Home Thermostat for Smart Temperature Control

Dániel Bíró*, Kristóf-Bálint Nagy*, Mátyás Gagyi†, Norbert Nagy-Seres†, Ákos Zsebe†, and Csaba Sulyok*
* Faculty of Mathematics and Computer Science, Babeș-Bolyai University
RO-400084 Cluj-Napoca, Romania
† Codespring
RO-400347 Cluj-Napoca, Romania
daniel.biro@stud.ubbcluj.ro; kristof.balint.nagy@stud.ubbcluj.ro; gagyi.matyas@codespring.ro;
nagy-seres.norbert@codespring.ro; zsebe.akos@codespring.ro; csaba.sulyok@ubbcluj.ro

***Abstract*—Traditional thermostats offer numerous shortcomings because of limited automation options, relying primarily on the time of day. If one does not get home at a preset time, they may be greeted by a cold house and/or waste energy on heating an empty house. The ThermoSmart project aims to make it possible for users to control the temperature of their home remotely. The architecture relies on a server that receives temperature data from a Raspberry Pi 4 thermostat, which is managed through a mobile application. The design of the system adheres to the guiding principle of simplicity. It also provides convenience features, such as a forecast which helps users find an optimal combination of adequate temperature and energy consumption. The thermostat operates with two thermometers, one of which is located inside the house, with the other outside. The data measured by these modules are used to create statistics, which are also intended to provide a basis for the user who desires to optimize the performance of his house.**

***Index Terms*—thermostat, temperature, message queue, WebSocket**

I. Introduction

For the average person, being greeted by a cold home after a cold winter's day could be a problem. Some would opt for a solution of setting the heating on their home thermostat based on time (e.g. to turn on at 6PM) or reached temperature (e.g. turn off when the house reaches $22°C$). This may be appropriate with certainty around the time of getting home; however this is not always the case.

Traditional thermostats, which can store what temperature one wants at what time of day and then maintain it by turning the heating on, have been in use since the 1950s [1]. It has always been a problem that if one leaves their house for an extended period of time and forgets to turn the heating off, the thermostat will maintain the given temperature at the given time(s) and waste energy in the process.

According to the U.S. Energy Information Administration, 85% of American homes have central heating, of which less than half use a programmable thermostat. Chris Mooney of the Washington Post [2] and Peffer et al. [3] speculate that using thermostats is cumbersome and confusing. Mooney also talks about how the thermostats in average homes are old and not of the inhabitants' choosing, but probably inherited, lacking in terms of user interface or functionality.

The software system consists of three components: a mobile application that provides a user interface, a server and a Raspberry Pi 4 [4] which acts as a thermostat, monitors the indoor and outdoor temperature, and switches the heating on and off.

Related work

Several solutions are available in the market; however, each of them presents certain limitations. For example, the thermostat from POER Smart Controls[1] does not have an external thermometer, so it loses valuable information, and it does not provide a weather forecast. The SMARTHER thermostat from Biticino[2] does not provide a good ascending experience on top of the problems listed above. The ThermoSmart project aims to solve these and the aforementioned problems through a targeted mobile application.

II. Architecture

The project aims to save as much money as possible when it comes to heating. This and a convenient user experience is made possible by introducing three architectural elements: a mobile app, a server and Raspberry Pi as a thermostat. The central element is the server, which connects the user's mobile app to the home thermostat.

Users can create a new account using their Google account alone, thus facilitating the registration process, after which they must assign at least one thermostat to themselves. Linking a thermostat to a user (or users) is possible via QR code[3]. The home unit is equipped with a QR code that is used as a unique identifier in the system. Users and thermostats are in a many-to-many relationship, i.e. one user can manage multiple thermostats and one thermostat can be managed by multiple users.

The left side of Figure 1 shows the two target devices, phones running Android and iOS, communicating with the server through RESTful [5] HTTP and WebSocket [6] based channels. The presence of two different channels is due to the goal of presenting real-time data to users. The REST connection is suitable for sending commands from the mobile to the server, but not in the opposite direction, because the phone does not have a public IP address. Using the

[1] https://www.poersmart.ro/

[2] https://www.bticino.com/products-catalogue/smarther-the-connected-thermostat/

[3] Quick Response code–two dimensional machine-readable point code

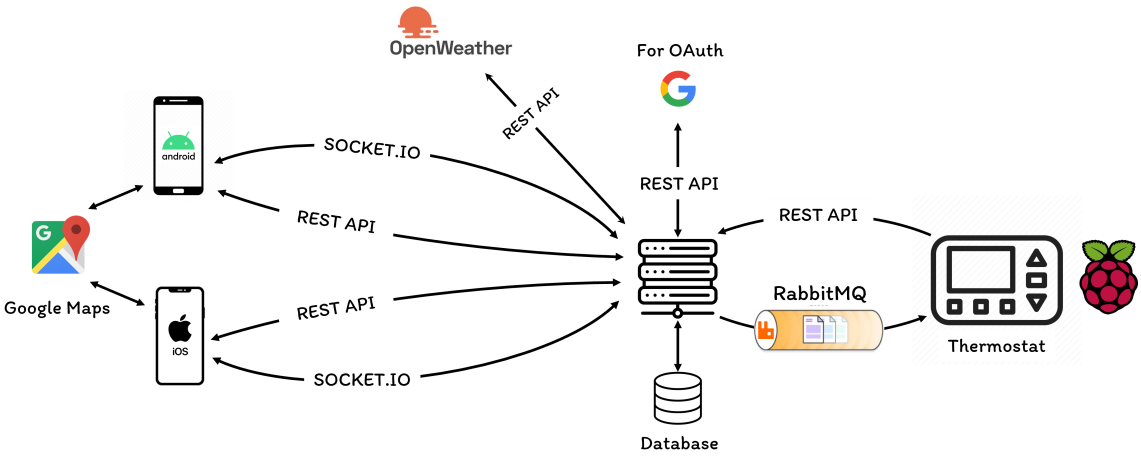

Fig. 1. System architecture

HTTP connection the mobile app can only be notified of any temperature change if it explicitly requests it, which results in continuous and largely redundant HTTP requests. To solve this problem, WebSockets are used, which implement bidirectional and delay-free communication (Socket.IO [7] library). The technology is subscription based. Mobiles subscribe to thermostats, meaning that they are interested in the data from a given thermostat. A custom data structure is used for maintaining the subscriptions of mobile apps to thermostats (linked lists embedded in a hash table).

The mobile application is also equipped with a weather forecast to help the user decide when to turn on the heating. For example, if it is going to be 15 degrees during the night, hence the house will cool down, but by morning it will be 22 degrees, it is not worth heating.

In the middle is the server, which manages a PostgreSQL database. The server is connected to the OpenWeatherMap API [4], for the reasons mentioned above, and to Google's OAuth [8], which is used to make Google login possible.

On the right is the Raspberry Pi 4, which communicates with the server using REST and AMQP [9] protocols. There are two channels, for a similar reason as between the server and the mobile. The difference in terms of functionality is that the thermostat receives the target temperature. The second connection is again based on a protocol capable of back and forth communication (AMQP), but unlike the mobile, the Raspberry Pi runs a full Linux operating system [10], which allows the use of more complex technologies. The sending party (server) sends the message to the *message broker* and the receiving party (thermostat) subscribes to the topic specified by the sending party, thus receiving the message. The communication between the two elements is solved via the *CloudAMQP* [5] provider, using the *RabbitMQ* [9] protocol.

[4]https://openweathermap.org/api
[5]https://www.cloudamqp.com/

III. The Application Server

The core element of the project is the server, which establishes and maintains communication between the mobile application and the thermostat. It is capable of processing and handling incoming requests, and then sending a response back to the requesting party.

A. Technologies

The server is written in TypeScript, using the Express.js framework and the Node.js runtime environment [11].

Node.js is a JavaScript runtime environment based on Google's V8 engine. It is capable of asynchronous and synchronous programming, even though it runs on one thread. Being famous for creating web applications, web servers and command-line utilities, the technology is used worldwide. Using its built-in package manager, *npm* third-party packages can be added to the project easily.

Other third-party packages are also used in this project. These include the *amqplib* library that imports the *AMQP* protocol, the *Winston* and *Morgan* loggers, the *ESLint* static code analyzer, the *Passport.js* authentication middleware, the *Axios* HTTP client and the *nodemon* code analyzer.

The persistent data is stored in a PostgreSQL database, which is a popular choice for large scale web applications. TypeORM[6] is used in order to build the connection between the server and the database and to perform the necessary operations. An ORM (Object Relational Mapping) [12] is a technology that creates a connection between object-oriented programming languages and relational databases by generating tables from model classes and by ensuring optimized functions.

B. Architecture

The structure of the server consists of three main parts, corresponding to the multi-layer architecture pattern [13]. These are the following: the *API* (Application Programming

[6]https://typeorm.io/

Interface) component, which handles the requests coming from the mobile application, the *domain* component, which manages the communication between the API layer and the data access layer, and the *persistence* component, which is used to access the database using the *CRUD* operations.

C. Application Programming Interface

This section is also divided into three different parts. The *dto* package contains the DTOs (Data Transfer Objects); these are used to map data incoming or outgoing with a request into classes which can be later processed by the server. The *middleware* package contains middlewares, functions that run before accessing given endpoints. They are used to check for user authentication. The *route* package contains the routers, which divide each request by the path given inside them. This package contains the functions that handle the access to the predefined endpoints. Every function can handle one type of *HTTP* request: *GET*, *POST*, *PUT*, *DELETE*. Apart from the types mentioned above, a path must be specified, and optionally query parameters can be added.

The project supports many main endpoints. The `/temperature` endpoint is used to save the temperature retrieved from the thermostat in the cache or in the database. The `/device` endpoint is used to perform device-related operations, such as registering a new thermostat, updating its name and assigning a device to a specific user.

D. Domain

The domain section contains three main sub-packages. The *mapper* functions that convert the entities of the data access layer into data models that are returned to the API. These data model classes can be found in the *model* package. The *service* package contains the functions that sustain the connection and the communication between the *API* and the *persistence* packages.

E. Persistence

The persistence layer is made up of three elements. The first package, *db* is responsible for creating the connection with the database and initializing the cache. These functions are called every startup or restart. The second package, *entity* consists of the entities used by the ORM. The last package, *repository* includes functions that communicate with the database.

F. Data model

The data model used in the database is as follows:

- The *device* table contains all the information about the registered thermostats: their identifier, their name as well as the preferred temperature according to the user's setting.
- The *user* table consists of data about the user, saving information from their Google profile. It retrieves the user's full name, email and the URL of their profile picture.
- The *program* table is made up of the user-set temperature preference, broken down into intervals for each day of the week: the starting and the ending date, the set temperature and the day the program starts.
- The *user_and_device* table indicates which thermostat is paired to which user, doing so by saving each entity's identifier in the table.

```
const strategyOptions:StrategyOptionsWithRequest={
  clientID:
    process.env.GOOGLE_CLIENT_ID as string,
  clientSecret:
    process.env.GOOGLE_CLIENT_SECRET as string,
  callbackURL:
    `${process.env.BASE_URL}/auth/google/callback`,
  passReqToCallback: true,
};
```

Fig. 2. Configuration used to set up Google as the OAuth2.0 provider

G. Login

Since the login can only be done with a Google profile, *Passport.js* is used to facilitate the process of logging in. Google uses the OAuth2.0 protocol, so the `passport-google-oauth20` strategy is chosen to log in. To use the middleware, it is necessary to register the application in the Google Developers Console. A Client ID and a Client Secret are then generated and these must be specified when setting up the strategy; Figure 2 shows the setup options.

Before use, an Express.js session needs to be created, Passport needs to be initialized and the Passport-specific *session* function needs to be called. The latter logs in the Express session using the Passport session strategy.

Due to the above, every time a login attempt is made, a strategy called *GoogleStrategy* is ran, where three cases can happen: successful login and the user already exists, successful login but no such user exists in ThermoSmart yet and is created in the database, or unsuccessful login. If the login is successful, the done function is called, which tells the program to jump to the next endpoint.

Two endpoints are required to log in, one to start the login and one to redirect the user back to if the login is successful. The user is then redirected to the mobile app again.

IV. Mobile Application

The mobile app is the user interface from which one can control the temperature of their home, get statistics and set the position of their thermostat on the map, which is essential for accurate weather forecasting, another feature of the app. You can also set traditional programs, such as 21 degrees every morning at 7 am. One can connect to a specific thermostat via the mobile app.

Each function is specific for one thermostat. The system is generalized to handle multiple thermostats. It is possible to choose between the registered thermostats. When one switches to another thermostat, the old data is replaced by the data associated with the new thermostat.

The program is written in TypeScript using the React Native [14] library. React's Context API is used for data

management and distribution. An external library (React Native Elements[7]) is used for design.

A. Login

On startup, the application checks whether there is any user data already saved on the phone. If there are, it retrieves the last used thermostat and displays the main page. If there are none, it prompts the user to log in. The default login strategy is via Google.

B. Communication

Two different technologies are used for communication: simple HTTP calls and WebSockets (for further details see Section II).

To simplify HTTP calls, the *Axios* library is used, with which it is possible to globally set the server URL. The DTO design pattern is used at all requests.

Secondly, the mobile app must show the home temperature in real time. This could be done with continuous HTTP requests, but there is a better way. WebSockets are capable of real-time back and forth communication based on observer pattern, so this technology is used in this project.

The Socket.IO library is used to create a channel upon application start. The library ensures that any interruption is automatically remedied at the first opportunity and no data is lost in the process. This channel is used to receive instantaneous data from the thermostat (e.g. outdoor, indoor temperature, success of heating on/off).

C. Chart

One of the aims of the ThermoSmart project is to help users to heat their homes optimally. One of the ways this is done is by providing statistics on the relationship between the temperature inside the house, the temperature outside the house and the heating itself.

The server provides a large amount of data, as thermal information is saved at small intervals (12 data/hour). The graph can only plot a finite number of points to look aesthetically pleasing, so the data is averaged out.

D. Map and weather forecast

The Google Maps service is integrated in order to determine the position of the thermostat. This is used for obtaining precise weather forecast. This information is meant to help users make the best decisions when it comes to heating their home.

V. Thermostat

The thermostat is simulated using a Raspberry Pi 4 minicomputer. We chose this device because it supports the sensors we use and is therefore an easier and cheaper solution than a dedicated thermostat. We connected two DHT22 [15] temperature and humidity sensors, one to monitor the temperature outside the dwelling and the other to monitor the temperature inside the dwelling. A relay is also connected

[7]https://reactnativeelements.com/

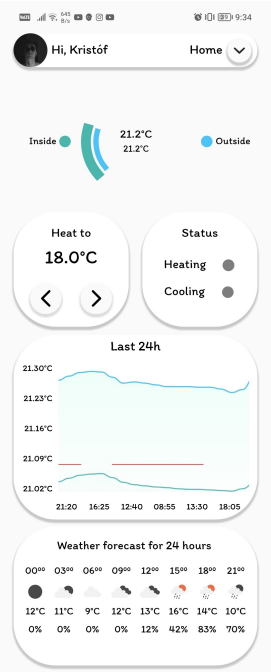

Fig. 3. Main screen

to the device which allows us to control the temperature. Heaters can be turned on by bringing two cables (which come out of them) into contact, which is the purpose of a relay.

Raspberry Pi is a computer the size of the palm of one's hand. It has its own processor, RAM and video card. However, it has no back-up storage, which can be remedied by inserting a micro SD card. One can run a full-fledged operating system on it. The vendor recommends the proprietary Linux-based Raspberry Pi OS, which is designed for this purpose.

The Raspberry Pi differs from classic computers in that it has forty programmable pins. Throughout the project two temperature and humidity sensors and a relay are connected to the device.

The code running on the Raspberry Pi is written in TypeScript and follows object-oriented principles like the rest of the project. It communicates with the server via the *REST API*, while the server sends data back to it via the RabbitMQ cluster of the *CloudAMQP* provider. Its core is based on a loop that runs over and over again at predefined intervals. It performs two tasks. The first is to get the temperature and humidity readings from the sensor and then send their average to the server. The second is to turn the heating on or off depending on the setting.

VI. Usage

In order for the thermometers to provide real information, both should be placed in a place protected from the sun, one inside the house and the second outside.

Sign in via Google. The user needs a Google Account to use the mobile app.

After logging in, one is presented with a QR code reader. The thermostat is provided with a QR code, which can be scanned here to connect.

In the top bar (see Figure 3), on the left side, the Google profile picture is shown. Pressing on the avatar brings up the profile screen, which provides more information about the user and where one can switch to dark mode for night use.

On the right is the name of the thermostat currently being managed. The arrow next to it opens a list of all thermostats connected to the user. Here, if one clicks on one of them, the application will from now on control this thermostat and display information from it.

In the middle is the "*monitor*", which shows live information about the temperature. The outer ring represents the inside (upper number in larger font) and the inner ring represents the outside (lower number in smaller font) temperature.

Below the "*monitor*" on the left is the most important "*card*" to control the temperature. By long-tapping the buttons, one can quickly make a big change, pressing them only once will move the target temperature up or down by half a degree.

To the right of it is the status card. It indicates whether the heating and cooling equipment is on. If they are, a coloured circle appears in place of the grey ones.

Below it is the graph, which provides statistics for the past day. The red line indicates when the heating is turned on.

Below the status card is the weather forecast for the next 24 hours which comes live from the internet and is linked to the thermostat location.

It is possible to remove a thermostat from a user via the mobile app. Also, one can change the name of the thermostat which is global among users. Lastly, it is possible to set the position of the thermostat using the integrated Google Maps inside the app for precise weather forecast.

VII. Conclusion and Future Development

There are several strands to the project that are planned to be developed further and introduced in the future.

The most important option is the introduction of an artificial intelligence that would design an optimal heating program based on past (database) and future (weather forecast) heat information. In this way, all the user would basically have to do would be to tell the AI how many degrees they would like and when, and the AI would do the rest. This would create programs that humans would never think of.

Another option is to create an administration interface, which would be a web page where thermostats could be entered into the system, users could be deleted, and the system could even be restarted in case of a failure.

Ultimately, increasing security is an important improvement, as at the moment anyone with a QR code for a thermostat can control it. This can be done by introducing a password.

References

[1] Endesa, "Illustrated History of the Thermostat," 2017. [Online]. Available: https://www.endesa.com/en/blogs/endesa-s-blog/air-conditioning/illustrated-history-of-the-thermostat

[2] C. Mooney, "Americans could save a fortune this winter — if they only understood their thermostats," 2014. [Online]. Available: https://wapo.st/3WMXjEb

[3] T. Peffer, D. Perry, M. Pritoni, C. Aragon, and A. Meier, "Facilitating energy savings with programmable thermostats: evaluation and guidelines for the thermostat user interface," *Ergonomics*, vol. 56, no. 3, pp. 463–479, Sep. 2012.

[4] D. Denton, *Raspberry Pi 4: A Comprehensive Guide to Raspberry Pi 4 Setup, Learning Programming and Developing Innovative Projects*. Independently Published, 2020.

[5] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, 2008.

[6] A. Lombardi, *WebSocket: Lightweight Client-Server Communications*. O'Reilly Media, 2015.

[7] R. Rai, *Socket. IO Real-Time Web Application Development*, ser. Community experience distilled. Packt Publishing, 2013.

[8] R. Boyd, *Getting Started with OAuth 2.0*. O'Reilly Media, Inc., 2012.

[9] A. Videla and J. Williams, *RabbitMQ in Action: Distributed Messaging for Everyone*. Manning, 2012.

[10] W. Harrington, *Learning Raspbian*, ser. Community experience distilled. Packt Publishing, 2015.

[11] D. Choi, *Full-Stack React, TypeScript, and Node: Build cloud-ready web applications using React 17 with Hooks and GraphQL*. Packt Publishing, 2020.

[12] K. Roebuck, *Object-Relational Mapping: High-impact Strategies - What You Need to Know*. Emereo Pty Limited, 2011.

[13] M. Fowler, *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*, ser. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012.

[14] A. Paul and A. Nalwaya, *React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications*. Apress, 2019.

[15] "DHT22 technical specification." [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf