University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization:	Spring semester, 20..... Open / Restricted access
Writer:	…… (Writer's signature)
Faculty supervisor: External supervisor(s):	
Thesis title:	
Credits (ECTS):	
Key words:	Pages: …… + enclosure: …… Stavanger, …….. Date/year

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Developing the SHPIA 2.0 Architecture for the Raspberry Pi

Master's Thesis in Robotics and Signal Processing
by

Lars-Erik Nes Panengstuen

Internal Supervisors

Florenc Demrozi

June 15, 2023

Abstract

Sensors and electronic devices has invaded out daily lives. These devices provides an opportunity to collect invaluable data for research purposes. This research has the ability to give us an insight into lifestyle illnesses and allow us to more accurately target decease through preventative measures. The SHPIA architecture uses inexpensive Bluetooth devices along with a raspberry pi micro-computer to provide researches with a versatile platform for gathering this invaluable information.

Acknowledgements

I would like to thank my supervisors for their fantastic enthusiasm and help with writing this thesis.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, there has been a growing interest in smart-home devices like smartwatches, smartphones and digital assistants to aid in daily task and monitor peoples habits. These devices have the ability to gather invaluable data for lifestyle-research purposes. Lifestyle data refers to information about an individual's habits and behaviors, including their physical activity, sleep quality, and dietary habits. This type of research have a couple of factors that makes it extremely hard to get accurate data on. Lifestyle is a complex and multifaceted concept that encompasses a broad range of behaviors, habits, and choices that can vary widely between individuals, cultures, and socioeconomic groups. Because of the nature of lifestyle diseases, they can take years or even decades to develop, meaning the studies have to monitor subjects for a very long time to get accurate data. Additionally, there is often no way of eliminating variables. People are going to live their life, and getting someone to change one aspect of their life for extended period of time can be extremely costly or sometimes impossible. This means that researches rely on more data to be able to isolate the variable since there often is no way to control for it. These factors has meant that lifestyle research has been extremely costly and often result in inconclusive conclusions. This is the primary reason we hear that egg, wine, exercise and coffee is good for you one week and bad for you the next.[1]

Traditional methods of collecting this data each have their own issues. Cameras pose a huge privacy issue, as well as cost associated with installation, storing and analyzing video.

Self-reporting methods such as surveys and diaries are prone to biases as well as requiring some work from the subjects.

Smart homes have the ability to offer a more objective and accurate way to collect this data by using sensors and devices to automatically capture information about daily routines and behaviors.

However in order to gather enough data for research, the smart-home solution needs to be adopted in many homes or care facilities. The only way this is going to happen is if the barriers to entry like cost, tech-literacy requirement and installation work, are kept low.

Research into lifestyles have the potential to provide a better understanding of human behavior and aid in making environments and policies to promote healthy lifestyles through preventative measures.

1.2 Objectives

The goal of this paper is to develop a complete system to collect data from sensors in an environment and makes it available in a simple, secure and efficient way.

in order to achieve this the system needs:

- Create a program that allows a microcomputer like a raspberry pi to connect to Bluetooth sensor devices, log data from the devices and upload it to a central server.
- A central server with a web-based API is needed to receive the data from all base-stations, store them in an organized fashion and make it available to researchers. This API needs a couple of features in order to complete this objective:
  - An ability to create and login to a user in order to control the data being collected.
  - An intuitive way for the user to classify and keep track of the different logging devices.
  - An ability to automatically receive and store the data from the research environment.
  - a way of classifying the data from the raspberry pi as belonging to a user.
  - a way of sharing the data between accounts.
  - a way of extracting that data to be used for research.
- It is important that the configuration of devices is both intuitive and simple. A website running on the local network hosted by the pi would allow the user to configure logging devices through any device with a browser.

- Adding the ability to measure the signal strength of the Bluetooth connection could prove useful to estimate the location of the device relative to the base station. This could provide an extra tool when conducting research.
- Add compatibility with all devices in the SHPIA architecture.

1.3 Approach and Contributions

This paper expands the SHPIA framework described in [2] with the ability to run on a microcomputer like raspberry pi rather than a Android device. The previous SHPIA solution was designed for the android devices and could therefor not be used with microcomputers. A new server as well as a website was made with no prior code. The microcomputer had some python code which allowed the microcomputer to connect to the thingy devices [3]. This code was expanded to work with the server and website [4].

1.4 Outline

Chapter 2 covers previous work and explores the feasibility of smart home for research.

Chapter 3 start by giving a general overview over the whole system and how the different parts of the system relates to each other. The subsequent sub chapters in chapter 3 goes through each part of the system in more detail.

Chapter 4 covers the setup used to evaluate the system and the results of these evaluations.

Chapter 5 Concludes the paper by summing up the successes of the paper as well as the potential improvements for future work.

Chapter 2

Related Work

The Smart home literature is way to vast to cover completely in this section. The smart home space can be divided into two groups, Commercial and research focused. Since SHPIA is a research focused smart home solutions it makes sense to look at other research focused solutions. The first thing to consider in smart home research is weather the smart home sensors used is even sufficient to accurately monitor activities of daily life.

The CASAS platform[5]. uses a base-station that communicates to the senors wirelessly through the zigbee protocol to collect data from an environment. The devices collected data on light, motion, temperature and the positions of doors in the environment. From this data, the authors were able to recognize ten different Activities of daily life with a 60% accuracy.

Another paper was published in 2012 [6], where the authors used different human activities recognition models to recognize twelve activities with a 50 to 60% accuracy. They did this with a combination of motion and door sensors placed in the environment.

In [7] a sensor network was installed in an eldercare facility to monitor cognitive health changes in patients. The system used motion, video and bed sensors to gather data. The authors lay out different ways this data can be used to monitor daily activities of elderly like sleep quality and forgetfulness to catch early signs of cognitive decline.

In [8] the authors present smart home system running on a single personal smartphone. A combination of the motions sensors and the WiFi-data is used to monitor the activities of elderly patients.

[9] goes through 55 papers on non intrusive smart home solutions. The paper is a comprehensive deep dive into the different sensors and communication protocols used to monitor different activities. The paper makes a couple of assertions.

- The authors concludes that a comprehensive long term study, convincingly proving the efficacy of smart home solutions is still lacking.
- While behavioral data can be easily acquired, unobtrusive ways of collecting reliable psychological data is still lacking.
- Most papers does not mention any considerations for the privacy concern of the subjects. This could be a contributing factor for the lack of a long term comprehensive studies.

From these studies we can conclude that motion, temperature and light sensors can give a valuable insight into activities of daily life of research subjects.

The smart home solution that this paper will be working with is SHPIA(Smart Home Platform for Intelligent Applications)[2]. This smart home solution uses an android device as the base-station and uses Bluetooth low energy to communicate with sensor devices. SHPIA currently support 4 sensor devices: Estimote Locator, Estimote Sticker, Estimote global tag and the Nordic Thingy 52.

The Nordic thingy 52 is a compact multi-sensor device designed to collect a range of data of various types. The sensors in the Thingy 52 include: accelerometer, Magnetometer, gyroscope, thermometer, barometer, light sensor, hygrometer and carbon dioxide meter. The Thingy 52 uses a two-side BLE communication. This puts a limit on how many devices can be used with one base-station.

Chapter 3

Approach

When designing a an online system. The first thing to consider is what frameworks to use. There are usually thousands of frameworks to choose from and they all are capable of the same results. The difference often lies in things like coding language, speed, supported features.

The .Net framework was chosen for the back end server. The reasons for choosing .Net were.

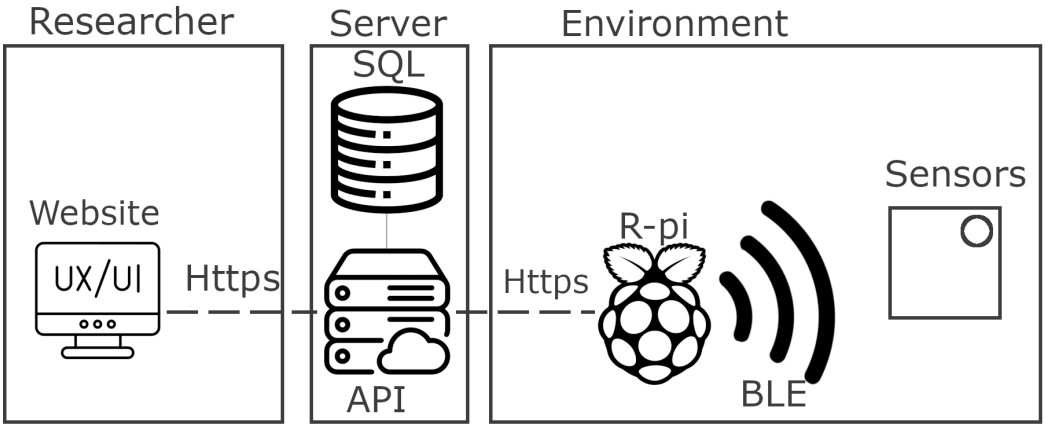
- .Net is made and supported by Microsoft since 2002. This gives it great compatibility with other Microsoft services like Visual Studio and Microsoft SQL
- .Net core is one of the most popular back-end frameworks. Which greatly increases the ability to find resources online
- .Net is uses the programming language C#which makes it one of the fastest back-end frameworks.

These factors mean that the .Net framework will give more streamlined development as well as good prospects for future development.

For the database management system the Microsoft SQL system was used due to its integration into visual studio.

The primary deciding factor when choosing a front-end frame work was popularity since this will determine the amount of online resources available in development. The most popular front-end frameworks are Angular and React. Although react is more popular, the Angular framework was chosen due to its modularity.

3.1 Overview



The whole system can be divided into three main parts.

- Raspberry pi: Where the data is being collected.
- The API: The central server where the data is being processed and stored.
- The Front-end application/ UI: The way users interact with the data.

The Raspberry pi is in the environment where devices are collecting the data and communicates with them via Bluetooth. The raspberry pi is simultaneously running a website on the local network, allowing it to be configured via a web browser on any device connected to the same local area network. The raspberry pi stores the data locally for a set amount of time before sending it to the central API to be stored on an SQL-database. The front-end allows the user to make or delete environments as well as extract the data from the servers.

3.2 API

The API acts as a intermediary between the database and any user or device.

Having a web-API handle the database comes with a number of advantages.

- The server can receive data from any device with an internet connection, meaning that anyone can make programs that interfaces with this ecosystem.
- The API handles all security and account management. This make developing a user interface much easier.

The API can be divided into Two main parts, the user management and data handling.

3.2.1 User Creation

The simplest way of user creation, would be to ask the user to fill in a username and a password and create the user from that. However would allow anyone to create a user, or a thousand, or a million users for that matter. This could quickly bog down the system, making the user experience worse and creating unnecessary strain on the server. In order to deal with this we need to verify that the user is authentic. To do this we need something from the user that everyone has access to, but it would be hard for an attacker to get millions of. The most common industry standard, is to use a valid Email address. To do this the username is exchanged with an email-address. When a user is created the user is not immediately activated, but rather an activation token is created on the server. The server then sends an activation link to the specified Email-address containing the activation token. When the user follows the link sent to their Email, the server receives the activation token from the user, letting the server know that the email is valid and authenticating the user. A demo of this can be seen in [C] at around 00:30. A similar system is used to reset password. When a request to reset a password is received from the server a password reset token is created and a link containing that token is sent to the users Email. When the user follows the link sent to their email, it lets the server know that the person contacting the server has access to the user email, and allows them to change the password for that user. The password-reset feature is currently working on the back-end. However there is currently no way of interfacing with this endpoint through the front-end application.

3.2.2 User Authentication

When a person creates a user on the server, they provide an Email as well as a password. In order to later authenticate the user, we have to store the Email and password on the server. However storing a user's password directly is not a good idea. Anyone with access to the database would be able to see the password of any user.

Passwords getting leaked is a huge concern when running user databases. A study from google in 2019 [10] showed that only 35% of people use different passwords on all sites, and as much as 15% uses the same password on all sites. A leak of passwords can not only lead to the leak of information on the site of the leak, but also on other sites that might contain even more sensitive information.

To deal with this, each password is encrypted using the SHA512 algorithm. SHA512 is a one-way encryption algorithm. This means that the hash can be calculated from any password very easily, but calculating the password from a hash is virtually impossible. SHA512 encrypts the password and stores the hash in the database. When the user wants to login, the server can take their password, run it through the SHA512 algorithm and see if the resulting hash matches the stored hash.

One problem with this solution is that if the whole database was leaked and an attacker had access to all hashes, anyone with the same password would also have the same hash. This means that if the attacker has a large list of known password hashes, they can cross-reference the hashes of those two lists and get the passwords of users using the most common passwords. In order to prevent this, salt is added to each password. Salt is a random string that is added to the password during encryption to make sure that the same passwords will not lead to the same hash. All of these measures are taken to protect the user's password.

Every time the users sends a request to the server the server needs to verify that the user actuality has the authority to make the request. To do this the user could login for each action, but this leads to a very bad user experience. That's why once the user has logged in, the server needs another way of authenticating the user for each subsequent action. To do this a JWT(JSON Web Token) is used. A JWT is an encrypted key that contains information about the user in a JSON format. This can contain any information like user id, access passes, admin status, etc. When a user logs in, the server creates a JWT using a secret encryption key stored on the server, and sends it back to the user. The program the user is using(typically a website) then stores this token containing the credentials of the user. When the user wants to post subsequent requests that require validation to the server, the JWT is sent along the request to validate it. The server can then simply look at the JWT and check if the credentials on the JWT has the authority to perform the action. The JWT contains a signature at the bottom which is tied to the information in the token, which can only be made using the secret encryption key. This ensures that a potential attacker cannot simply swap out credentials in the JWT to gain access to other accounts.

3.2.3 Data Organization

In order for the data to be stored in a contextualized and organized manner without the need for the user to invent a complex naming scheme, the data-points are stored in containers called environments on the server. Environments can be created and deleted by the user using the web interface. The environment can also have several sub-environments and sub-sub-environments etc. Here is an example of how this could be implemented in a house.

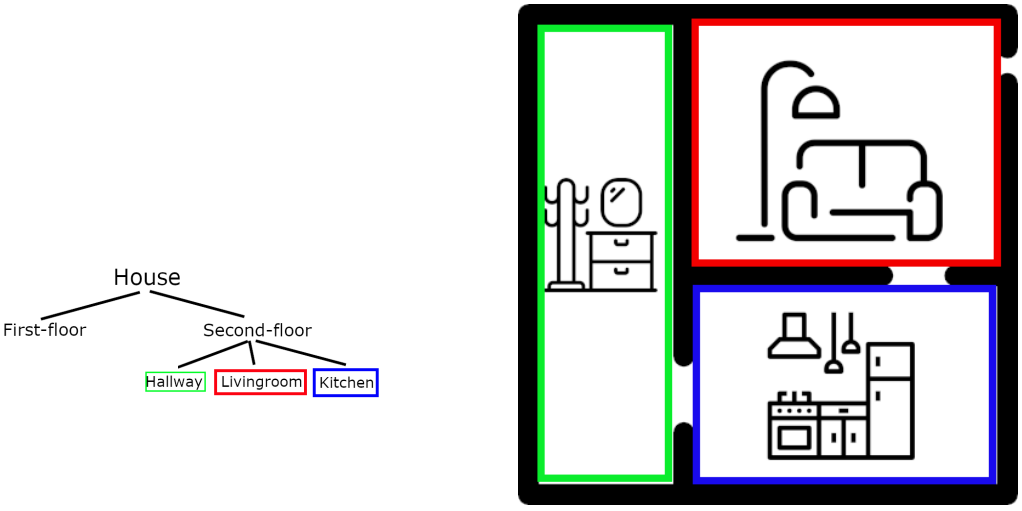


Figure 3.1: Create primary environment

Here we have a diagram showing the physical layout of the second floor in a house, as well as a diagram showing the structure of environments in the database. The user has created a primary/ top-level environment for the house itself which has two sub environments, first-floor and second floor. The second floor is then further divided into living-room, kitchen and hallway. If the user wanted to log the amount of air particles coming from the stove, they could add a device called "stove" to the kitchen environment. Additionally if they wanted to measure the amount of air particles coming from their automatic vacuum cleaner, they could attach a device to the vacuum, and make it log to the second floor environment, since it will be moving between rooms but not between floors. If the house had a second vacuum cleaner in the first floor, a device could be added to the first floor environment, also called vacuum cleaner. This lets the user name devices based on what they are and let the environments handle the differentiation of devices. This helps eliminate naming-schemes like "vacuum_cleaner_first_floor" and gives an intuitive way of organizing devices.

3.2.4 Receiving Data

When the API is receiving data there is a few things that has to go along with the raw data itself. The server knows the type of data based on what endpoint is being hit. For example, when the server gets a post request on the "/raw_data" endpoint it knows that the data is raw data from the motion sensor and the device making the post request has to make sure the data is in the correct format to be written into the database. But the server also needs to know what environment to file the data under, and that the user making the request has the authority to write to that environment. To do this, the server uses a similar approach to the login method.

In the following diagram the figure to the left represents the central API server, the screen in the middle represents the browser the user is using, and the raspberry on the right is representing the raspberry pi, logging the data.

When the user first logs into the website, the website gets the login token for the user. 3.2

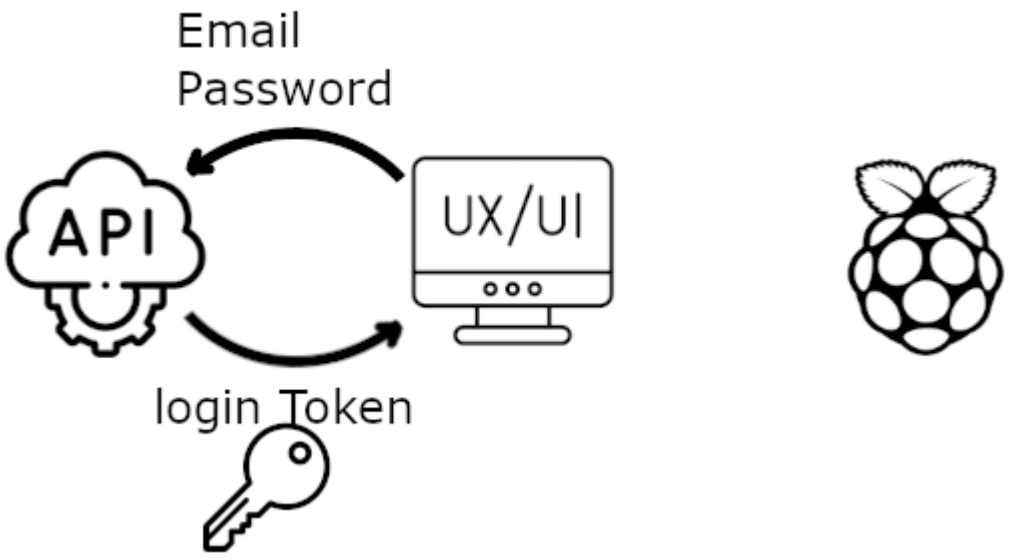


Figure 3.2: User Login

When the user wants to add or configure a device to a specified environment, they press the "configure device" option on the environment in the browser. This will open a new window hosted by the pi, where they can configure the device. However before the website opens this window, it sends a request for a JWT for the specified environment to the server. The server check that the specified environment belongs to the user making the request, and returns a JWT to the website.3.3

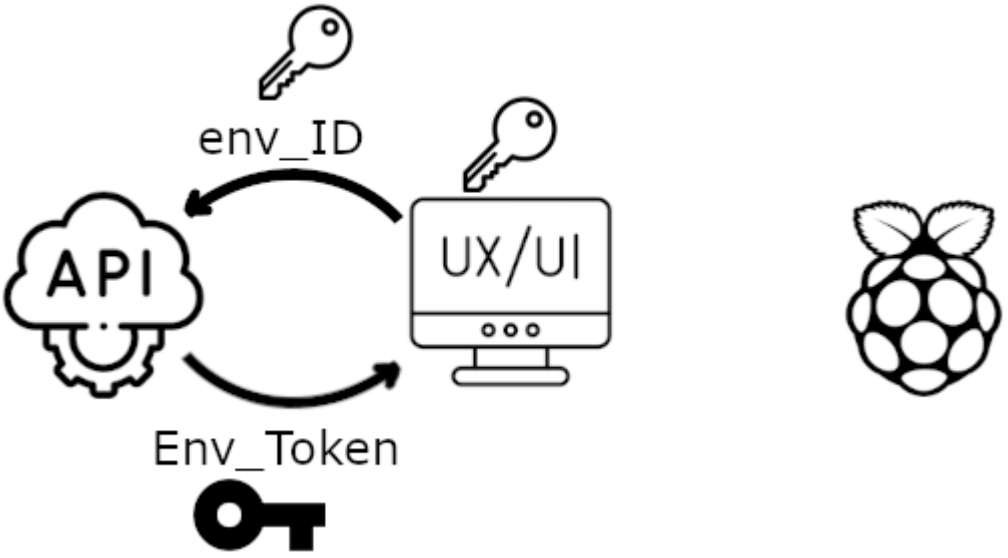


Figure 3.3: Getting Environment Token

When the website opens the window on the pi it simultaneously passes the environment token. The Pi then stores this token along with any device configured in the window.3.4

Figure 3.4: Passing the environment token

When the device has logged data onto the pi and the pi is ready to upload the data, it passes the environment token it got when the device was configured, along the data in the header.3.5

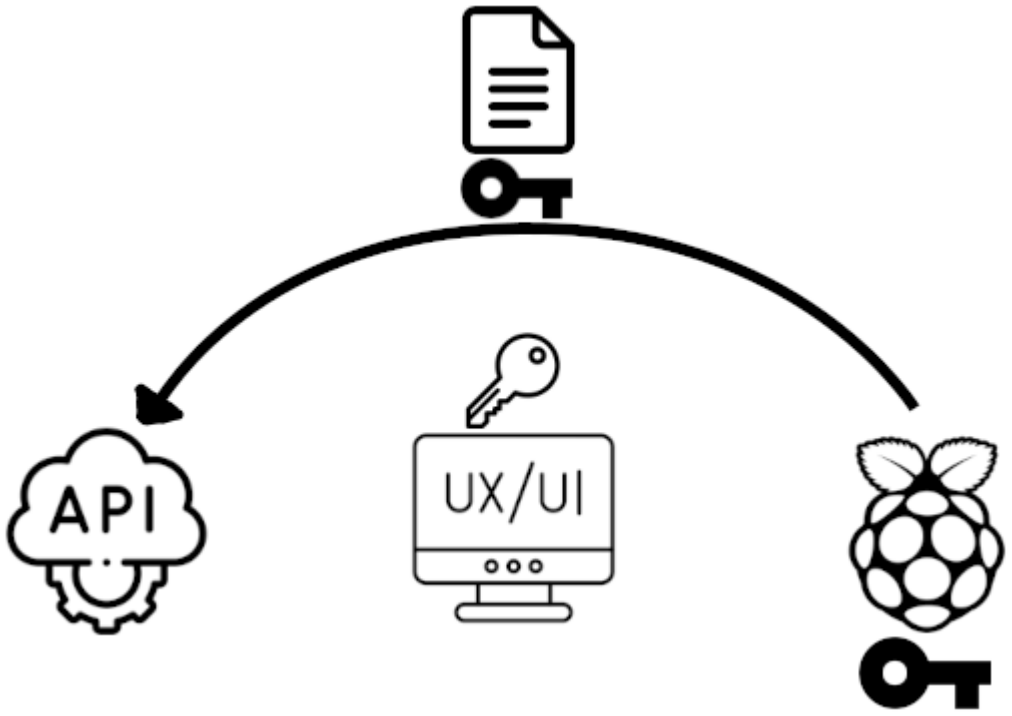


Figure 3.5: Uploading data to the server

The server can then log the data under the environment specified in the environment token and be sure that the data came from a device configured by an authorized user.

3.2.5 Database

In the SQL database the data is divided into tables. SQL tables can relate to each other in one of four ways. These are, "one to one", "one to many", "many to one", and "many to many". In this project "only one to many" relationships are used for data-tables. When two tables have a one to many relationship it means that records from table B can reference or belong to exactly one record in table A, but records in table A can reference an unlimited amount of records from table B. The way this is done is that a column is added to table B containing the Id of the record in table A. For instance if a data-point belongs to an environment, the environment ID is stored in an environment ID column of the data-point table.

Figure 3.6 shows the configurations of the SQL database. The diagram shows the relations of the different tables. The pressure, gas and humidity table is not shown in the diagram to keep it simple. The pressure, gas and humidity table are identical to the temperature table. The gas table contains a Co2_value and a TVOC_value instead of a single gas_value.

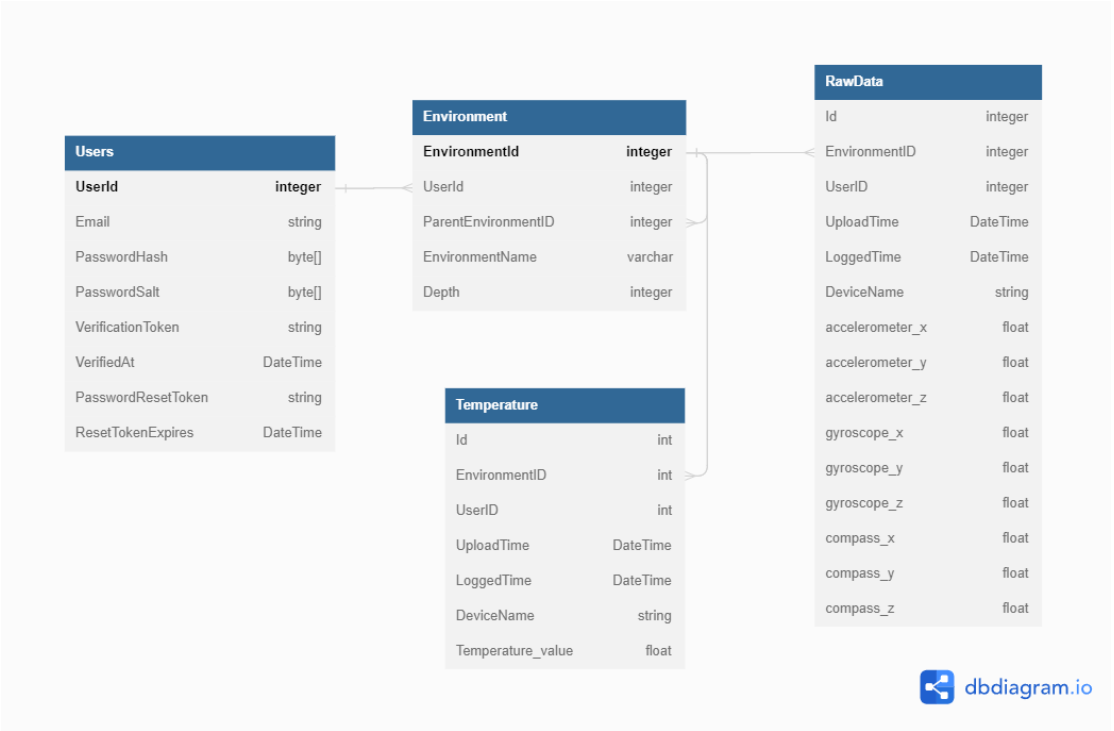


Figure 3.6: SQL Database diagram

3.3 Front-End

The front end is a website that allows the user to interface with the database from anywhere with an internet connection through a browser. The website is built using Angular. The website has the following functions.

- Allow the registration and login of users.
- Allowing the user to create and delete environments.
- Allowing the user to view, access and download the data collected by their devices.
- Working as a gateway to configure the pi on their local network.

3.3.1 Registration/Login

Registering works like most other websites. When the register button is pressed the user will be prompted to input an email and password, as well as confirming the password. When the users presses the register button an email will be sent to the email specified by the user. This email contains a link to confirm the user. Once the user has followed the link, the user can login using the email and password they specified. A demo of this is shown in [C] at 00:30.

3.3.2 Environments

Once the user is logged in, all primary-environments belonging to the user will be displayed in the environment-tree on the left side. To create a new primary environment the user can click the three dots next to the environments header and click "Create environment". This will prompt the user to input a name for the environment. Once the "Create" button is hit, the environment will be added to the server and the environment will have been created.

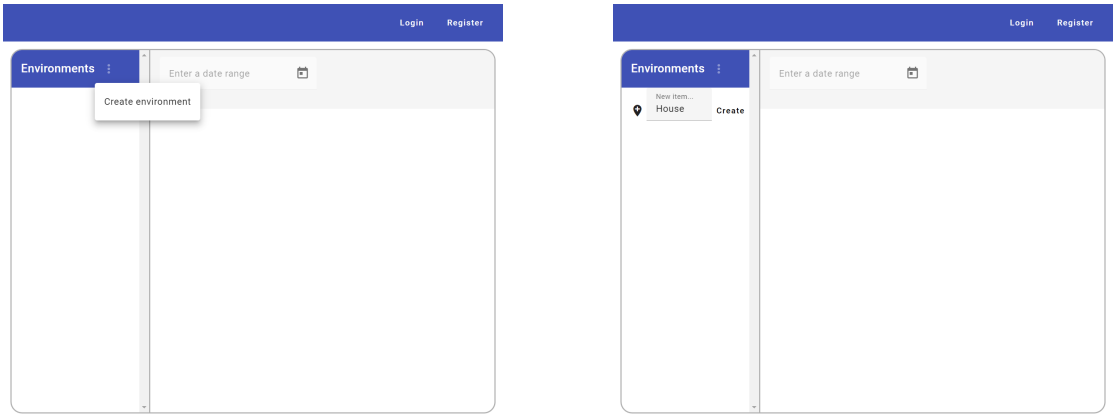


Figure 3.7: Create primary environment

Every environment has three dots next to them, with a drop-down menu, allowing the user to create a sub-environment, delete the environment or configure devices under the selected environment. Using this we can create the House environment from figure 3.1

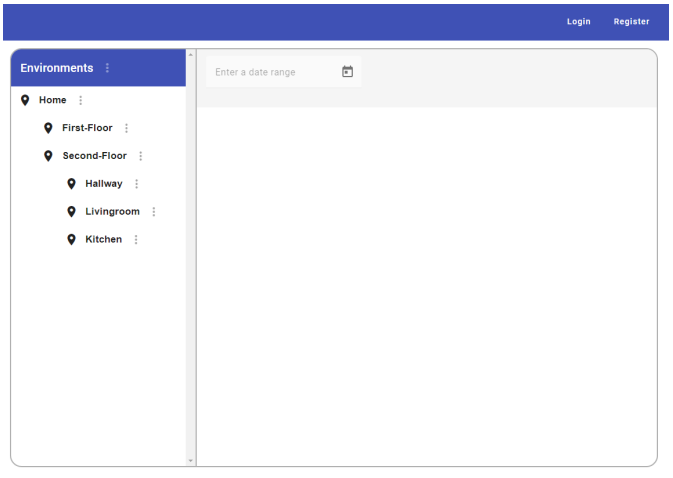


Figure 3.8

When the user wants to configure a device to write to a particular environment, if they are on the same local area network as the raspberry pi, they can select the

"configure device" option on the menu for the environment they want to write to. This will open a window to the right of the environment-tree view, that is hosted on the raspberry pi.

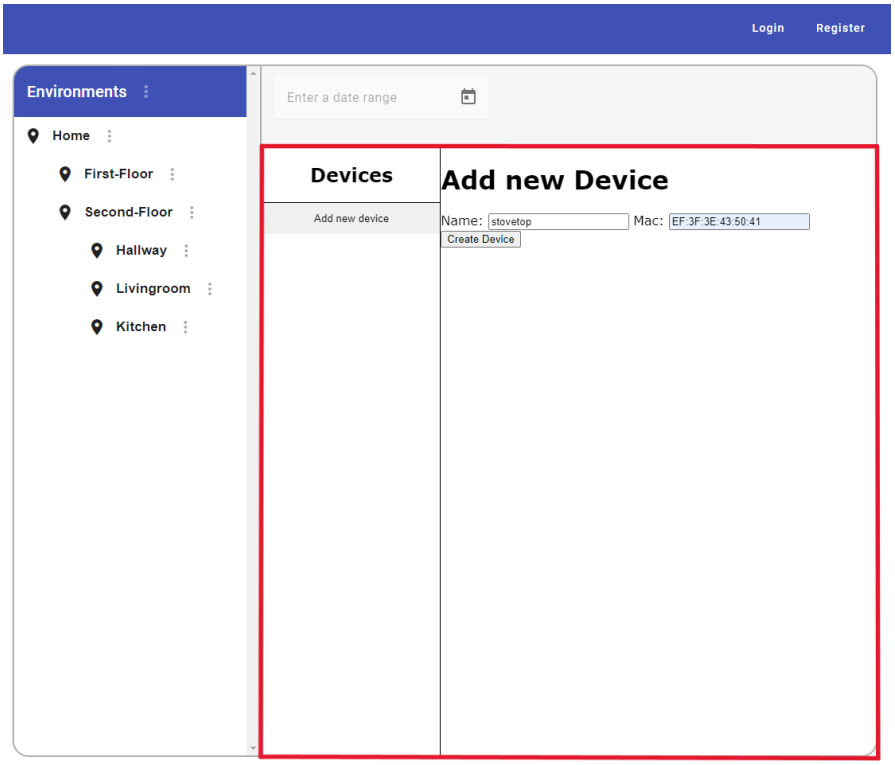


Figure 3.9: The window in the red square is hosted by the Raspberry pi

This window allows the user to configure and connect devices on the raspberry pi.(the red square itself is only on the picture and does not show up on the website) The "Device" column shows the devices already configured under the current environment in the Pi, as well as an "add new devices" option. This is described further in the raspberry pi section.

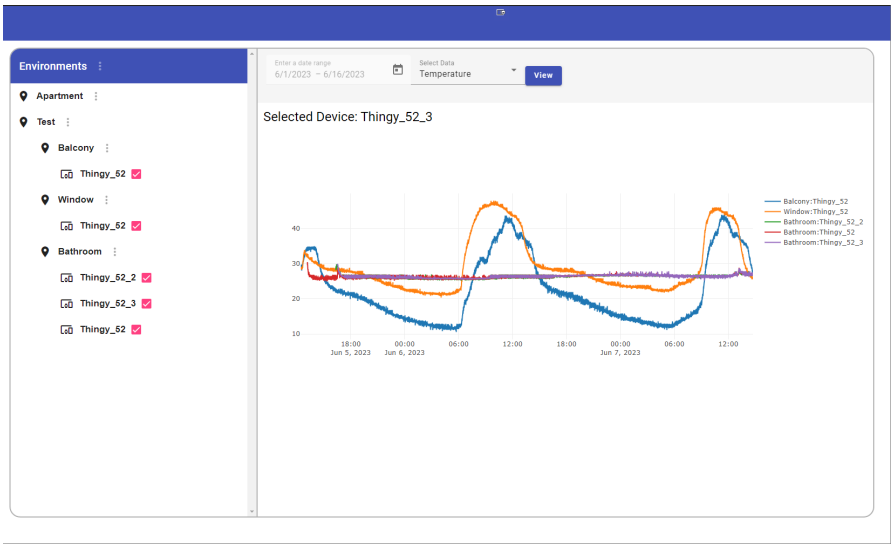


Figure 3.10

3.3.3 Viewing the data

When a device has logged data to an environment, the device will show up under that environment with at special icon as shown in figure 3.10. In order to see the data, the devices of interest can be checked with the checkbox to the right of the device. When a device is checked a graph as well as two inputs are shown in the right window as shown in 3.10. In order to see the data from the checked devices, the user can input a start and end date into the first input as well as a datatype into the second input. When the view button is hit the website will load all available data of the specified type within the time window specified in the date input.

3.4 Raspberry Pi

The raspberry pi is the base station in the research environment. The pi has three main functions:

- Connect and collect data from sensor devices.
- Serve the UI to let the user configure sensor devices.
- Interface with the central API to automatically upload data periodically.

3.4.1 Sensor Device Connection

The starting point for the raspberry pi code can be seen in [3]. This code worked by scanning all Bluetooth devices every 30 seconds. It would then look for any mac address that matched a mac address in the devices.json file. If there was a device available, the code would start a separate thread to connect and communicate with that device. This thread would continue until the code was shut down.

In order to make this code controllable through a UI. Each function a user needed to use had to be separated into a standalone function that could be ran at any time, in any order. The main four functions that the user needs access to through the UI are: connect device, disconnect device, start device recording and stop device recording. The functions were also modified to take inputs based on what sensors to activate. When a device is configured by the user, the name and environment id of the device is automatically stored by the python code.

For each package received from a sensor module, a code called a delegate is ran. This takes the data received from the device as an input. A number code at the start of the data from the device determines what sensor the data is coming from. Based on this number-code, the received data is passed on to a function which converts it to a string and stores it until it can be uploaded. In order to keep the RAM clear this data should be written to a file. However, opening and closing a file can be quite resource intensive, especially if we consider that each device can be transmitting at up to 200Hz. Instead each data-point is put in a dictionary based on the type and environment. Once every 2 seconds a separate thread loops through all data and writes it to the appropriate .csv files.

Another loop that runs every 10 seconds then loops through all active environments and tries to upload every .csv file to the server. When the server returns a 200 code, indicating that the data was received. The python loop deletes the .csv files. If the server returns an error or nothing at all, the thread continues and tries again the next cycle.

3.4.2 UI

In order to minimize the amount of tech-literacy required by the user to configure the SHPIA-system, the pi needed a UI that does not require the user to know python or how to interact with the terminal.

In order to make the system controllable trough a web interface. A server had to be hosted on the raspberry pi. The Flask Framework was selected to run the server due to its Built-in development server and debugger. Flask is also a python based framework, allowing it to easily interface with existing python code. This greatly increases the user experience and opens the system up to a lot more users.

If the raspberry pi is setup in the home of the user, with the Flask-server running. Whenever the "configure device" option is pressed on an environment, the website requests the environment-token for that environment and adds that token to the URL of the raspberry pi Flask server, shown in the red square in 3.9. The Flask server then decrypts the token to figure out the environment ID of the environment. The token is then stored in a dictionary under that environmentID. Whenever a device is configured in this window, an object is added to a list of devices. The environment ID of the environment token in the URL is stored as a parameter in this object. This way, when the data from that device needs to be uploaded. The token which needs to be included with the data is available.

Chapter 4

Experimental Evaluation

4.1 Experimental Setup

In order to test the system, the API was run in development mode on a personal computer. The router in the apartment of that computer was configured to route all incoming data that was not automatically rerouted, to the API port on that computer. This was done with DMZ-hosting function in the router. The IPV4 address of the router was then used as the URL of the API.

The Flask server on the pi was ran in development mode through the "flask run" command. This creates a development server on the local pi which can be accessed if the computer accessing the web-page has a secure shell connection(ssh) to the pi. Alternatives to this ssh approach is discussed further in the future work section.

The front-end app is hosted using the build in angular function "ng serve" to host the app on localhost:4200.

Five Thingy 52 devices were setup in the apartment and set to record pressure, temperature and humidity once every 10 seconds. As shown in 4.1, one device was setup on the balcony, this was to measure the pressure which we can hopefully compare to the local weather station measurements to get a validation of the data. Another device was set in the window next to the balcony, this would hopefully resemble the temperature data from the device on the balcony. Three devices was placed in the bathroom as far away from the pi as possible, this would test the reliability of the Bluetooth connection in less favorable conditions.

This test was left to run from 12:25 June 5. to 14:44 June 7. During this time the computer and Pi was monitored to see if the system slowed down or the memory filled up. This test was mostly setup to test the reliability of the system.

In addition to the reliability the test also gave an insight into the usability of the system. The setup from user creation to fully setup system can be seen in [C]

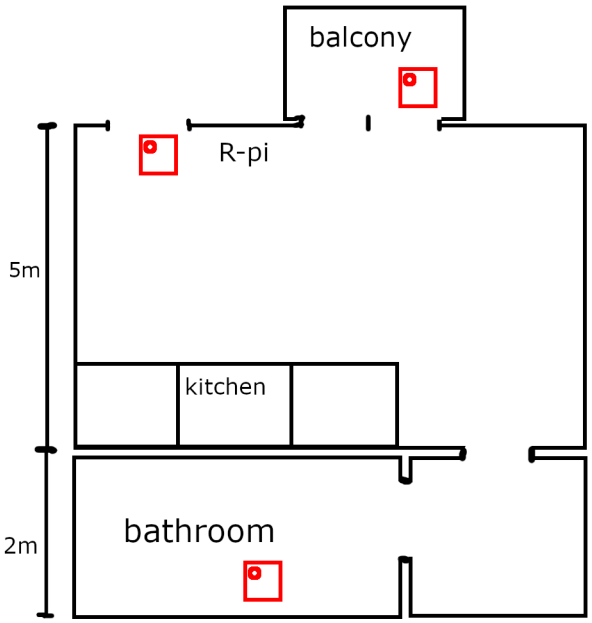


Figure 4.1: Apartment Test Diagram(red squares are devices)

4.2 Experimental Results and Analysis

Figure 4.2 shows the data collected from five sensors during the two day test. All plots shown in the experimental section was downloaded from the front-end website.

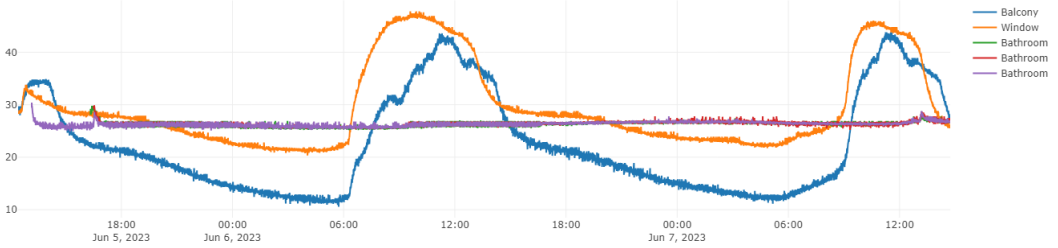


Figure 4.2: Data captured during two days test

The point of these test is not really to test the accuracy of the data collected, more so that the system worked and collected data over a period of time. We can see from the plot that the three devices that were placed in the bathroom logged a consistent temperature over the duration of the test, while the window and balcony temperature changed quite a bit when exposed to direct sunlight. During the test a brief internet outage caused an unhandled error in the python thread responsible for periodically uploading the data, causing it to shut down. This caused the pi to stop uploading the data at around 23:13 6. July 2023.

	Id	EnvironmentID	UserID	UploadTime	LoggedTime	DeviceName	Temperature_value
91110	95061	2077	1	2023-06-06 23:12:55.7641552	2023-06-06 23:12:47.0305820	Thingy_52	15,53
91111	95062	2078	1	2023-06-06 23:12:55.7932969	2023-06-06 23:12:45.4369220	Thingy_52	24,13
91112	95063	2078	1	2023-06-06 23:12:55.7933061	2023-06-06 23:12:55.4271130	Thingy_52	23,94
91113	95064	2079	1	2023-06-06 23:12:55.8216274	2023-06-06 23:12:45.7746760	Thingy_52	26,77
91114	95065	2079	1	2023-06-06 23:12:55.8216390	2023-06-06 23:12:46.9931720	Thingy_52_2	26,62
91115	95066	2079	1	2023-06-06 23:12:55.8216427	2023-06-06 23:12:48.5083770	Thingy_52_3	26,62
91116	95067	2079	1	2023-06-06 23:12:55.8216454	2023-06-06 23:12:55.7648390	Thingy_52	26,88
91117	95068	2077	1	2023-06-07 13:00:38.2432644	2023-06-06 23:12:57.0507960	Thingy_52	15,4
91118	95069	2077	1	2023-06-07 13:00:38.2432762	2023-06-06 23:13:07.0109500	Thingy_52	15,71
91119	95070	2077	1	2023-06-07 13:00:38.2432798	2023-06-06 23:13:17.0312470	Thingy_52	15,66
91120	95071	2077	1	2023-06-07 13:00:38.2432822	2023-06-06 23:13:27.0214450	Thingy_52	15,42

Figure 4.3: Temperature upload error

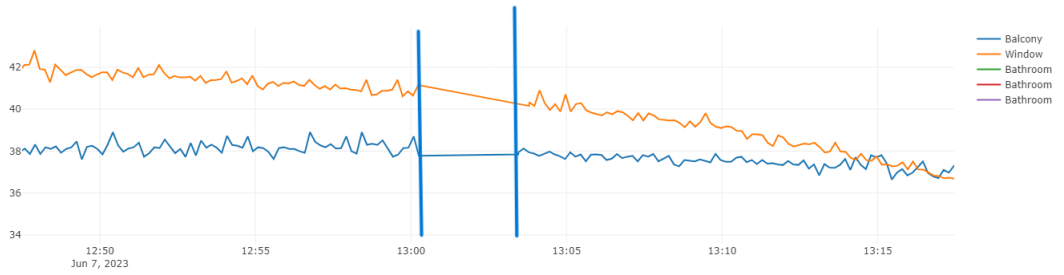


Figure 4.4: Pi system downtime

figure 4.3 shows the individual data-points of the temperature table. Above the red line the upload time is only a couple of seconds behind the logged time. However at around 23:13 the uploader stopped working. This was not noticed by anyone until the day after. However, due to the redundancy in the system, the pi continued to log the data locally. Once the error was noticed at around 12:30 the day after. The python code was stopped and restarted. The uploader then uploaded all local data-points to the server.

Although the zoomed out plot seems to show no downtime, if we zoom in at around 13:00 we can see that there was a period of around 3-4 minutes where the python code was stopped and restarted. Shown in-between the two blue lines in 4.4.

Note that the upload time in figure 4.3 show that the data was uploaded at 13:00:38. That is at the beginning of the supposed downtime in figure 4.4. This is because, once the code bug was noticed, it was fixed on another computer while the pi as still running. This meant that once the pi was shutdown it could instantly get the updated code and get back up running. When the code is restarted the devices has to be reconnected. The uploader on the other hand will instantly upload any data stored on the pi. This error served as a good test of the system and proved that even if the internet were to go out for a day the system would still keep running and gathering data.

Chapter 5

Conclusions

This paper presented a new system for logging data from smart home devices. The result is a complete system with three parts that work together whilst requiring very little technical knowledge from the user.

The server can automatically collect data from thingy 52 devices, through a raspberry pi devices over the internet in a secure and efficient manner. The front-end application allows users to collect, manage and view data in a single simple interface using browsers which most people are very familiar with.

The raspberry pi allows for easy configurations whilst hiding the complex system need to gather and transmit data in secure manner from the user. The environment feature allows the data to be sorted in an organized fashion.

The Bluetooth strength signal was abandoned once it was discovered that the bluepy library does not have the capability to measure the rssi value of devices that are paired to the pi. Adding all Bluetooth devices to the Pi was also deemed outside the scope of this paper due to time constraints. However this paper goes a long way in integrating these other devices into the system. Since the system is focused on generalization, once the python code for the pi is created where the pi can connect to the other devices. The other devices can simply put the data into the same .csv file that the existing code uses and it will automatically upload it as if it was coming from a thingy 52 device.

5.1 Future Directions

- The next logical step in the progression is to add the remaining devices outlined in the SHPIA architecture [2]. This would give the users more tools to more accurately monitor the activities of subjects.
- The addition of devices using beacon Bluetooth mode rather than a pairing mode would also open up the possibility to monitor the signal strength of the device while gathering data.
- The raspberry pi server running of the flask development server is not ideal as it is not designed to be used in production. Alternatives to this approach was looked into but other solutions required a lot of configurations of the pi itself, which would greatly increase the tech-literacy requirement of the system. A solution where and image of a pi with this configuration preinstalled would solve both these problems but might require a lot of development to get working.
- Currently the SQL database is being under-utilized. SQL has incredibly powerful search algorithms which could be extremely useful for researchers when looking for patterns in data. However currently the only queering the server allows is sorting based on date. If the server could open up the query algorithm in a secure and usable manner this could provide a invaluable tool for researchers.
- Adding a table to the SQL database allowing the users to specify other users they want to give access to their environment would allow people to share data. currently only the person who was logged in when the device was configured has access to the data.
- The Front-end although more than useful is still a bit rough around the edges. There are a few quality of life improvements that would make the website more user friendly. Some examples of this is, missing logout button, missing refresh environment tree button, no support for cookies which means users needs to login every time the site refreshes. These are not detrimental to the function of the website but does lead to a worse user experience.
- There is currently a bug in the bluepy code which does not allow the thingy 52 to send gas data to the pi. The error specifies that the data received from the device is too long.

Appendix A

Test setup video

https://www.youtube.com/watch?v=AJ-FKfqY0XYab$_c hannel = LarsPan$

Appendix B

back-end github repositoriy

https://github.com/ghettolash/smarthomeAPI2

Appendix C

Front-end github repositoriy

https://github.com/ghettolash/SHPIA-frontend/tree/master

Bibliography

[1] Mara Z. Vitolins1 and Talsi L. Case2. What Makes Nutrition Research So Difficult to Conduct and Interpret? — ncbi.nlm.nih.gov. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7228817/, 2020. [Accessed 08-Jun-2023].

[2] Florenc Demrozi and Graziano Pravadelli. *SHPIA: A Low-Cost Multi-purpose Smart Home Platform for Intelligent Applications*, pages 217–234. 10 2022. ISBN 978-3-031-18871-8. doi: 10.1007/978-3-031-18872-5_13.

[3] Christian Turetta. Wearable-data-collector. https://github.com/CristianTuretta/Wearable-Data-Collector/tree/dev-cristian, 2023.

[4] Lars-Erik Nes Panengstuen. Wearable-data-collector. https://github.com/CristianTuretta/Wearable-Data-Collector/tree/dev-lars, 2023.

[5] Diane Cook, Aaron Crandall, Brian Thomas, and Narayanan Krishnan. Casas: A smart home in a box. *Computer*, 46, 07 2013. doi: 10.1109/MC.2012.328.

[6] Narayanan Krishnan and Diane Cook. Activity recognition on streaming sensor data. *Pervasive and mobile computing*, 10:138–154, 02 2014. doi: 10.1016/j.pmcj.2012.07.003.

[7] Marjorie Skubic, Gregory Alexander, Mihail Popescu, Marilyn Rantz, and James Keller. A smart home application to eldercare: Current status and lessons learned. *Technology and health care : official journal of the European Society for Engineering and Medicine*, 17:183–201, 02 2009. doi: 10.3233/THC-2009-0551.

[8] Jiaxuan Wu, Yunfei Feng, and Peng Sun. Sensor fusion for recognition of activities of daily living. *Sensors*, 11, 11 2018. doi: 10.3390/s18114029.

[9] Ju Wang, Nicolai Spicher, Joana M. Warnecke, Mostafa Haghi, Jonas Schwartze, and Thomas Deserno. Unobtrusive health monitoring in private spaces: The smart home. *Sensors*, 21:864, 01 2021. doi: 10.3390/s21030864.

[10] Online security survey google % harris poll%$_2$019, $Feb2019.URL$.