName("motor")[0].innerHTML =
                            "OFF"
                    }
                    if (parseFloat(this.responseXML.getElementsByTagName("pump")
                        [0].innerHTML) == 1)
                    {
                        document.getElementsByTagName("pump")[0].innerHTML =
                            "ON"
                    }
                }
            }
        }
    }
}
else{
    document.getElementsByClassName("pump")[0].innerHTML = "OFF"
}
}
}
request.open("GET", "inputs" + nocache, true);
request.send(null);
setTimeout('GetArduinoInputs()', 10000);
}

setInterval(function getLinechart()
{
    nocache = "&nocache=" + Math.random() * 1000000;
    var request = new XMLHttpRequest();
    console.log(request);
    request.onreadystatechange = function()
    {
        if (this.readyState == 4) {
            if (this.status == 200) {
                if (this.responseText != null) {

                    dataSeries = this.responseText.split("/")[0].innerHTML;

                    var res = dataSeries.split(";");

                    console.log(res);

                    for (var i = 0; i < (res.length-1); ++i)
                    {
                        var temp = res[i].split(",");
                        tempt[i] = parseFloat(temp[0]);
                        tempH[i] = parseFloat(temp[1]);
                        tempC[i] = temp[2];
                    }

                    console.log(tempt);
                    console.log(tempC);
                    console.log(tempH);
                } else {
                    request.open("GET", "linechart" + nocache, true);
                    request.send(null);
                },50000);
            }
        }
    }
}
function GetIO()
{
    nocache = "&nocache=" + Math.random() * 1000000;
    var request = new XMLHttpRequest();
    console.log(request);
    request.onreadystatechange = function()
    {
        console.log(this);
        if (this.readyState == 4) {

if (this.status == 200) {
    if (this.responseText != null) {
        document.getElementById("LED1").innerHTML =
            document.getElementById("LED1")[0].innerHTML;
        document.getElementById("LED2").innerHTML =
            document.getElementById("LED2")[0].innerHTML;
    }
}

    request.open("GET", strLED1 + strLED2 + nocache, true);
    request.send(null);
}

function GetButton1() {
    if (LED1_state == 1) {
        LED1_state = 0;
        strLED1 = "&LED1=0"
    } else {
        LED1_state = 1;
        strLED1 = "&LED1=1"
    }

    GetIO();
}

function GetButton2() {
    if (LED2_state == 1) {
        LED2_state = 0;
        strLED2 = "&LED2=0"
    } else {
        LED2_state = 1;
        strLED2 = "&LED2=1"
    }

    GetIO();
}

</script>
<style>
body {
    margin: auto;
    padding: 0px;
    text-align: center;
    width: 100%;
    float: right;
}

h1 {
    text-align: center;
    font-family: Arial, "Trebuchet MS", Helvetica, sans-serif;
}

h2 {
    text-align: center;
    font-family: Arial, "Trebuchet MS", Helvetica, sans-serif;
}

.box {
    margin: 5px auto 5px auto;
    padding: 0 10px 0 10px;
text-align: center;

}

.IO_box {
    margin: 5px auto auto auto;
    border: 1px solid black;
    padding: 0px 0px 0px 10px;
    text-align: left;
    display: inline-block;
    width: 200px;
}

.IO_box1 {
    margin: 5px auto auto auto;
    border: 1px solid black;
    padding: 0 10px 0 10px;
    width: 190px;
}

.box_button {
    margin: 20px auto 5px auto;
    border: 1px solid black;
    padding: 0 10px 0 10px;
    width: 150px;
}

.custom-button-style1{
    margin: 5px 5px 5px 5px;
    padding: 10px 15px;
    color: #FFFFFF;
    background-color: #555;
    border: 0 none;
    -webkit-border-radius: 3px;
    -moz-border-radius: 3px;
    border-radius: 3px;
    cursor: pointer;
}

.small_text {
    font-size: 70%;
    color: #737373;
}

#chart_div {
    margin: 5px auto 5px auto;
    height: 400px;
    width: 600px;
    padding: 10px 15px;
    border: 1px solid black;
}

input[type="button"] {
    margin: 5px auto 5px 5px;
    padding: 0 10px 0 10px;
    width: 50px;
    height: 30px;
    display:inline;
}

h4 {
    margin: auto auto 20px;
    padding: 0 10px 0 10px;
    width: 50px;
    height: 30px;
APPENDIX 5/5

display:inline;
}
.motor{
  float:right;
  margin: auto 30% auto auto;
}
.pump{
  float:right;
  margin: auto 30% auto auto;
}
.lampa
{
  float:right;
  margin: auto 30% auto auto;
}
.kosteus
{
  float:right;
  margin: auto 30% auto auto;
}

h1
{
  margin: 2% auto auto auto;
  padding: 0 10px 0 10px;
}

</style>
</head>

<body onload="GetArduinoInputs(); lineChartBuild()"

<h1>Arduino Greenhouse</h1>
<div class="box">
  <span class="time">...</span>
  <span class="date">...</span>
</div>

<div class="IO_box">
  <h2>Soll Moisture</h2>
  <p>Kosteusanturi 1: <span class="kosteus1">...</span><%p>
  <p>Kosteusanturi 2: <span class="kosteus2">...</span><%p>
</div>

<div class="IO_box">
  <h2>Temperature</h2>
  <p>Humidity: <span class="lampa">...</span><p>
  <p>Temperature: <span class="kosteus">...</span><p>
</div>

<div class="IO_box">
  <h2>Motor Status</h2>
  <p>Motor <span class="motor">...</span><p>
  <p>Water Pump <span class="pump">...</span><p>
</div>

<div class="IO_box1">
  <h4>Motor</h4><h4>Pump</h4>
  <input type="button" id="LED1" value="OFF"
 onclick="GetButton1()" />

<input type="button" id="LED2" value="OFF" onclick="GetButton2()" />
</div>
</div>

<canvas id="chart_div" height="400" width="600"></canvas>
</div>

<script>
var startingData = {
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],
datasets: [
{
fillColor: "rgba(220,220,220,0.2)",
strokeColor: "rgba(220,220,220,1)",
pointColor: "rgba(220,220,220,1)",
pointStrokeColor: "#fff",
data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
},
{
fillColor: "rgba(151,187,205,0.2)",
strokeColor: "rgba(151,187,205,1)",
pointColor: "rgba(151,187,205,1)",
pointStrokeColor: "#fff",
data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
}
];

var canvas = document.getElementById("canvas")
var ctx = canvas.getContext('2d')
var myLineChart = new Chart(ctx).Line(startingData, {scaleShowLabels: true});

setInterval(function(){
for(var i = 0; i<24; i++){
myLineChart.scale.xLabels[i] = tempc[i];
myLineChart.datasets[0].points[i].value = tempt[i];
myLineChart.datasets[1].points[i].value = tempf[i];
}
myLineChart.update();
}, 5000);

</script>
</body>
</html>