Final Thesis
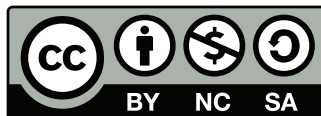
WTherm

Web-connected smart thermostat

Author:
Niek Blankers (A6b)

Supervisor:
Dolf Breederveld

January 30, 2015

Abstract

This thesis delineates the research and design process involved in the development of the WTherm, a smart thermostat. The thermostat was installed in a single household. A thermal model of the living room in question is used to calculate the amount of time until a certain target temperature will be reached. This calculation is used to start heating at exactly the right time to reach the targets defined in a schedule that's programmable through a web interface. As a result, the WTherm saves energy in comparison to a traditional room thermostat.

The calculated annual saving in this particular case exceeds € 150, which is more than three times the total cost of the materials used.

Contents

Preamble

The inspiration for this project originated from my father's request to develop a basic thermostat he could remotely program through a web interface. After a couple of weeks of research and development, this thermostat was installed in his house and dubbed the WTherm alpha project[1].

After installing the Nest smart thermostat[2] in my mother's house, I decided to revisit the WTherm project and to try to develop it further into a smart thermostat.

I am very thankful to Mr. Breederveld and Albert Vos, as I greatly appreciate their assistance over the course of this project. I would also like to thank the LeMaker[3] team for donating a Banana Pi after reading my research plan.

---

[1] https://github.com/NiekProductions/WTherm-alpha/
[2] https://nest.com/
[3] http://www.lemaker.org/

4

Introduction

In practically every house with a central heating system you will find a thermostat. The very first electric thermostat (or "temperature regulator") was patented in 1893 by Albert Butz. It described a bimetallic strip that, when bent by a decrease in temperature, would make electrical contact and power a motor which would in turn adjust the dampers of a furnace[1].

Butz's invention was revolutionary, but it lacked a feature before making it into the households of many. Nowadays, most thermostats feature a programmable schedule which can be used to set daily times at which the heating should turn on.

A recent development in the field of central heating control has been the introduction of smart thermostats. These thermostats offer Wi-Fi or wired connectivity for remote control and claim to improve energy efficiency. Being one of the first smart thermostats on the market in 2011, the Nest thermostat[4] aims to save energy by learning when a user leaves the house and automatically turning off the heater. The Nest Thermostat also learns the time required to heat up the house in order to show the amount of time necessary to reach the temperature desired by the user[2].

Heating a house consumes a substantial amount of energy. The speed at which a central heating system can heat up a room varies on a day-to-day basis. If the time during which the central heating is on can be reduced by even half an hour every day, a significant amount of the annual energy usage can be saved. This is the approach to energy conservation used in this project.

Research objective

The objective of this project is to design and develop a smart thermostat that saves heat by predicting or calculating the time it takes a room to heat up to a certain temperature. The thermostat must use this calculation to precisely follow a weekly schedule that can be programmed through a web interface. The scope of this project will be limited to a single situation in which a district heating unit is used to heat a living room.

The following research topics can be distilled from the main objective:

---

[4]https://nest.com/

- How can the heating time be predicted?

- What hardware and software is necessary to run the thermostat?

- How can a web interface be implemented?

- What safety features are required?

- How much energy does this solution save compared to a traditional room thermostat?

Additionally, the user interface needs to meet the following demands:

- A secure implementation of a login system for user authentication.

- The option to program a weekly schedule.

- The option to set a manual temperature that overrides the schedule.

- A page to view past temperatures through charts.

- A page to change user data.

Hypothesis

The hypothesis is that the WTherm saves heat in comparison with a clock thermostat set to start heating at a fixed time every morning.

Project summary

The first step to meet the aforementioned objective is to find a way to predict the time it will take the temperature of the living room to meet a target. This process is described in part I. In the first chapter, heat flow is explained through formulas. In the second chapter these formulas are applied to the situation and used to form a thermal model of the living room. This model is also verified and a way is described in which the model can be used to predict the time it will take the temperature in the living room to reach a certain level, solving the first part of the research objective.

Part II of this thesis deals with the construction of the thermostat. The necessary hardware and software to run the thermostat is discussed.

The database structure and interaction and the way passwords are securely stored is reviewed in chapters 6 to 8, while chapters 9 and 10 deal with the hardware level of the thermostat.

Chapters 11 and 12 give an overview of the code that runs the WTherm and its web interface, as well as an insight into the function of each file in the source code. Finally, the implemented safety features are discussed in chapter 13.

In part III of this thesis the hypothesis is tested by comparing the WTherm to a traditional clock thermostat over the course of a single week in the winter.

PART I

Predicting the heating time

This part describes how to calculate the amount of time it will take to heat up the room to a certain temperature. In the first chapter, the formulas describing how heat flows in a system will be explained. Next, these formulas will be applied to the situation in question and they will be used to form a thermal model of the living room that can then be used in the thermostat to determine when to start heating.

1 Thermal calculations

Heat capacity (or thermal capacity) is a constant that describes the ratio between thermal energy and temperature.It is expressed in Joules per Kelvin, indicating the amount of heat that results in a temperature change of a single Kelvin or degree Celsius[3]. Calculation of the heat capacity is done using the following formula:

$$C = \frac{Q}{T} \tag{1.1}$$

With C being the heat capacity in Joule per Kelvin, Q the added thermal energy in Joules and T the difference in temperature in Kelvin or degrees Celsius.

However, the heat capacity determined by the above-mentioned formula can only be used in a system where heat cannot be transferred away from the system. In practice, this is highly unlikely to occur since perfect isolation on a small scale is impossible. The heat that is transferred away needs to be taken into account. Heat can be transferred in three ways:

- Conduction

- Convection

- Radiation

Conduction is heat transfer inside a material. Heat flows from particles with more kinetic energy to particles with less kinetic energy[4]. Heat transfer through conduction can be described in a linear formula as follows:

$$P = A \cdot U \cdot (T_i - T_o) \tag{1.2}$$

P being the conducted power in Watts, A the surface area in m^2 of the material through which the heat is conducted, U the heat transfer coefficient in $W/(m^2K)$ and $(T_i - T_o)$ the difference between inside and outside temperature[5].

"Convection is heat transfer by mass motion of a fluid such as air or water when the heated fluid is caused to move away from the source of heat, carrying energy with it."[4]. Inside for example a system with a heat source and a medium, this would be the medium that expands and changes in density as it's heating, causing a flow that transports energy.

Heat transfer through radiation is caused by electromagnetic waves, most of which are in the infrared region. According to the Stefan-Boltzmann law[6], radiated heat uses the form:

$$Q = A \cdot \sigma \cdot T^4 \cdot t \tag{1.3}$$

In this formula, Q is the transferred heat in Joules, σ the Stefan-Boltzmann constant ($5,6703 \times 10^{-8}$W/m^{2}K^4 in the case of a blackbody, less depending on the material), A the area in m^2, T the absolute temperature in Kelvin and t the total time in seconds.

This concludes the description of the formulas that will be used in the specific situation set out in the next chapter.

2 Creating the thermal model of the room

The previous chapter introduced the formulas that describe the thermal energy in a system and the heat flow in and out of the system. In this chapter, these formulas will be applied to the situation in question, in this case a living room. Three manners of heat flow were described, namely conduction, radiation and convection. Since the living room is mostly isolated against convection to the outside, the effect of convection will not be noticeable. Radiation only significantly occurs at very high temperatures, so it is also out of the question.

Conduction is the only important form of heat transfer in this case. This means that the heat flowing out of the living room can be described in a formula similar to (1.2), replacing the area and U-value with an overall heat transfer coefficient $C_{transfer}$ (or C_t) in W/K:

$$C_t = \frac{P}{dT} = \frac{P}{T_i - T_o} \quad \text{provided that dT} \neq 0 \tag{2.1}$$

The First Law of Thermodynamics states that energy is saved; it can be neither created nor destroyed[7]. According to this principle, one can deduce that when x amount of heat is added to a system and y amount escapes, the amount of heat left in the system is x-y. If the thermal profile of the system is considered according to this principle, one can state that the heat contained in the room is equal to the difference between the total heat added by the heating system and the heat loss:

$$Q_{heat} = Q_{in} - Q_{loss} \tag{2.2}$$

Q_{loss} can be calculated from the C_t value (2.1) and Q_{in} from the central heating power. The U value in W/K can be determined by keeping the room at a certain temperature and measuring the required power. This temperature has to exceed the outside temperature as the amount of heat flowing into the room can't easily be measured. If the thermal profile is considered in time steps of dt and temperature steps of dT, one can state that:

$$Q_{heat} = C \cdot dT \tag{2.3}$$

$$Q_{in} = P \cdot dt \tag{2.4}$$

$$Q_{loss} = C_t(T - T_o) \cdot dt \tag{2.5}$$

Now according to (2.2) through (2.5) one can form a differential equation as follows:

$$\frac{dT}{dt} = \frac{P - C_t(T - T_o)}{C} \tag{2.6}$$

This differential equation contains a linear component, making it unpractical to solve. A better approach would be to create a model. The term 'model' refers to a simplified version of the target system, including only the relevant aspects.[8] The thermal model will create the temperature curve by repetitively calculating dT for small steps of dt. dT can be defined as:

$$dT = \frac{Q_{in} - Q_{loss}}{C} = \frac{P \cdot dt - C_t(T - T_o) \cdot dt}{C} \tag{2.7}$$

With Q_{in} and Q_{loss} being defined in (2.4) and (2.5) on page 10. The temperature curve can then be calculated:

```
1  # Starting values
2  t = 0 s
3  dT = 0 K
4  T = .. degC
5  To = .. degC
6
7  # Constants
8  Ct = .. W/K
9  P = .. W
10 C = .. J/K
11 dt = 60 s
12 t_max = 3600 s
13
14 # Loop
15 while t < t_max:
16   Qin = P * dt
17   Qloss = Ct * (T-To) * dt
18   dT = (Qin - Qloss)/C
19   T = T + dT
20   t = t + dt
```

Listing 2.1: (pseudocode) calculating the temperature curve

In conclusion, the constants that need to be determined to create the thermal model are:

P The heating power in Watts

$\mathbf{C_t}$ The heat transfer coefficient in W/K

C The room's heat capacity

2.1 Calculating P, $\mathbf{C_t}$ and C

To complete the thermal model described in the previous section, the heating power, heat transfer coefficient and the heat capacity will need to be determined. This will be done according to a single measurement, where the room will first be heated up to 19.5°C and then maintained at this temperature by a traditional room thermostat.

Calculating the heating power is fairly simple. The heater will constantly be powered on when the temperature hasn't reached the target of 19.5°C yet, so if the time it takes to heat up as well as the heat spent in this time are measured they can be used to calculate the average heating power.

The heat transfer coefficient C_t will be calculated from the measurements when the temperature has already reached the target. According to (2.2), if

Q_{heat} remains constant, the added heat Q_{in} is equal to the lost heat Q_{loss}. Any heat spent to keep the room temperature constant is heat that's being conducted out of the room. The added energy divided by the amount of time during which the temperature is kept constant will result in the total power used in Watts. Divide this by the difference in inside and outside temperature following (2.1) and the resulting constant is C_t.

C_t can then be used to calculate the heat capacity C. For every interval between measurements, the lost heat Q_{loss} is determined by multiplying C_t by the difference between the inside and outside temperature. The total added heat Q_{in} can then be subtracted by this heat loss, resulting in the net added heat Q_{net}. The heat capacity C is equal to the net heat used to increase the room temperature divided by the total temperature rise conforming (1.1).

In conclusion, to actually create the thermal model, it is first of all necessary to calculate the heating power P, heat transfer coefficient C_t and heat capacity C. The room will be heated to 19,5°C and kept at that temperature for several hours. The added heat will be measured, as well as the inside and outside temperatures.

2.1.1 Reading added heat from the heat meter

To create a model of the temperature in the living room, a way to measure the heat added to the system is necessary. The central heating consists of a district heat exchange unit with an external heat meter.

The heat meter is a Landis & Gyr Ultraheat 2WR5. The 2WR5 can be read out directly from the LCD display, which displays the accumulated quantity of thermal energy in gigajoules with three decimal numbers. 0,001 GJ is equal to 1 MJ, which is the same as $3{,}6^{-1}$ kilowatt hours, or about 278 Wh.

Figure 2.1: 2WR5 heat meter
Source: tls.se

To log the added heat, an IP camera[1] was mounted directly in front of the heat meter, with a small light next to it illuminating the screen. A snapshot can be fetched by sending a request to a HTTP URL, in this case http://<camera-IP>/snapshot. cgi. A shell script was used to run on a remote linux machine every five minutes:

---

[1]A security camera connected to the network

```sh
#!/bin/sh
wget http://<camera-IP>/snapshot.cgi --user=admin --password=
    password -O snaps/snapshot-`date +%Y-%m-%d-%H%M%S`.jpg -T 60
```

Listing 2.2: Downloading and renaming a snapshot

This script resulted in pictures like figure 2.2, which were subsequently copied into a spreadsheet.

Figure 2.2: Sample picture from IP camera

2.1.2   Temperature readings

Temperature measurements from inside and outside the living room are also needed. The house already has two wireless temperature sensors installed. One is located in the living room, while the other is outside of the house. These sensors are connected to a Homewizard[2], an internet-connected smart-home device which can be accessed by requesting different URLs. To obtain the inside and outside temperature, an HTTP request is made to http://<homewizard-IP>/<password>/telist[9]. The WTherm alpha project was used in this case, as it already logs the inside and outside temperatures every five minutes. The MySQL database containing the measurements was later exported into a spreadsheet.

---

[2]http://www.homewizard.nl/

2.1.3 Obtained measurements

A graph of the data obtained from the heat meter and temperature sensors looks like this:
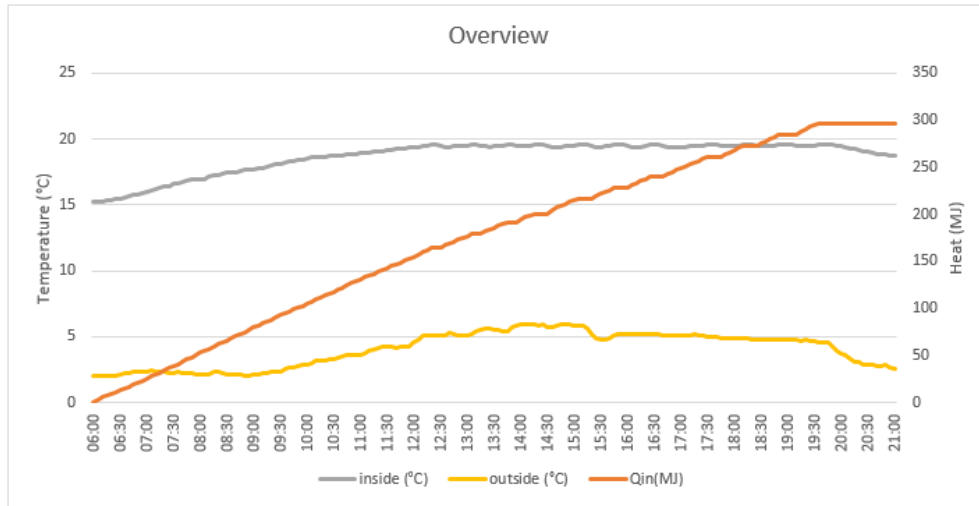

Figure 2.3: Raw temperature/heat measurements

Between 06:00 and 12:00, the room was heated from 15,3°C to 19,5°C. This temperature was maintained until 19:30.

The total energy used while heating up is 155MJ. Converting this to kWh and dividing it by 6 hours results in an average heating power P of 7,2kW.

Keeping the room at 19,5°C from 12:30 to 19:30 required 36,39kWh, or 5,2kW. The average difference in inside and outside temperature. Following (2.1), the heat transfer coefficient C_t ends up at approximately 364 W/K.
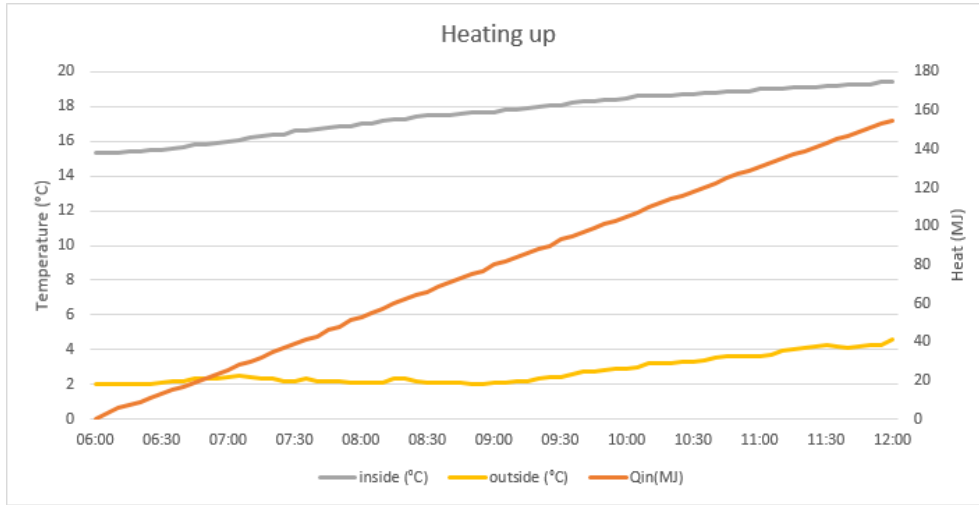
Figure 2.4: Measurements while heating

Using C_t, the amount of heat lost in every five minute interval between measurements can be calculated. Subtracting the added heat by the lost heat as described at the start of this section on page 11 results in Q_{net}:
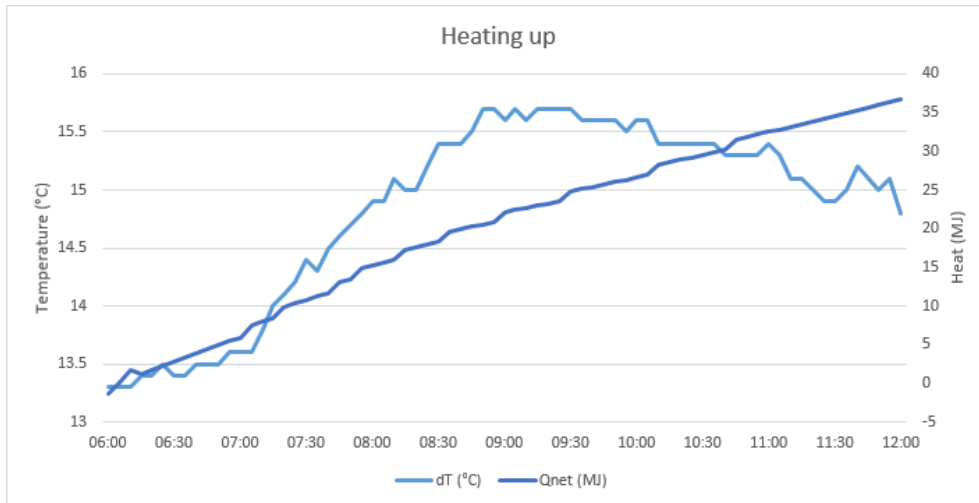

Figure 2.5: Calculated Q_{net} and dT

The net added heat when heating up was roughly 37MJ. Dividing this by the temperature rise of 4,1°C according to (1.1) results in a heat capacity C

of about $8,978 \times 10^6$ J/K. All of the measurements and calculations can be found on GitHub using the link found on page 43.

The necessary constants for the model listed on page 11 have now been determined. In the next section, the resulting model will be verified against the measurements.

2.2 Verifying the thermal model

To validate this model, it can be compared to the measurements. Since the heating time has to be predicted, the outside temperature is assumed to be constant. In this case, an average outside temperature of 2,75°C during the measurements is used. The starting temperature T is 15,3°C. The following graph was drawn using the code from listing 2.1. A dt of 5 minutes and a total time of 6 hours was used:
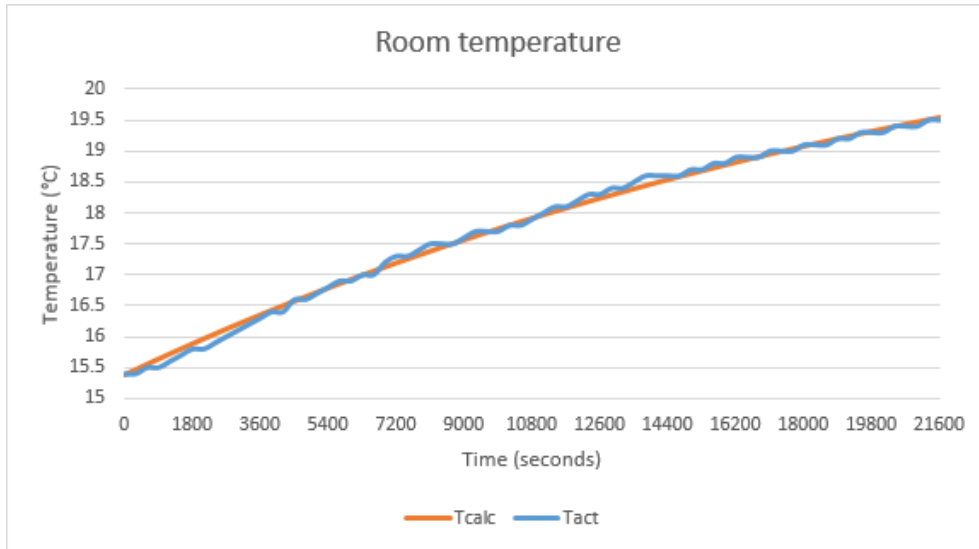

Figure 2.6: Testing the temperature model with the measurements superimposed.

$Tcalc$ in this chart is the temperature as calculated by the model, and $Tact$ the actual measured temperature for comparison. The calculated temperature reaches a maximum of 19.5°C, which corresponds with the temperature measured after 6 hours.

2.3 Using the thermal model

In the previous sections, the model was determined and tested to match to the actual situation. Using this model to predict the time it takes the tem-

16

perature to reach a certain target can be done using a while loop that steps through the model for intervals of dt. The following code is an adaptation of listing 2.1 from page 11 that can be used in the thermostat:

```
1  # Starting values
2  t = 0 s
3  dT = 0 K
4  T = 15 degC
5  Target = 20 degC
6  To = 5 degC
7
8  # Constants
9  Ct = 364 W/K
10 P = 7176 W
11 C = 8978053 J/K
12 dt = 60 s
13
14 While T < Target:
15    Qin = P * dt
16    Qloss = Ct * (T-To) * dt
17    dT = (Qin - Qloss)/C
18    T = T + dT
19    t = t + dt
20
21 Required time = t
```

Listing 2.3: (pseudocode) calculating the heating time from 15 to 20 degrees with a constant outside temperature of 5 degrees

Now that the model has been created and approved, the next part will describe how it was integrated into the thermostat.

PART II

Constructing the thermostat

This part will describe how the thermostat itself was constructed. First the hardware choice will be justified.

3 Hardware choice

Three hardware platforms were considered to run the thermostat, namely:

- Arduino

- Raspberry Pi

- Banana Pi

Arduino Uno

The Arduino Uno is an open-source microcontroller platform developed by the Italian Arduino company. It comes with an easy-to-use IDE for writing software for the board. The board features an 8-bit Atmel ATmega328p microprocessor running at 16 MHz with 32 kilobytes of programmable flash, 1K of EEPROM, 2K of SRAM and 23 I/O lines, of which 6 are PWM channels and 6 are ADC inputs[10]. The Arduino is programmed using C++ from the Wiring-based IDE, with a number of code libraries to simplify programming[11].

Figure 3.1: Arduino Uno
Source: arduino.cc

A very early prototype of the WTherm alpha was run on an Arduino. The Arduino doesn't have built in networking, so an external Ethernet module was connected. Since no TCP stack was already built into the Arduino, the networking had to be programmed on a very low level. Combined with the limited memory available this makes it unsuitable to run a reasonable website and database. The Arduino is great for direct low-level access to hardware, but the disadvantages outweigh this advantages which is why I chose not to run the thermostat on an Arduino.

Raspberry Pi

The Raspberry Pi is a very popular low cost credit-card sized computer. It's based around a 700MHz ARMv6 processor with 500MB of RAM. It features on-board networking, GPIO, video output, USB and a low power consumption. The Raspberry Pi is able to run a variety of linux distributions, including debian and fedora[12].

Figure 3.2: Raspberry Pi

Source: wikimedia.org

The Raspberry Pi almost meets the demands. It can run a web server and it has built-in networking. It does however run off an SD card. The many read/write-operations of a the database can cause the flash storage in an SD card to become corrupted, which is an issue that I encountered in the WTherm alpha project.
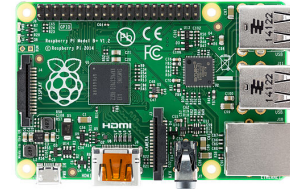
Banana Pi

The Banana Pi is a hardware platform similar to the Raspberry Pi. Instead of a single-core ARMv6 processor, it uses a dual-core ARMv7 running at 1 GHz with 1GB of RAM. This increase in processing power results in higher web server performance. In addition to GPIO, USB and networkingit also has a Serial ATA connector. This means that it's able to run off an external hard drive, making it more suitable for running a database[13]. This is why I chose the Banana Pi for this project.

Figure 3.3: Banana Pi

Source: bananapi.org

4 Web server

Now that the hardware choice has been made, the next step is to choose a web server to run the code on.

Apache

Apache is a very popular open-source web server that runs on nearly every modern operating system. Apache has played a key role in the early development and growth of the World Wide Web. [14].

Apache is a process-based server. This means that it has to spawn new processes for each task, making it very CPU and memory-intensive. Since the hardware platform for this application has a limited amount of memory and processing power, Apache would not perform very well[15].

Nginx

Nginx (pronounced "engine-x"), just like Apache, is a free, open-source web server. *"Nginx is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption."*[16]

Unlike Apache, nginx is an event-based web server. It operates asynchronously, and handles requests in a low number of threads. Nginx consumes less RAM than Apache, has better caching and serves web pages more quickly[15]. This is why nginx was chosen to run the thermostat.

5 Programming language

Now that the hardware platform and the web server have been chosen, all that's left is to choose a programming language. In this chapter, two languages will be compared: Python and PHP.

Python

Python is a widely used open source high-level language that's supported by large developer communities. It uses strict indentation enforcement and aims to be the most readable programming language. Since the Python interpreter supports modules and packages, only the necessary modules have to be imported. This results in less runtime than comparable code in a different programming language.[17]

PHP

The WTherm alpha project's source code was written in PHP. Much like Python, PHP[1] is an open-source, interpreted, high level language. It is especially suited for web development. PHP is a server-side language, which means that the code is executed on a server, generating output (normally HTML) which is then sent to the client. Unlike the case with the client-side language Javascript, the client only ever gets to see the code's output, not the underlying code itself. PHP is a very simple language, which is why it's excellent for beginners[18]. Because of my previous experience and knowledge with PHP, it's used for this project.

6 Database structure

The thermostat will be running from a MySQL database. MySQL is an open-source relational database management system. It uses SQL[1] to manage data[19]. This chapter will describe the database structure.

---

[1]a recursive acronym for "PHP: Hypertext Preprocessor"
[1]Structured Query Language

Status

Since the thermostat script will run every 5 minutes, a status table in the database is used to store the last measurements and state. This means that the user interface can request the last measurements without having to obtain direct access to the sensor outputs. This table consists of the following columns:

T Float - The inside temperature

T_o Float - The outside temperature

T_target Float - The target temperature

Heating Boolean - Whether the heater is on

Override Boolean - Whether manual temperature is enabled

Last_update Timestamp - The last time the script has successfully ran

Now the most recent measurements and status are stored in a central table, accessible to both the thermostat script and the client web interface.

Schedule

The thermostat will need a schedule, containing the target temperatures and the time/day when this temperature needs to be reached. Since the thermostat uses a weekly schedule, the table structure reads as follows:

T_target Float - The target temperature

time Varchar - The time (00:00-23:59)

day Integer - The weekday (1-7)

A primary key constraint between **time** and **day** was set up to prevent duplicate entries.

Log

This table is used to store measurements, which can be used

Time Timestamp - The time

T Float - The inside temperature

T_target Float - The target temperature

T_o Float - The outside temperature

Heating Boolean - Whether the heater is on

The **Time** column is made a primary key to prevent duplicate entries.

Users

This database table is used to store login users and their password hashes.

username Varchar - A username

password Varchar - The user's password

The **username** column was defined as a primary key to prevent duplicate users.

The SQL code for this database can be retrieved from GitHub using the link found on page 43.

7 Preventing SQL injection

Traditional PHP code for executing a MySQL query to validate a login would be similar to the following:

```php
<?php
// Get the user input
$user = $_POST['username'];
$pass = $_POST['password'];

// Execute the query
$result = mysql_query('SELECT * FROM login WHERE username='.
    $user.' AND password='.$pass);
?>
```

Listing 7.1: Example PHP MySQL login code

In this case $user and $pass are user submitted values from a login form. The code would work fine, until the user submits something like `' OR 1=1;--` as their username. This would transform the query in listing 7.1 into `SELECT * FROM login WHERE username='' OR 1=1;--`. This query would always return a result, since the integer 1 is always equal to itself. The `;--` ends a query, so the entered password is irrelevant.

Prepared statements are used to prevent this type of malicious SQL injection. A prepared statement version of listing 7.1 would look like this:

```php
1  <?php
2  // Get the user input
3  $user = $_POST['username'];
4  $pass = $_POST['password'];
5
6  // The query, note how $user and $pass have been replaced by :
       user and :pass
7  $sql = "SELECT * FROM login WHERE username=:user AND password=:
       pass";
8  // Prepare the query
9  $stmt = $db->prepare($sql);
10 // Execute the query, replacing :user with $user and :pass with
       $pass
11 $stmt->execute(array(
12   ":user" => $user,
13   ":pass" => $pass
14 ));
15 // Fetch the results
16 $user = $stmt->fetch();
17 ?>
```

Listing 7.2: Safe PHP MySQL login code

In this snippet, the SQL query is first prepared (hence the 'prepared statement') and then executed. The user input cannot be abused, if ' OR 1=1;-- were to be entered into the username field, it would only return a result for a username that is equal to ' OR 1=1;--. Prepared statements can also be used to execute the same statement repeatedly with high efficiency.

8 Password hashing

As a form of protection, the WTherm will have a user management system. This requires user login data to be stored in a database. Storing a plain text password in the database is a bad idea. If a malicious party gets access to the database, either by hacking or using a flaw in the website they have direct access to the passwords.

One way of preventing direct access to the password is by using a one-way hashing algorithm such as MD5 or SHA-2. Upon registration, the password would be sent through this algorithm and the password hash will be saved to the database. When a user logs in, the entered password is also sent through the hashing algorithm and then compared to the password in the database. If they match, the login is successful.

Plain hashing is also not the most secure way to save the password. Being a

one-way algorithm, a rainbow table can be used. A rainbow table is a list of common words and their equivalent hashes. This table can be compared to the hashed passwords in the database, and used to get the original password. Hashed passwords are also vulnerable to brute force attacks, in which the attacker repetitively hashes random strings to see if the hash matches to the password hash.

Therefore, a more secure way of storing the password was used for the login system at hand. Before hashing, a *salt* (a random string) is added to the password. This results in a hash that's unique, even for two users with the same password. Both the hash and the salt are stored to the database. When the user tries to log in, the salt is brought in from the database. The entered password is then hashed using the same salt, and compared to the hash stored in the database. *Salting* a password prevents the use of rainbow tables. It doesn't give total security, as an attacker can still brute-force the password. Salted hashing does make it incredibly difficult to brute force the password, provided that the user chooses a complex or long password.

9 Reading from the temperature sensors

The WTherm needs a way to access readings from the temperature sensors. These readings are obtained using the same approach as described in section 2.1.2. A request is made to the HomeWizard through the cURL library in PHP.

10 Interfacing with the heating

The heating is controlled by a water flow valve. This valve is either closed (when no power is applied to it), or completely opened when 24 Volts AC is applied to it. To control this valve from one of the Banana Pi's GPIO pins, a relay was used. A transistor was used to switch the relay, since the Banana Pi can't sink the required current to switch the relay. A reversed diode is soldred across the relay to absorb the current from the collapsing magnetic field when the relay is switched off, preventing it from damaging the transistor or Banana Pi.
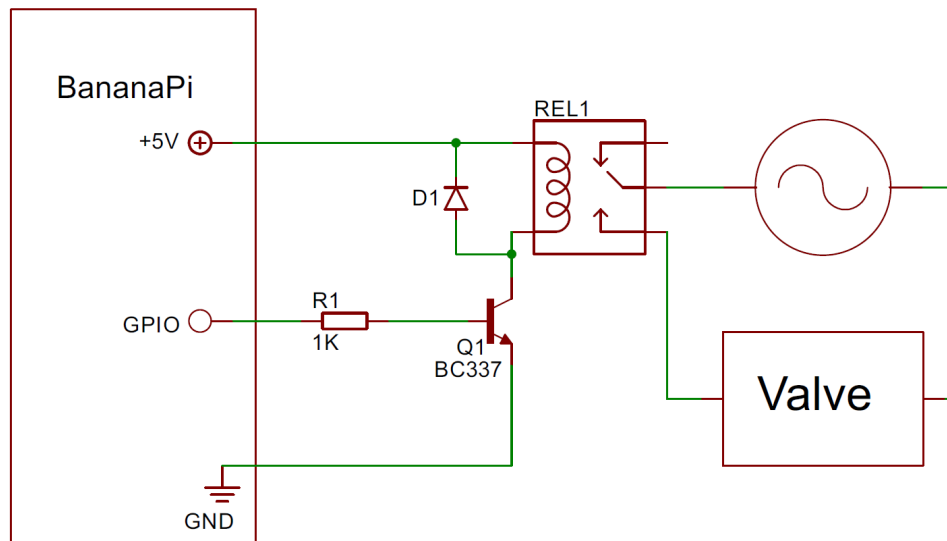
Figure 10.1: Heat valve control schematic

The standard command line gpio utility built into the Banana Pi is used to control the GPIO pin in the following way:

```php
// Switching on the heater
shell_exec('/usr/local/bin/gpio write [gpio pin] 0');

// Switching off the heater
shell_exec('/usr/local/bin/gpio write [gpio pin] 1');
```

Listing 10.1: Controlling GPIO in PHP

11 Layout

The WTherm runs a web interface for user interaction. This chapter will depict how the layout for this user interface was assembled together.

11.1 HTML and CSS

Bootstrap[1] is an open source front-end framework designed to make web-development a lot faster and easier. Bootstrap's CSS framework was used for the WTherm's web interface.

_____

[1]http://getbootstrap.com/

11.2 Template parser

By using a template parser, a single HTML file with placeholders can be used for every page. This makes editing easier, and it makes the code appear more organized. A simplified version of the template used for the WTherm looks as follows:

```html
 1 <!DOCTYPE HTML>
 2 <html lang="en">
 3 <head>
 4   <title>WTherm &bull; <?=$this->name?></title>
 5   <link href="layout/bootstrap.min.css" rel="stylesheet">
 6   <link href="layout/layout.css" rel="stylesheet">
 7 </head>
 8 <body>
 9   <?=$PAGE_BODY?>
10 </body>
11 </html>
```

Listing 11.1: simplified HTML template

Notice that the HTML title, menu and body have been replaced by PHP tags. These tags are later replaced by calling the template parser class:

```php
 1 <?php
 2 include('base.php');
 3
 4 $page = new Layout; // Start new layout
 5 $page->startHTML("Home"); // Indicate that HTML code is
      following
 6 ?>
 7
 8 <img src="test.png">
 9
10 <?php
11 $HTML = $page->fetchHTML(); // Fetch the above HTML
12 $page->output($HTML); // Output the layout
13 ?>
```

Listing 11.2: Usage of the template parser class

The example above outputs the template, with "Home" in the title tag and the image *test.png* in the page body.

12 Code overview

Since the total line count for the software is over 1700, it was not included in this document. The final code can be found in the GitHub repository listed on page 43.

In this chapter, the functions of the various scripts are explained. The code is split up into a "Back end" (everything in the *code* folder except the *www* folder) and a "Front end" (everything in the *www* subfolder). The following diagram is an overview of the code structure.
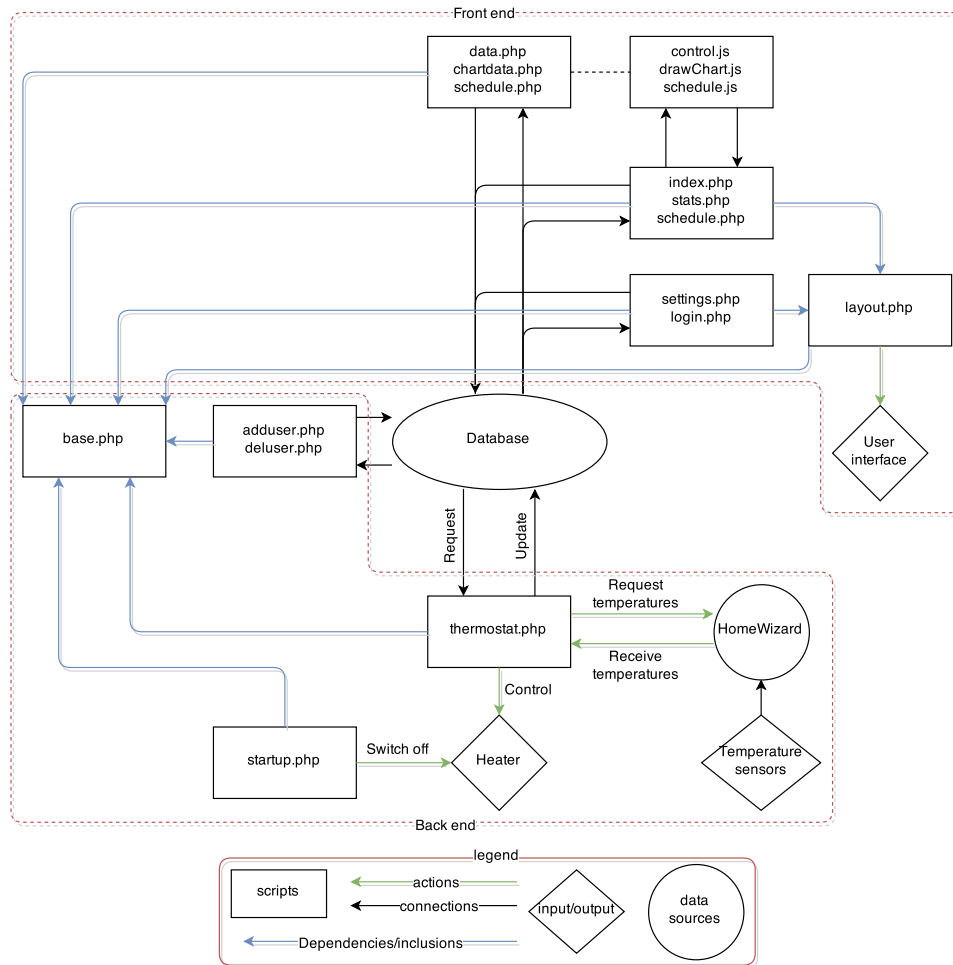

Figure 12.1: Code structure overview

12.1 Back end

Back end source code is inaccessible to users. The source code described in this section runs the low-level functions of the thermostat.

base.php

This file contains global functions, classes as well as configuration variables. This includes the template parser class described in section 11.2, the class that calculates the time it will take to heat up to a certain temperature based on the thermal model and a class that handles database connections. *base.php* is included in nearly every script.

thermostat.php

This script performs the measurements and controls the heater. Its function is described in the following flowchart:
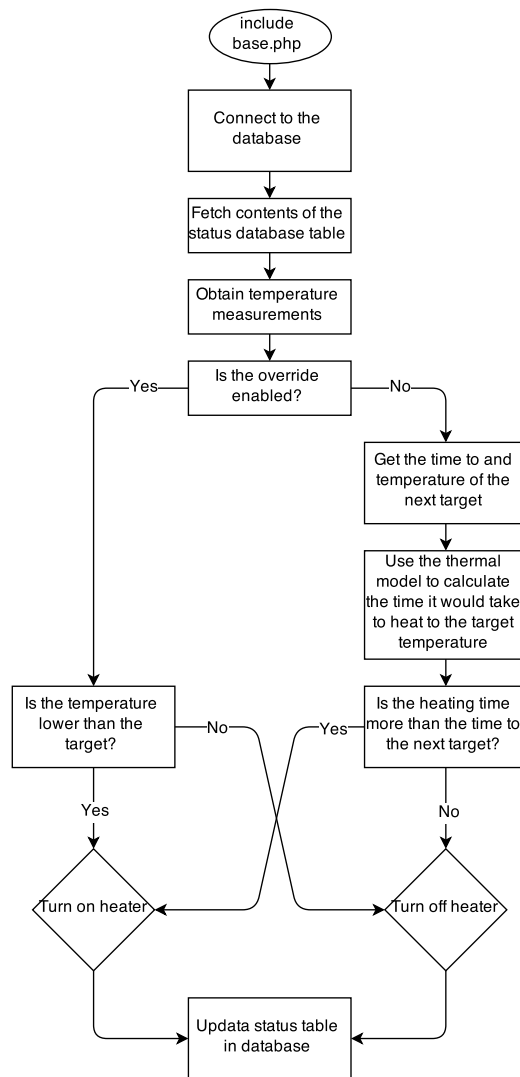
Figure 12.2: Overview of *thermostat.php*

adduser.php and deluser.php

These scripts can be used to add or delete a user through the command line. Command line usage of *adduser.php* is as follows:

```
php5 adduser.php [username] [pass]
```

Listing 12.1: Usage of *adduser.php*

Deleting a user through *deluser.php* is done in the following manner:

```
1  php5 deluser.php [username]
```

Listing 12.2: Usage of *deluser.php*

startup.php

This script is executed on startup and is used to switch off the heater for the safety concerns discussed in the next chapter.
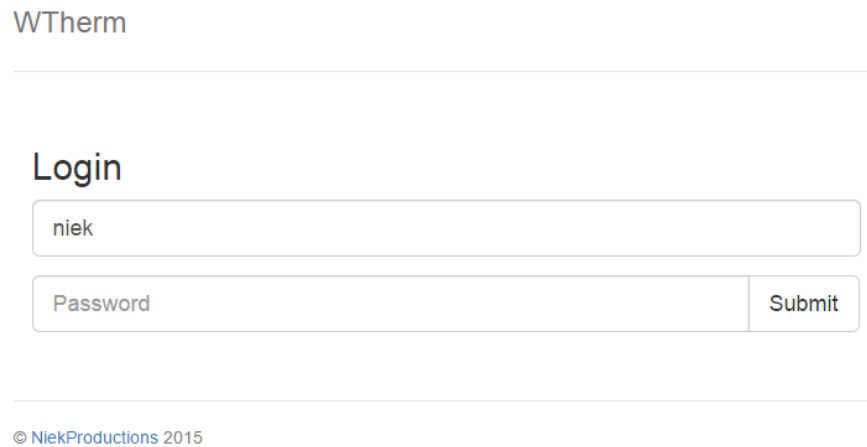
12.2 Front end

The front end consists of every page that is user-accessible.

JavaScript

JavaScript, a programming language that is executed client-side, is used to communicate with the database through a technique called AJAX[1]. The JavaScript code requests data on the background (*asynchronously*) and uses the response to update the page. This means that the page doesn't have to keep refreshing to update, resulting in a better user experience. This technique is also used for user input. A JavaScript function is triggered when the user interacts with the web page, and updates the database through a PHP file.

---

[1] Asynchronous JavaScript And XML

login.php

Figure 12.3: Preview of the login page

This page includes the login form. When this form is submitted, the login is validated. If the username/password combination matches with a username and password hash from the login table in the database the session is registered and the user is redirected to *index.php*. This page is also used to log out, by opening *login.php?logout*.
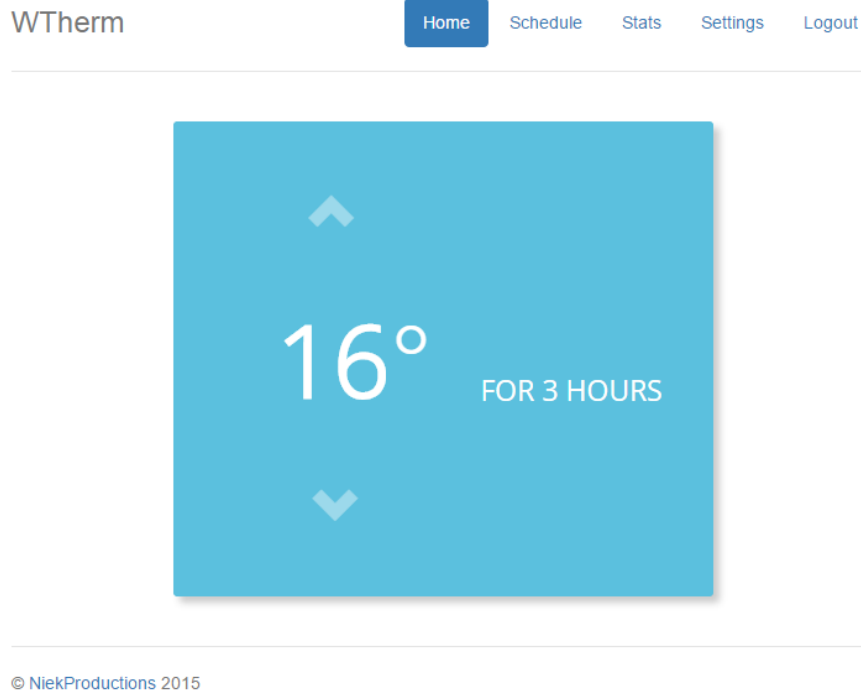
index.php

Figure 12.4: Preview of the main page

This is the main page, where the user can view the target temperature. If the temperature hasn't reached this target yet, the blue square will turn red and the time it will take to heat up will be displayed. Otherwise, the time until the next setpoint according to the schedule will be displayed. This page can also be used to set an override temperature. The override is enabled when the user presses one of the arrows above and below the target temperature. The thermostat will then regulate that temperature until the temperature is pressed. At that point, the thermostat will return to the regular schedule.

control.js is a JavaScript file included in *index.php*. It handles user communication with the database. For example, when the page loads, *control.js* requests the target temperature and override status from *data.php* which in turn communicates with the database. *control.js* also interacts with *data.php* to set a temporary target temperature in the database.

schedule.php
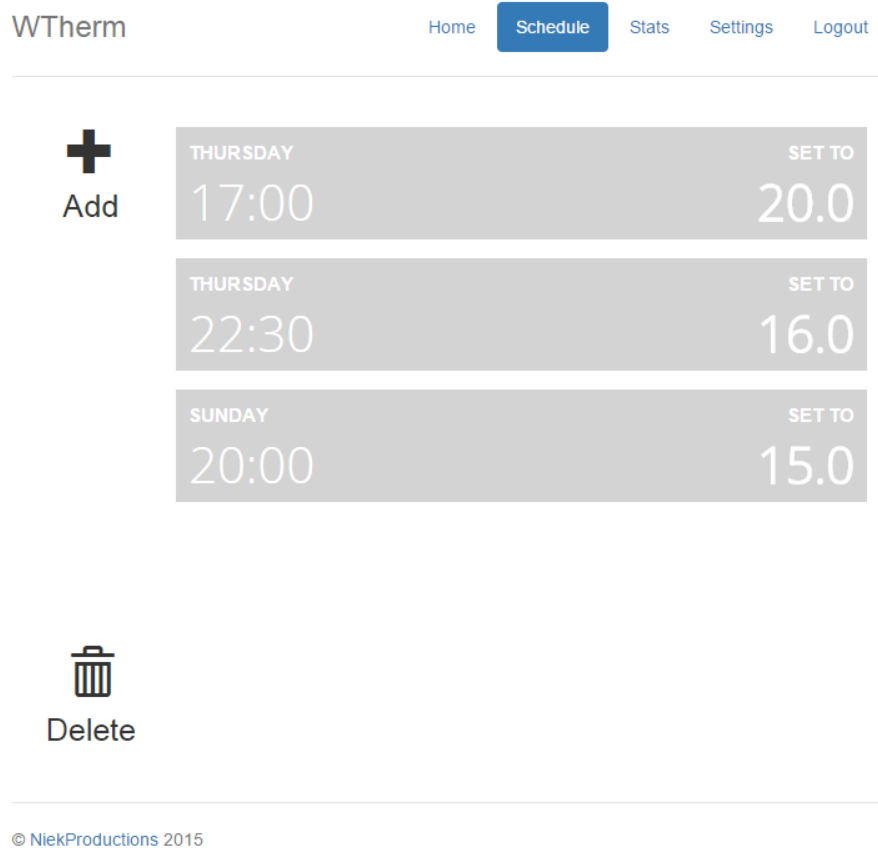
Figure 12.5: Preview of the schedule

This page is used to set the schedule. The schedule consists of targets (or
setpoints) that can be added by clicking the *Add* button on the left of the
page. If a setpoint is selected by clicking on it, it will turn blue and be
editable. A setpoint can be deleted using the *Delete* button.

schedule.js is used in *schedule.php* to edit and save the setpoints. The
Add and *Delete* buttons trigger JavaScript functions that interact with the
setpoint. After every edit, JavaScript passes the array of SetPoints onto
schedule.php to be saved to the database. After every edit, the schedule is
saved.

stats.php

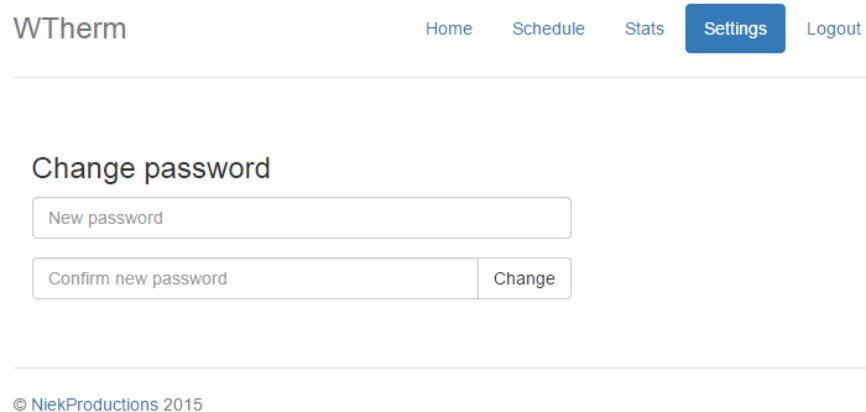Figure 12.6: Preview of the statistics page

This page contains charts of the inside and outside temperature over the period of a day, a week or a year. It uses the JavaScript ChartNew.js library[2] to draw the charts.

drawChart.js is used to specify options for the chart. It uses AJAX to obtain the contents of the log database table through *chartdata.php*.

---

[2]https://github.com/FVANCOP/ChartNew.js/

settings.php

Figure 12.7: Preview of the settings page

The settings page allows a user to change their password.

13 Safety features

In the previous chapter, the function of every source code file was described. This chapter will cover the safety features that were implemented.

13.1 Dealing with reboots

If the thermostat ever loses power, the heater is automatically switched off since the relay described in chapter 10 is normally-open, meaning it will disconnect the heating valve's power if it's not powered.

On the software side, a script is run on reboot that switches the heater off in the case of a power loss.

13.2 Script freezes

If the *thermostat.php* script ever freezes, it will not give rise to a safety concern, since the script is automatically restarted every five minutes.

13.3 Preventing unreliable measurements

The thermostat might be unable to request the temperature from the Home-Wizard. To prevent this causing issues, the heater will be switched off in case no temperature measurements can be obtained for a subsequent period of 10 minutes. This prevents the heater from switching on when the temperature might already have exceeded the target.

PART III

Calculating energy savings

Now that the source code has been explained, the next step is to find out if the WTherm can actually save a significant amount of energy. This is done by comparing the WTherm to a traditional clock thermostat over the course of a single week in the winter.

Imagine a situation in which a household would like the temperature to be 20°C at 9:00 every morning. They set a traditional clock thermostat to start heating at 6:30. Assuming a constant starting temperature of 18°C, the heating time was calculated for an entire week in December 2014. Outside temperatures were used from the weather archive of the Royal Netherlands Meteorological Institute (KNMI)[1]:

---

[1]http://www.knmi.nl/klimatologie/uurgegevens/

Based on the thermal model from page 16, it can be calculated that the guess above is almost correct for the first day. The outside temperature was 5.0°C and it would take 2.6 hours to heat up to 20°C. The same goes for the second and third day, with a heating time close to 2 and a half hours.

The fourth day however, the outside temperature reached significantly higher number and it would have taken just over an hour to heat up to 20°C. This means that the thermostat spent an additional one and a half hours keeping the temperature at 20°C, at the cost of a significant amount of heat. An overview for the entire week is listed in the table below:

Day	Outside temperature (°C)	Human guess (hours)	Actual heating time (hours)	Difference (hours)
1	4.7	2.5	2.6	-0.1
2	4.9	2.5	2.5	0
3	5.0	2.5	2.4	+0.1
4	10.8	2.5	1.2	+1.3
5	11.5	2.5	1.1	+1.4
6	7.8	2.5	1.6	+0.9
7	6.5	2.5	1.9	+0.6

Table 13.2: Heating time overview

This table shows that the traditional thermostat would have spent an additional total of 4.2 hours keeping the temperature at 20°C. Using the outside temperatures and the heat transfer coefficient from 2.1, it can be calculated that during these 4.2 hours, 15.6kWh has been used in order to maintain a constant room temperature. Assuming the WTherm to have switched on the heater at exactly the right time, it would have saved exactly this amount of energy.

According to the KNMI, the winter of 2014-2015 lasts for three months, or about 13 weeks [20]. If all of those weeks would be similar to the one described here, a total amount of 7.3 gigajoules of heat could have been saved

The Dutch energy supplier Nuon employs a maximum rate of € 22,64 per GJ for district heating[21]. This amounts to yearly savings of around € 160.

The same energy supplier charges € 0,64 per m³ of natural gas[22]. The heating value of natural gas from the Netherlands is $32 \times 10^6 \mathrm{Jm}^{-3}$ [23], meaning that the same 7.3 gigajoules would cost € 146.

In this situation, over € 150 can possibly be saved every year, which is more than triple the total cost of the used materials listed in the bill of materials on page 44. This proves the WTherm to be a good investment.

PART IV

Conclusion

The objective of this research was to develop a smart web-based thermostat, WTherm, that saves energy by calculating the time it takes to heat up a living room to a certain temperature. The smart thermostat uses this calculation to precisely follow a weekly schedule that can be programmed through a web-interface. The calculation is based on a thermal model that incorporates several formulas. This thermal model is used to start heating at the right time, reducing the total heating-on time.

Temperatures over the course of a week during the winter were used to compare the WTherm to a traditional room thermostat. The hypothesis was proven that, in comparison with a traditional room thermostat, a significant amount of energy can be saved. In this specific situation, over € 150 can be saved every year. This amount of money is larger than the total cost of the materials used, thus proving that the WTherm is a more cost-efficient replacement for a traditional clock thermostat.

PART V

Discussion

The results might be arguable for a number of reasons.

First of all, the thermal model is based on measurements performed in a room that wasn't heated for a couple of days, meaning that every object in the room also had to be heated up. To verify if the same model could apply to a situation in which the room has been heated the previous day, the traditional room thermostat was set to heat up to 19,5°C at 7:20. This temperature was reached 50 minutes later. The starting temperature was 18,6°C and the average outside temperature was 3,35°C. According to the model, the goal temperature should have been reached after one and a half hours, which is incorrect.

Secondly, the model was created with a relatively high outside temperature. This means that if the difference between inside and outside temperatures is over 19.7°C, $Qloss$ is higher than Qin and the room will not heat up at all according to the model. In reality, this is not the case. Many variables were assumed constant. The heating power for example depends on air humidity and if the sun is shining through the windows the room will also heat up in a smaller amount of time.

The two issues addressed above can be solved by creating different thermal models for various inside and outside temperatures. Ideally, this could be done automatically using the automatic heat meter readout described in the next chapter.

Another possible concern relates to the heater, which is only turned off when the target temperature has been reached. This can cause an overshoot in temperature, since there's a delay between switching the heater on and a temperature rise being measured. A simple solution might be to switch off the heater a couple of tenths of degrees below the target temperature to account for the overshoot, but this constant offset won't be very accurate as the overshoot depends on external factors. This also might result in rapid heater toggling, as the temperature fluctuates around the target. A better

solution would be to use a PID[2] loop. A PID loop would control the heating by calculating Proportional, Integral and Derivative responses and summing those three components to compute the output[24].

14 Possible future research

14.1 Automatic heat meter readout

According to the Landis & Gyr catalog sheet, the heat meter installed in this household has got several ways of retrieving the exchanged heat[25]. One of the ways described is through an IR[1] interface. Landis & Gyr has published a specification sheet[26] detailing the protocol. Obtaining the added heat over the IR interface can be done using an optical read-out head[2]. If this transceiver would be connected to the WTherm, it could autonomously create and improve thermal profiles. This would also simplify integration of the WTherm into other households since the read-out head is IEC 1107/EN 61107 compliant, meaning that it will work with other heat meters that support this standard.

14.2 Reliable heat measurements

The inside and outside temperatures are currently obtained through two wireless sensors. These sensors and need a replacement battery. that can only be read in the accuracy range of a single decimal number. To increase the accuracy, a high-precision wired digital thermometer such as the NXP SE95[3] can be used. This sensor has a resolution of 0,03125°C.

14.3 Compatibility with other households

At the moment, the thermostat only works in a single situation. A future research subject might be possibility for future research would be to

---

[2]Proportional-Integral-Derivative

[1]Infrared

[2]http://www.multical.hu/Optical_Read-out_Head_datasheet.pdf

[3]http://www.nxp.com/documents/data_sheet/SE95.pdf

Bibliography

[1] A.M. Butz. "Temperature-regulator for electric heaters". US 510889 A. Dec. 1983. URL: http://www.google.com/patents/US510889 (visited on 01/21/2015).

[2] Nest Labs. *Inside and Out.* 2014. URL: https://nest.com/thermostat/inside-and-out/ (visited on 01/21/2015).

[3] University of California, Davis. "Heat Capacity". In: *ChemWiki* (Nov. 2014). URL: http://chemwiki.ucdavis.edu/Physical_Chemistry/Thermodynamics/Calorimetry/Heat_Capacity (visited on 11/25/2014).

[4] C.R. Nave. "Heat Transfer". In: *HyperPhysics - Georgia State University* (1998). URL: http://hyperphysics.phy-astr.gsu.edu/hbase/thermo/heatra.html (visited on 01/21/2015).

[5] Alfred Moser. "BUILDINGS AND HEAT TRANSFER". In: *Thermopedia* (Feb. 10, 2011). URL: http://www.thermopedia.com/content/603/ (visited on 12/02/2014).

[6] Unknown. "Radiation Heat Transfer". In: *The Engineering Toolbox* (May 2012). URL: http://www.engineeringtoolbox.com/radiation-heat-transfer-d_431.html (visited on 12/02/2014).

[7] Unknown. "Energy, Enthalpy, and the First Law of Thermodynamics". In: *Bodner Research Web* (2014). URL: http://chemed.chem.purdue.edu/genchem/topicreview/bp/ch21/chemical.php (visited on 12/09/2014).

[8] Roman Frigg. "Models in Physics". In: *Routledge Encyclopaedia of Philosophy* (Nov. 23, 2013). URL: http://www.romanfrigg.org/writings/Models_in_Physics_REP.pdf (visited on 12/27/2014).

[9] Manuel van Rijn. *homewizard-api.* 2012. URL: https://github.com/manuelvanrijn/homewizard-api/blob/master/homewizard-api.rb (visited on 12/27/2014).

[10] Atmel Corporation. *ATmega48PA / ATmega88PA / ATmega168PA / ATmega328P datasheet.* 2009. (Visited on 12/01/2014).

[11] Arduino. *Arduino uno.* 2014. URL: http://arduino.cc/en/Main/arduinoBoardUno (visited on 12/01/2014).

[12] Raspberry Pi Foundation. *RASPBERRY PI DOCUMENTATION.* 2015. URL: http://www.raspberrypi.org/documentation/ (visited on 01/27/2015).

[13] Banana Pi. *Banana Pi*. 2014. URL: http://www.bananapi.org/p/product.html (visited on 01/27/2015).

[14] Apache Software Foundation. *Apache HTTP Server Project*. 2014. URL: http://httpd.apache.org/ (visited on 01/27/2015).

[15] WikiVS. *Apache vs nginx*. 2015. URL: http://www.wikivs.com/wiki/apache_vs_nginx (visited on 01/27/2015).

[16] WikiVS. *Apache vs nginx*. 2015. URL: http://wiki.nginx.org/Main (visited on 01/27/2015).

[17] The Python Software Foundation. *What is Python? Executive Summary*. 2001. URL: https://www.python.org/doc/essays/blurb/ (visited on 12/28/2014).

[18] The PHP Group. *What is PHP?* 2001. URL: http://php.net/manual/en/intro-whatis.php (visited on 12/28/2014).

[19] Oracle Corporation. *What is MySQL?* 2015. URL: http://dev.mysql.com/doc/refman/4.1/en/what-is-mysql.html (visited on 01/03/2015).

[20] Koninklijk Nederlands Meteorologisch Instituut. *Nader Verklaard Seizoensdata*. 2015. URL: http://www.knmi.nl/cms/content/33516/seizoensdata (visited on 01/24/2015).

[21] Nuon. *Stadswarmtetarieven 2015 Kleinverbruik t/m 100 kW*. 2015. URL: http://www.nuon.nl/Images/standaard-warmte-tarieven8-21954.pdf (visited on 01/24/2015).

[22] Nuon. *Variabele prijzen standaard gas*. 2015. URL: http://www.nuon.nl/gas/standaard-gas/prijzen.jsp (visited on 01/25/2015).

[23] NVON. *BINAS*. 2008. Chap. 28A.

[24] National Instruments. *PID Theory Explained*. Mar. 29, 2011. URL: http://www.ni.com/white-paper/3782/en/ (visited on 01/29/2015).

[25] Landis and Gyr. *2WR5 Catalog Sheet*. 2014.

[26] Landis and Gyr. *2WR5 Configuration manual*. 2014.

Appendix

15 License

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/.

16 Software

All of the code for this project is available on GitHub at https://github.com/NiekProductions/WTherm/. Unlike this document, it was released under the MIT License. This is a permissive license, meaning that people can do anything they'd like with the code as long as they provide attribution to the original source. It also does away with any liability.

17 Demo

A demo of the web interface is available at http://niekproductions.com/wtherm/.

18 Git

To store code and keep track of changes Git[1] was used. Git is a piece of software designed for version control. It creates a repository and updates it with information about file versions and the differences between them.

19 LaTeX

A word processor like Microsoft Word could have been used to write this thesis, but it comes with several disadvantages. Mostly, it's hard to manage a document in the format that Word uses. Version control software like Git

---

[1]http://git-scm.com/

is able to handle the file format, but the version differences are not clear since the document isn't stored in plain text. MS Word also doesn't support elements like code blocks with syntax highlighting.

Instead, LaTeX[1] was used to write this thesis. LaTeX is a mark-up language, much like HTML. It was explicitly designed for writing large scientific papers, and saves documents in plain text making them easy to read. LaTeX also makes it easy to insert chunks of code and reformat the documents afterwards.

20 Bill of materials

Part	Price	Amount	Total price
Banana Pi	€ 39,99	1	€ 39,99
Relay, transistor, passive components	€ 1,50	1	€ 1,50
		Total:	€ 41,49

Table 20.1: Bill of materials

21 Installation instructions

Installation instructions can be found in the Readme file on GitHub using the link provided above.

---

[1] http://www.latex-project.org/

22 Log

Date	Location	Activity	Duration
30-04-2014 to 07-07-2014	Home	Preliminary research - WTherm version 1 (source of inspiration), used for snippets of code	30 hours (estimated)
18-09-2014	Home	Finding a subject, outlining idea, brainstorming	1:00
09-11-2014	Hackerspace	Brainstorming, outlining research question	2:00
10-11-2014	Home	Started writing research proposal	1:00
11-11-2014	Home	Continued working on research proposal	1:30
12-11-2014	Home	Continued working on research proposal	0:45
18-11-2014	School	Finished research proposal	2:20
25-11-2014	School	Familiarizing with LaTeX, setting up Git, starting first chapter	2:20
27-11-2014	School	Continued writing first chapter in LaTeX, started bibliography	1:20
29-11-2014	Home	Worked on document formatting, moved the log to the document. Continued chapter one.	3:15
30-11-2014	Home	Continued working on chapter one	1:30
01-12-2014	Home	Started hardware choice chapter	0:30
02-12-2014	School	Researched radiated heat and heat transfer	2:20
05-12-2014	Home	Researched how to interface with heat meter, planned measurements, started converting bibliography to BibLaTeX	2:30

06-12-2014	Home	Performed test measurement for thermal model, continued converting bibliography to BibLaTeX	3:30
07-12-2014	Home	Researching modeling in physics, finished converting bibliography to BibLaTeX	1:30
09-12-2014	School	Started a thermal model of the room	2:20
11-12-2014	School	Processing measurements, planned new measurements	1:30
12-12-2014	Home	Set up camera for measurements	0:30
13-12-2014	Home	Performing measurements	2:00
14-12-2014	Home	Processing measurements	3:00
16-12-2014	School	Continued processing measurements	2:20
17-12-2014	Hackerspace	Tried to compose a formula describing the heating time, documenting	3:00
18-12-2014	School	Drafted thermal model of the living room	1:30
21-12-2014	Hackerspace	Corrected thermal model	1:00
24-12-2014	Home	Finished description of equations	1:45
26-12-2014	Home	Processing measurements for thermal model	6:45
27-12-2014	Home	Finished thermal model, documenting	8:00
28-12-2014	Home	Started part 2	1:00
28-12-2014	Hackerspace	Researched proramming languages and webservers	4:00
01-01-2015	Home	Set up control measurement	1:00
03-01-2015	Home	Started writing code, set up new measurement	7:45
04-01-2015	Hackerspace	Continued writing code, working on discussion, rewriting first chapter	4:00

06-01-2015	School	Documenting MySQL login code and password hashing	2:20
11-01-2015	Home	Writing code. Finished schedule, database class, template parser and login system	10:30
12-01-2015	Home	Setting up Banana Pi, installing LEMP stack with PHPMyAdmin	2:30
13-01-2015	School	Writing code, working on thermCalc class	2:00
14-01-2015	Home	Writing thermostat code	2:30
16-01-2015	Home	Writing code, finished thermostat script, mostly finished user interface	10:15
17-01-2015	Home	Installing thermostat, configuring git, writing chart page	6:30
18-01-2015	Hackerspace	Reviewing chapters, formatting	5:00
19-01-2015	Home	Writing introduction	1:00
21-01-2015	Home	Continued writing introduction, revising first chapter	4:00
24-01-2015	Home	Calculated energy savings	4:15
25-01-2015	Hackerspace	Creating diagrams, rereading and rewriting large parts	6:45
26-01-2015	Home	Proofreading and rewriting parts	2:00
27-01-2015	Home	Finishing I-2	4:45
28-01-2015	Hackerspace	Rewriting, writing code overview	4:15
29-01-2015	Home	Finishing code overview, introduction, conclusion	8:45
		Total (excluding preliminary research)	156 hours

Table 22.2: Log

List of Figures

List of Tables