Universidade do Minho
Escola de Engenharia

Communication and Monitoring System for People with Alzheimer and Dementia

CMS

Afonso Oliveira, PG53599
Gonçalo Moreira, PG53841

Professor:
Adriano Tavares

Embedded Systems Project

September 2023

Contents

List of Figures

List of Tables

1 Introduction

Alzheimer's is a neurodegenerative disease that causes the progressive loss of mental functions, characterized by the degeneration of brain tissue, including the loss of nerve cells, the accumulation of an abnormal protein called beta-amyloid, and the development of neurofibrillary tangles. On the other hand, dementia can be caused by various conditions, including cerebrovascular diseases or impaired cerebral blood flow. Although memory is affected in both cases, the manifestations vary.

Alzheimer's disease causes memory loss, amnesia, a deterioration in the ability to speak, difficulties in performing daily activities, as well as behavioral and psychological changes. In contrast, dementia also affects memory but usually less aggressively than Alzheimer's. The speed of information processing is reduced and may worsen over time. Additionally, other problems can be caused by undetected strokes or cerebrovascular accidents.

The incidence and prevalence of Alzheimer's and dementia increase with age, doubling every 5 years after the sixth decade of life. As Portugal has an aging population, it is possible to predict an increase in cases of dementia and Alzheimer's over the years.

Based on the 2021 census data, out of 10,343,066 residents in Portugal, 2,423,639 are 65 years old or older, an age group where these diseases are more relevant. According to the "Health at Glance 2017" report from the OECD, Portugal ranked 4th in dementia prevalence, with 19.9 cases per 1,000 inhabitants. This implies an estimate of 205,827 people in Portugal affected by these diseases.

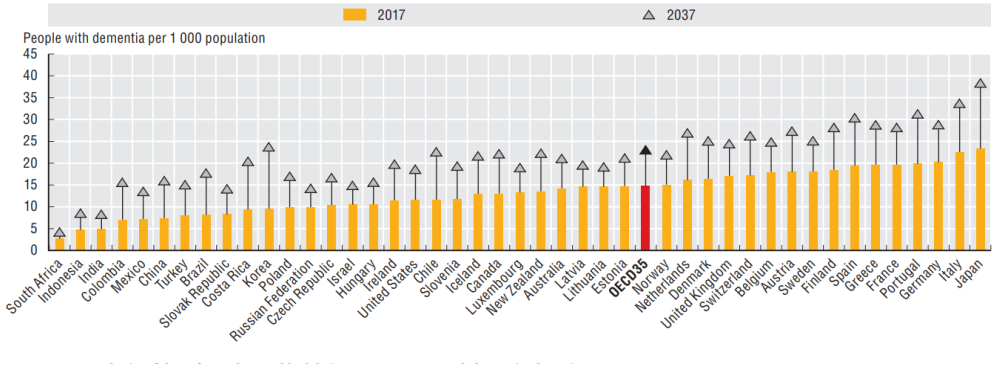


Figure 1: Dementia expected growth till 2037

Despite the significant number of affected individuals, awareness of Alzheimer's and dementia is still limited in the general population. This can result in unidentified cases due to lack of information or awareness of the symptoms of these diseases. It is important to emphasize the importance of education and support for individuals affected by these conditions and to raise awareness in society about these diseases to improve early diagnosis and treatment.

1.1 Problem Statement

Overall, this project will enhance the safety of individuals with Alzheimer's and Dementia by implementing various security systems that can promptly alert their caregivers to unusual occurrences during their moments of solitude. It will provide caregivers with increased flexibility in their schedules while ensuring they remain well-informed about the well-being of the individuals under their care. Additionally, it allows for the sick to have a the ability to feel more independent while remaining safe.

1.2 Goals

In addition to applying the knowledge gained from prior curriculum units, the group aspires to reinforce their understanding of the embedded systems specialization and explore fresh concepts within the realm of embedded technology. Therefore, the goals outlined for this project are the following:

- Ensure a reliable system
- Build a compact and discrete system to minimize it's influence in the house interior design
- Explore a market niche that we believe to be overlooked by other companies

2 Market Analysis

To evaluate the project's viability effectively, we must begin with a concise market analysis. This involves two primary phases: first, conducting thorough market research to understand the market's nuances, and trends, and second, defining the market's specific boundaries and segments.

Once the market is well-defined, we proceed by analyzing similar products currently available in the market. This includes a close examination of their features, functionalities, and use cases, followed by a comparative assessment against our proposed project. This comparative analysis will provide valuable insights into market gaps, opportunities, and the competitive landscape, informing our project's strategic direction.

2.1 Market Research

The home security market in Portugal has rapidly grown, reaching an estimated value of nearly 1 billion euros in 2021, with home security accounting for about 20%. Globally, this market was valued at 51.9 billion USD in 2021, and it's projected to reach 106.3 billion USD by 2030, with an annual growth rate of 8.6%, this can be seen in the next figure. This remarkable growth can be attributed to increasing security concerns and the integration of IoT technology for real-time data monitoring in households.

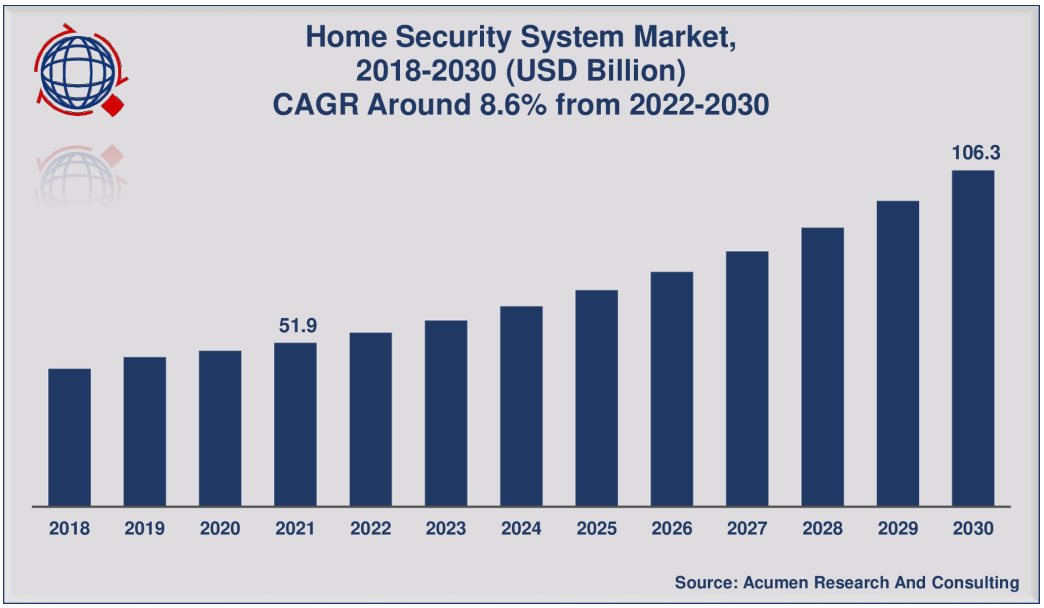


Figure 2: World Home Security Market

In conclusion, the market research for this home security and monitoring product designed to provide safekeeping and assistance to individuals with dementia, Alzheimer's, and other degenerative illnesses is grounded in a thorough analysis of the home security market. This approach allows us to gain valuable insights into the market dynamics, distribution channels, and end-user segments, which in turn helps us ascertain the viability of our project.

2.2 Market Definition

In light of the anticipated growth in degenerative diseases and the often overwhelming challenges they present, along with the constraints faced by many families in affording specialized caretakers, our product emerges as a solution. A bridge connecting these two crucial aspects of care.

For countless families grappling with the impact of Alzheimer's and dementia, the desire to provide their loved ones with the best care and attention is profound. However, this desire is often hindered by practical limitations, including the inability to be physically present at all times or the financial constraints or the the loved one being against with hiring full-time caregivers.

Our product serves as a middle ground, offering a adaptable approach to address these challenges. Firstly, it facilitates a deeper connection between family members and their loved ones. Through livestream communication capabilities, it transcends physical barriers, enabling real-time interaction and engagement. Loved ones can check in, converse, and ensure the emotional well-being of their family member, even when they are physically distant.

Secondly, our product enhances safety and security within the home environment. Leveraging the advancements in the booming security market, it grants family members control over various aspects of the home, from monitoring entry points to managing appliances. This not only fosters a sense of reassurance but also empowers families to create a safer living space for their loved ones.

2.2.1 Similar Products

While there isn't an existing product that precisely matches the main purpose of our project, the Ring Alarm Security system shares certain similarities. Ring Alarm Security includes some components that overlap with our project, such as security features and sensors. However, it lacks the core elements that differentiate our project, namely bidirectional communication and the ability to control various variables within the home.

Figure 3: Ring Alarm Security

Advantages	Disadvantages
• Widely available	• Expensive
• High video quality	• Does not allow for bidirectional communication
• Easy to use	• Meant for security

3 System Overview

In the next figure, Figure 4, it's exposed the main components of the project, they can be divided in two system, the house system that contains the sensors, the user interface and raspberry, and the other system that is the mobile app in which the caretaker can see information needed to track and care, receive notifications and communicate with the person in care.

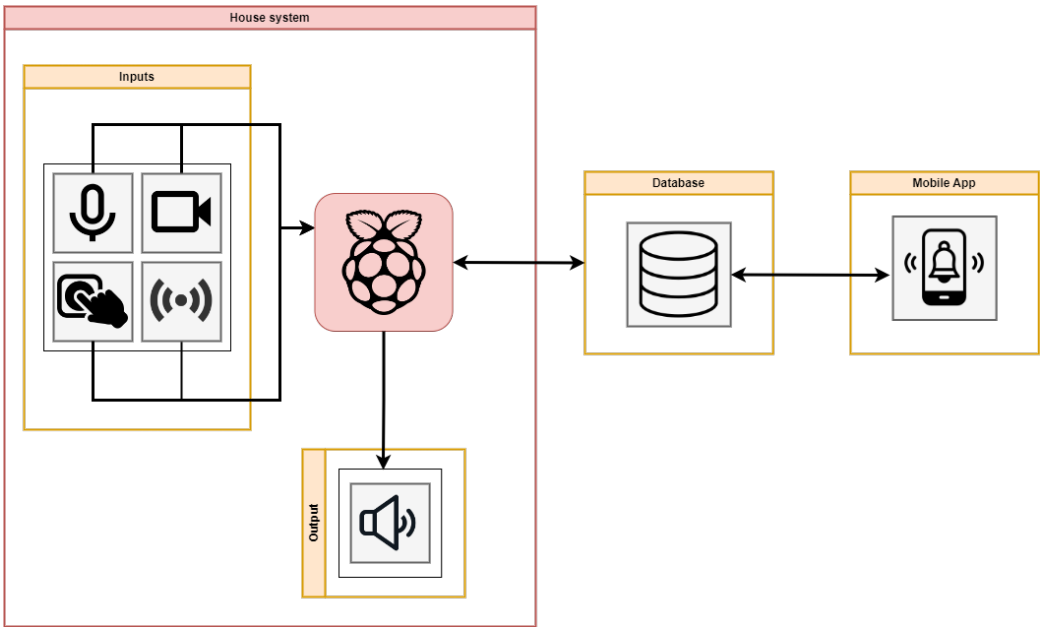


Figure 4: System Overview Diagram

The house system comprises four input devices, one output device, and one controller. These components work seamlessly together to create a comprehensive solution for ensuring the well-being and safety of individuals with Alzheimer's or dementia within the home.

Input Devices

- Camera: A camera is integrated into the system to provide visual monitoring of the person in need. It serves the dual purpose of enabling visual communication and continuously monitoring the individual.
- Microphone: The microphone facilitates two-way communication between the person in need and the caretaker, enhancing connectivity and enabling real-time communication. It also detects audio anomalies that may signal potential danger or accidents.

- Motion Sensor: The motion sensor is a crucial component for detecting movement within the house. It contributes to monitoring the individual's activities and enhancing overall security.
- Pressure Button: The pressure button serves as a last-resort communication method for the person in need. It allows them to signal for assistance in situations where other communication methods may not be feasible.

Controller

The controller serves as the central processing unit of the system. It receives inputs from the sensors, processes the data, and uses it as needed. It has the capability to stream live video and audio to a mobile app, enabling remote monitoring and assistance.

Mobile App

The mobile app is designed to empower caretakers with the tools and information they need to provide effective aid and support. Its features include:

- Sensor Log: The app maintains a log of sensor-triggered events, helping caretakers track the individual's activities and identify patterns.
- Sound Anomaly Detection: The app alerts caretakers to potential sounds that may require attention, enhancing safety and vigilance.
- Expandability: The app is designed to accommodate future sensors or actuators, ensuring scalability and adaptability as needs evolve.
- Live Stream: Caretakers can access a live video and audio stream from the house, enabling real-time monitoring and communication with the person in need.

This house system not only ensures the safety and well-being of individuals with care needs but also gives power to the caretakers with the tools and information required to provide timely and effective assistance. This product combines technology, communication and monitoring to enhance the quality of life for those in need and peace of mind to the caregivers.

3.1 System Architecture

System architecture is a conceptual framework that delineates the organization, behavior, and various perspectives of a system. It serves as a blueprint for arranging and coordinating system components and subsystems, ensuring their collaborative efforts in realizing the overarching system objectives.

This system architecture can be categorized into two fundamental aspects: hardware and software architecture, which will be elaborated upon in subsequent sections.

3.1.1 Software Architecture

The software architecture is divided into three main layers: The low-level layer, that is represented by the custom Linux operating system, the middlelevel layer, represented by the middleware and the higher-level layer, represented by the application layer.

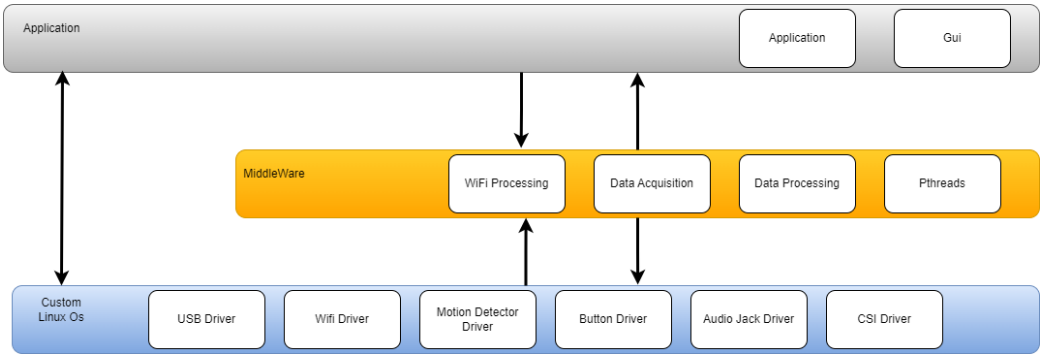


Figure 5: System Overview Diagram

3.1.2 Hardware Architecture

The hardware architecture comprises five input components: a Microphone (MIC), a motion sensor, a pressure button, a magnetic sensor and a camera. These input devices are responsible for collecting various types of data, such as audio input from the microphone, motion detection from the motion sensor, button presses from the pressure button, detection if the door is open, and visual information from the camera. They serve as sensory input sources for the system.

These input components communicate with a Raspberry Pi, which acts as the central processing unit. The Raspberry Pi serves as the intermediary and controller for processing and managing the data from the inputs. It receives data from the MIC, motion sensor, pressure button, and camera and processes this information as required by the system's logic.

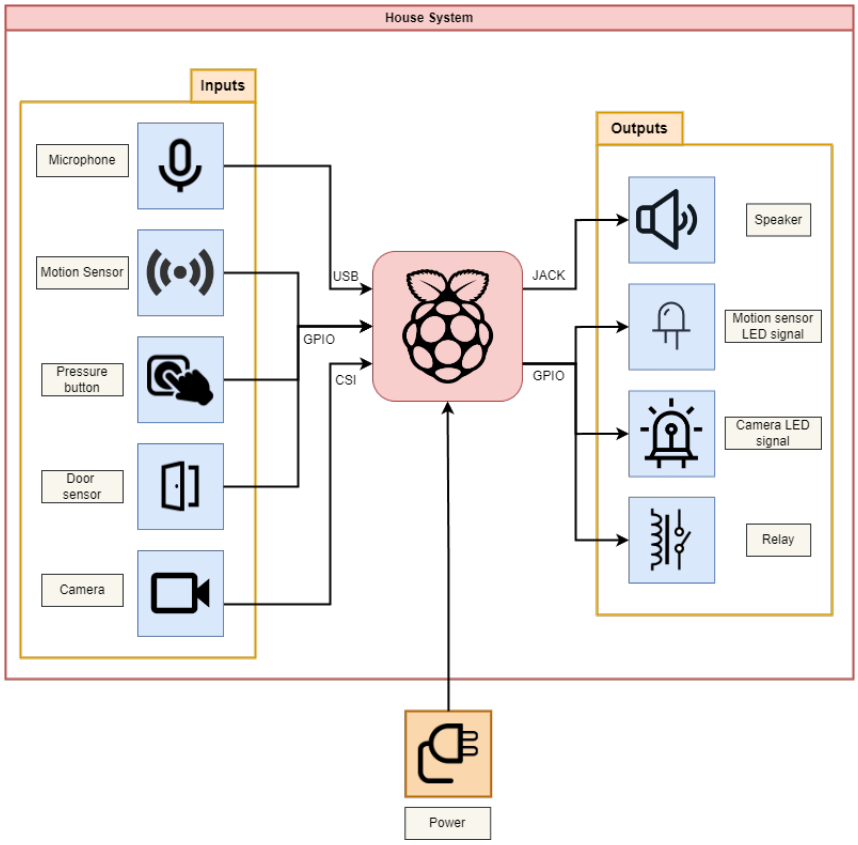


Figure 6: Hardware Architecture Diagram

Furthermore, the Raspberry Pi is connected to four output components: two LEDs (Light Emitting Diodes), one speaker and one relay. These output devices serve as the means through which the system can provide feedback or take actions based on the data and logic processed by the Raspberry Pi. The LEDs can display visual indicators, the speaker can produce audio output, making them essential for conveying information or interacting with the environment, the relay enables the control of other devices remotely. This hardware architecture forms the foundation for the system's sensory input, processing, and output capabilities.

3.1.3 Technology Stack

A technology stack diagram provides a comprehensive view of both the hardware and software architecture of a project. It serves as a visual representation that simplifies the understanding of project components and their interconnections.

The following diagram explains how this project hardware and software will interact with each other.

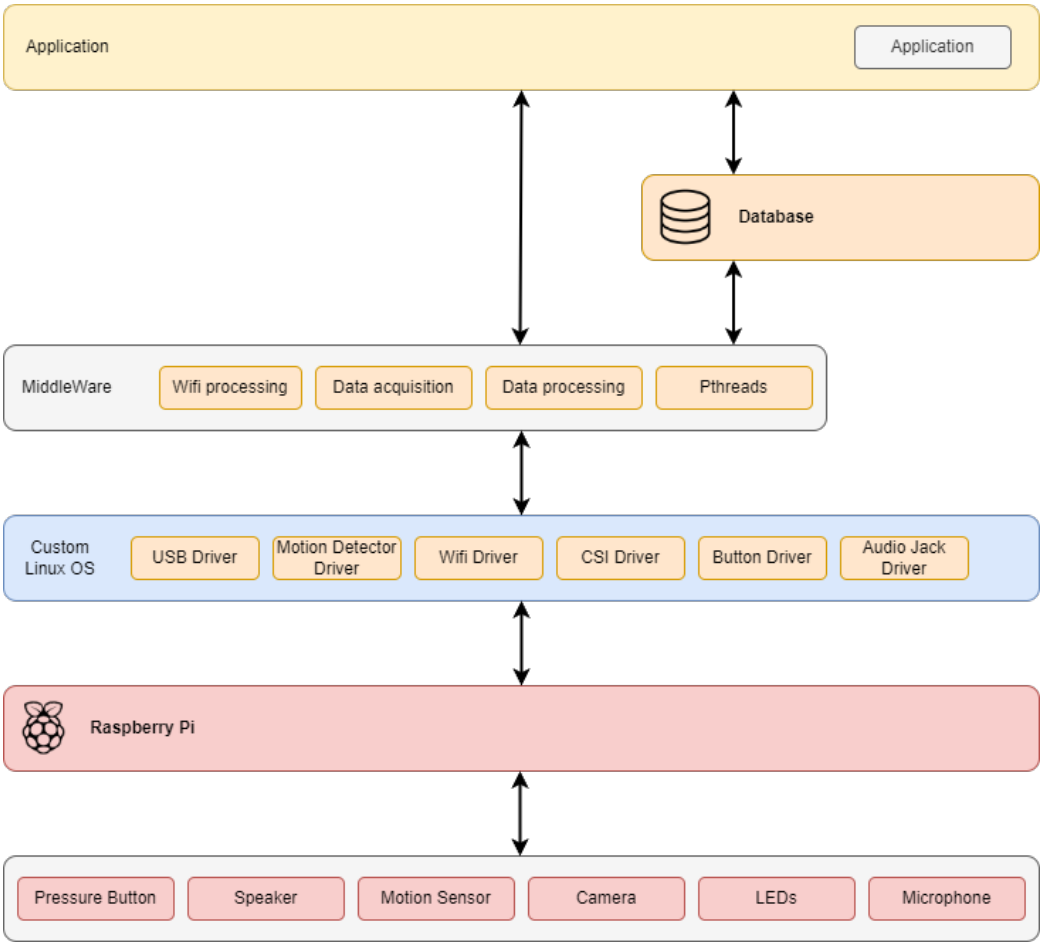


Figure 7: Technology Stack diagram

3.2 System Requirements and Constraints

System requirements and constraints are crucial in the development and implementation of projects and systems for several reasons like clarity and scope definition, alignment with objectives, risk mitigation and others.

System requirements specify what a system or project must achieve, encompassing its functionalities, performance criteria, and user expectations. These requirements are essential for guiding the development and ensuring that the final product meets its intended purpose.

3.2.1 Functional Requirements

- Stream Live Video
- Bidirectional Communication
- Detect Movement
- Notify the Caregiver about unusual happenings
- Record Video

3.2.2 Non-functional Requirements

- Reliability
- Intuitive user interface
- Scalability for future upgrades

System constraints, on the other hand, are factors that limit the development or operation of a system. These limitations must be recognized and managed to ensure the project's success.

3.2.3 Technical Constraints:

- Raspberry Pi
- C/C++
- Buildroot
- Linux
- Pthreads
- Device driver
- Multi-tasking

3.2.4 Non-Technical Constraints:

- Two member team
- Project deadline at the end of semester
- Low budget

4 System Analysis

In this section, we will delve deeper into the system, providing a more comprehensive examination. Given that our system encompasses both local and remote components, we will conduct a detailed analysis of each.

4.1 House System

4.1.1 Events

Events play a pivotal role in shaping system behavior. The system's capability should extend beyond merely detecting the presence of an event; it should also possess the ability to discern and process these events to generate the desired outcomes. The following Table 1, outlines the system events.

Event	System Response	Source/Trigger	Type
Camera Sampling	Read Camera Field	Timer	Synchronous
Button pressed	Communicate with database	User	Asynchronous
Door Open or Closed	Communicate with database	User	Asynchronous
Enable/Disable Relay	Communicate with database and change relay status	User	Asynchronous
Motion Detection	Communicate with database	User	Asynchronous
Store Data	Store sensors' data	User	Asynchronous
Send Data	Communicate with database	Timer	Synchronous

Table 1: Houses System Events

4.1.2 Use Cases Diagram

The House system use cases are shown in figure 7. A use case diagram serves as a visual representation of the potential interactions between a system and its users. It illustrates a range of use cases, highlighting the various types of interactions that different users may have with the system. Often, these use case diagrams are complemented by other types of diagrams for a more comprehensive system representation. These diagrams hold significant utility as they aid in clarifying the expected behavior of the system and facilitate the visualization of interactions between the system's actors and the system that is under development. Each use case within the diagram represents a distinct pathway by which users can employ the system, each resulting in a specific outcome or result.

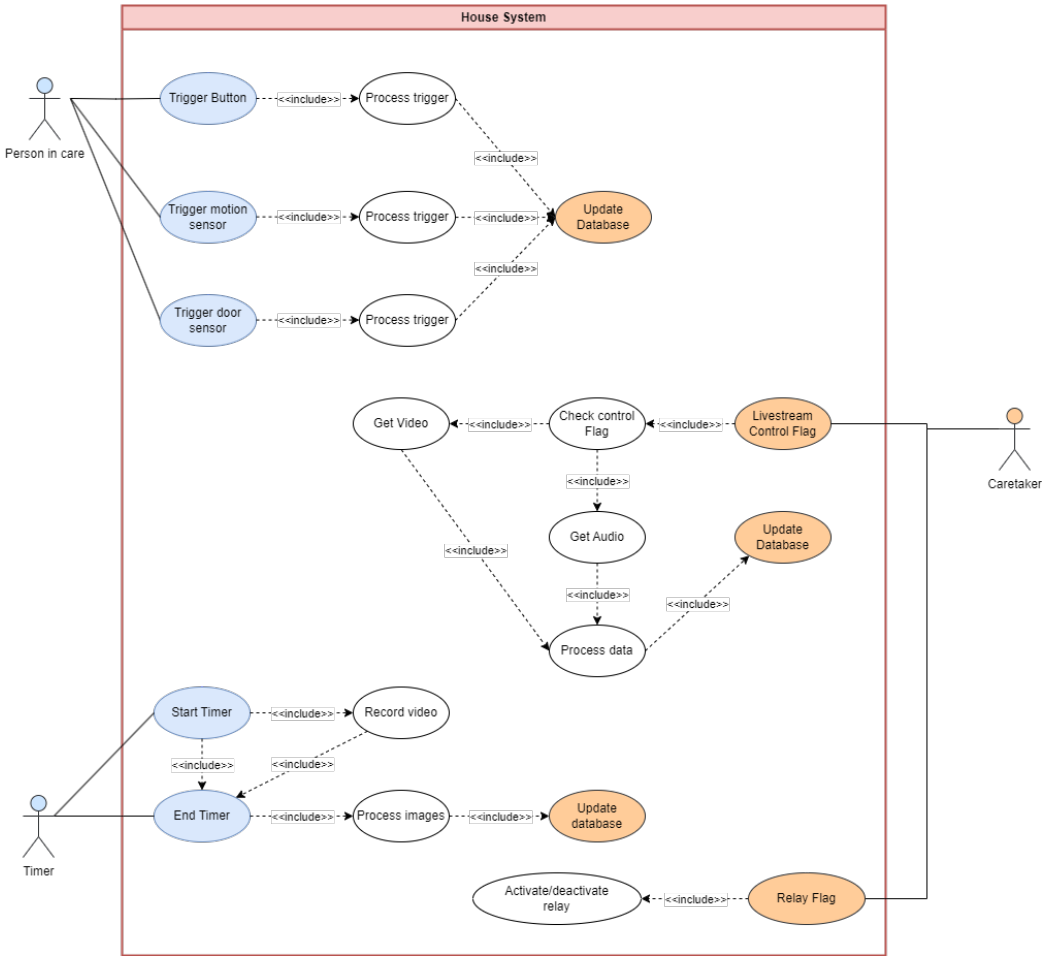


Figure 8: Use Cases Diagram

4.1.3 State Chart Diagram

Using a State Chart diagram, one can model the possible states for the system in question, and specify how state transitions occur as a consequence of occurring events. Figure 8 shows the State Chart diagram of the Local System. There is an initial system configuration after there is a Power ON, after that the system waits for triggers that come in 3 different forms. The first is the timer trigger that will process the Video and audio to be streamed. The second and third conditions are the sensors flags and the button pressed that will each act on their own way and make the system process the data produced by them.

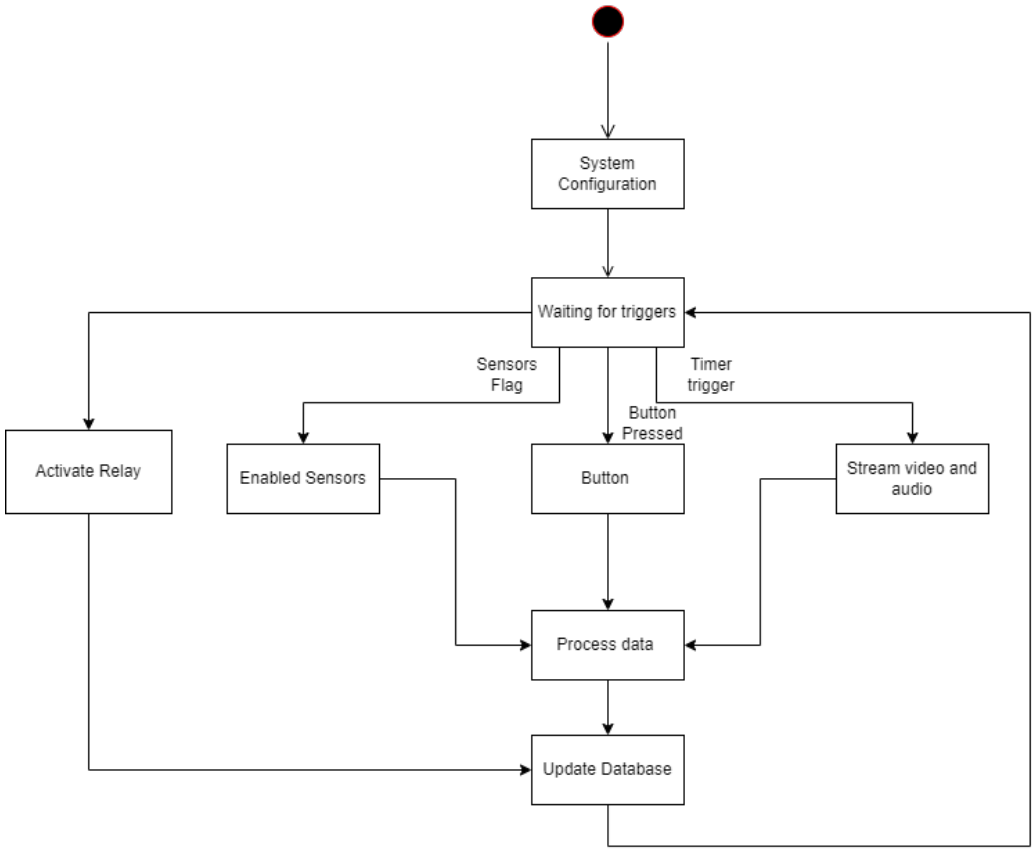


Figure 9: Local System State Machine Diagram

4.1.4 Sequence Diagram

In the sequential diagram depicted below, its illustrated the dynamic interactions within the house system with its external actor, the person in care and the database. The person in care plays a pivotal role in activating two specific actions within the house system: motion sensor activation and pressing the pressure button. Importantly, these actions are asynchronous, implying they can be initiated at any moment, independent of a fixed schedule.

Additionally, the diagram outlines an interaction that grants access to the camera and microphone. This access serves a dual purpose: enabling real-time video and audio communication and configuring settings for scheduled audio and video recording. All of the data gathered it's then sent to a database that will be the connector between the mobile app and the house system.

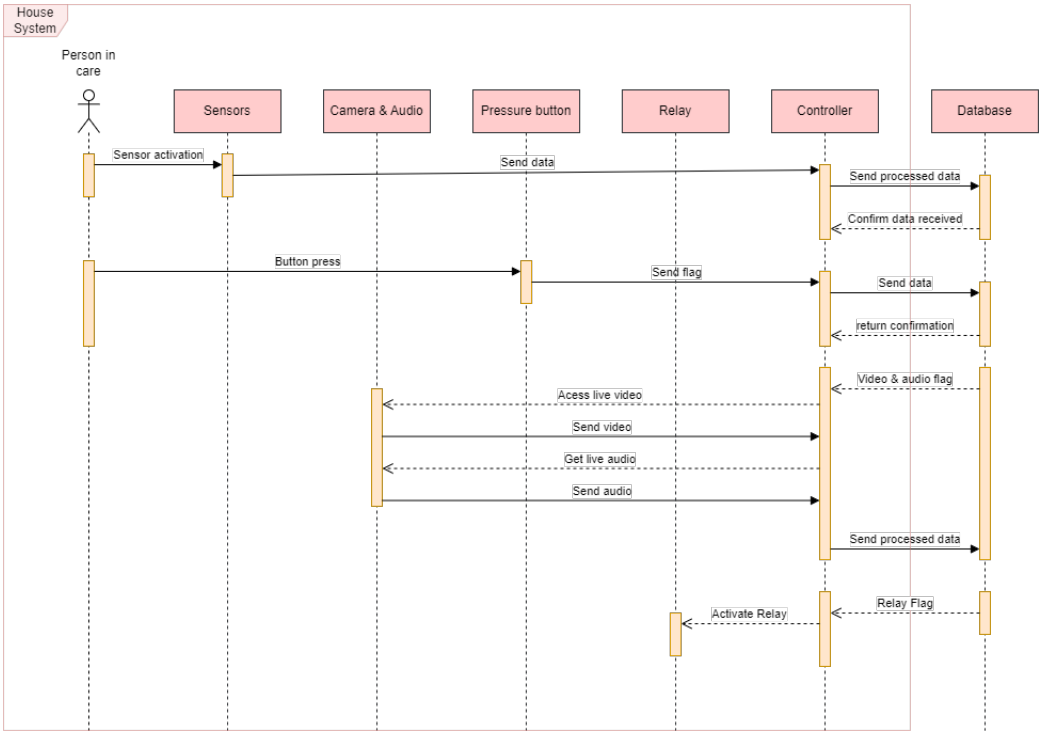


Figure 10: Local System Sequence Diagram

4.1.5 Estimated Budget

The following table portrays an estimated cost of this project with all of his components.

Component	Estimated Cost (€)
Raspberry pi 4 model B	63
Camera	22
Microphone	9
Relay	2,45
Magnetic Sensor	2,9
Motion sensor	5
Speaker	12
Pressure Button	1
LED x2	0,10
Model	10
Total	**127,45**

Table 2: Estimated cost of the House System

4.2 Remote System

4.2.1 Events

To gain a more profound comprehension of the system, it becomes imperative to identify potential events that may transpire. It is equally essential to elucidate how the system will react in response to these events, determine their root causes, and classify their types. Table 3 furnishes a comprehensive catalog of events that may manifest in the remote system, shedding light on these critical aspects for thorough understanding and analysis. In this table all of the events are asynchronous because they are triggered by the user.

Event	System Response	Source/Trigger	Type
Login	User information verification and Login to the app	User	Asynchronous
Register	Register user information on database	User	Asynchronous
App Notification	Create an app notification	User	Asynchronous
Sensor Logs	Show table of sensor activation	User	Asynchronous
Show Livestream	Open Livestreaming Window	User	Asynchronous
Relay Control	Communicate to database	User	Asynchronous
Send Audio	Record audio and send to database	User	Asynchronous

Table 3: Mobile App Events

4.2.2 Use Cases Diagram

Use case diagrams are invaluable tools as they serve to elucidate the anticipated behavior of the system, enabling a clear understanding of how it should function. Additionally, these diagrams provide a visual framework for comprehending the interactions between the system's actors, typically its users, and the system that is currently in the development phase. Within the diagram, each individual use case delineates a unique pathway through which users can engage with the system, ultimately yielding specific and predefined outcomes or results.

The Remote system use cases are shown in Figure 11. In the figure it is represented that the main actor which is the App User. The user has a series of action that he can perform that communicate with the database and its this communication that enables the control of the sensors and how the user is notified.

Figure 11: Mobile App Use Cases Diagram

4.2.3 State Chart Diagram

In Figure 11 there is a representation of the remote system state machine. This state machine is helpful to better understand how the mobile app will work and all the steps it can go through at any certain point, such as next states and a sequential view that can lead to a deeper understanding of this part of the system. There are two main states that are the Home Screen and Landing Page in which the big difference between them is the user being logged in or not.

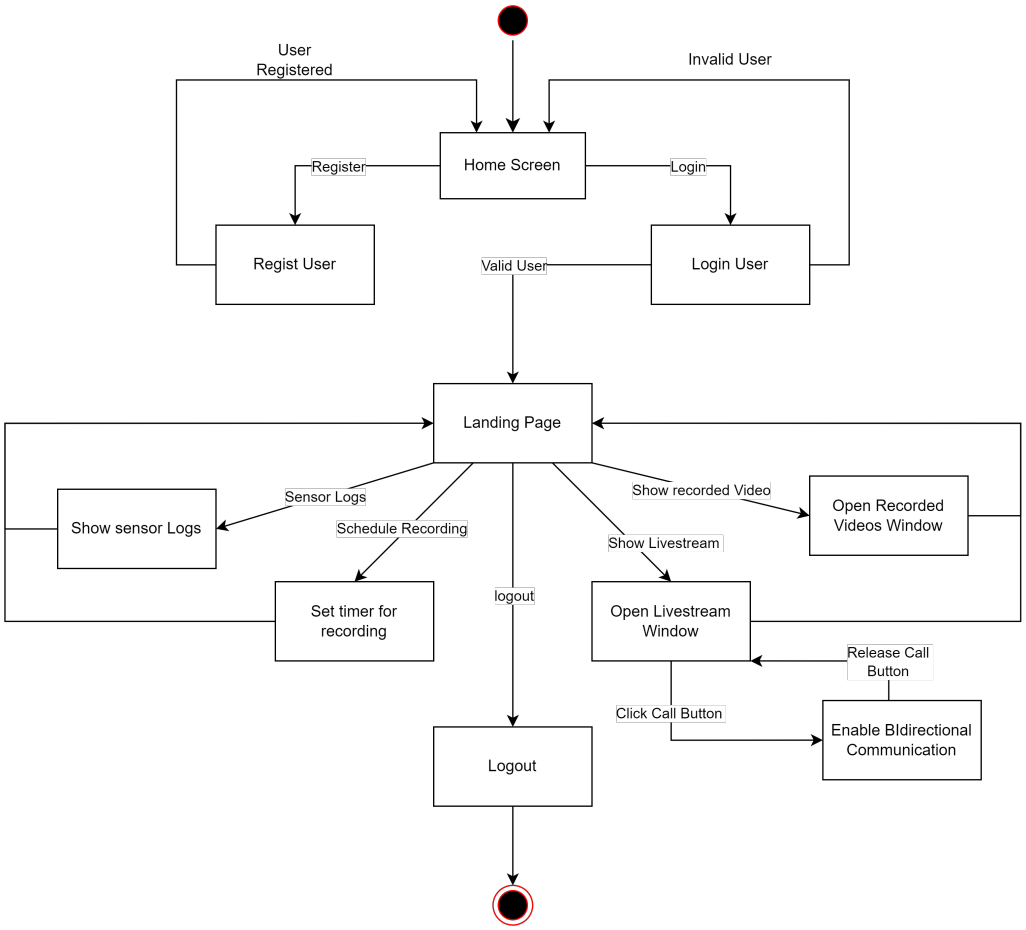


Figure 12: Remote System State Machine

4.2.4 Sequence Diagram

In the following sequential diagram, its offered a depiction of the mobile app's functionalities and their interactions with both the user and the database. The sequence begins with the user's initial actions, which involve creating an account and subsequently logging in.

Upon successful login, users gain access to a range of processes and receive notifications from the house system's sensors, particularly when the button is pressed. These processes encompass viewing sensor history, accessing recorded media, initiating live video and audio communication, and scheduling media recording.

Furthermore, the app includes a user-friendly logout feature to ensure secure access control.

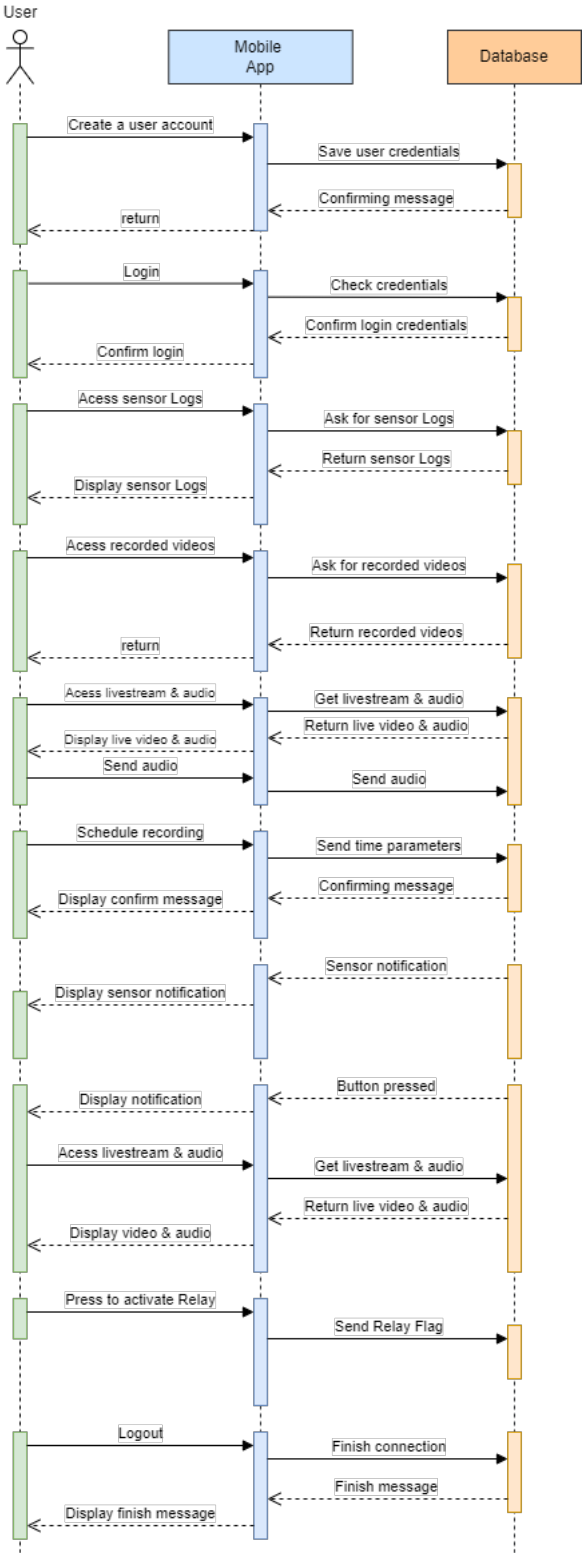


Figure 13: Remote System Sequence Diagram

5 Theoretical Foundations

5.1 Communication Protocols

5.1.1 Wi-fi

IEEE 802.11, widely recognized as Wi-Fi, is a component of the IEEE 802 series of Local Area Network (LAN) standards. It defines the collection of protocols for Media Access Control (MAC) and physical layer functions necessary to establish Wireless Local Area Network (WLAN) communication. This communication spans a broad range of frequencies, extending from 2.4 to 60 GHz.

TCP-IP

The Transmission Control Protocol (TCP)/Internet Protocol (IP) stands as one of the most widely employed protocols for internet-based communication. In this tandem, TCP is tasked with the role of gathering and reconstructing data packets, while IP ensures that these packets reach their intended destinations. These protocols are integral components of the OSI model, depicted in Figure 13.

Within this framework, TCP offers services to the transport layer, ensuring the dependable and sequenced delivery of data. TCP can be likened to a meticulous courier, guaranteeing the safe and ordered delivery of information. In contrast, the User Datagram Protocol (UDP) handles data transport without the assurance of guaranteed data delivery or acknowledgments. While this approach might be faster, it comes with the trade-off of potential unreliability.

The IP component of the TCP/IP protocol operates at the network layer, facilitating the availability of origin and destination addresses necessary for routing data across networks. This ensures that data finds its path to the intended recipients in the vast landscape of the Internet.

5.2 RTMP

RTMP, or Real-Time Messaging Protocol, is a technology that enables live video streaming over the internet. It's based on a reliable connection method called TCP, which ensures stable and error-free communication. This makes it suitable for real-time applications, such as live video streaming, where maintaining a consistent and smooth connection is crucial. RTMP is commonly used in platforms like live video streaming services and online gaming to deliver content with low latency and high quality.

5.3 CSI

CSI (Camera Serial Interface) is a technology designed for efficient camera-to-device communication. It enables high-speed data transfer between a camera and a host device with minimal CPU load. This is essential for real-time and high-quality video and image capture.

Raspberry Pi 4 utilizes CSI to connect to compatible camera modules. Its dedicated CSI connector allows for high-speed image and video data transfer, reducing strain on the Raspberry Pi's processor. The platform offers software support and APIs for easy camera control, making it versatile for applications like photography, video capture, and computer vision. Raspberry Pi's CSI compatibility ensures streamlined camera integration, ideal for a variety of projects.

5.4 Daemons

In the context of embedded systems, a "daemon" is a background process or service that operates independently of user interaction and runs without a controlling terminal. Daemons are commonly used for several purposes, such as managing system processes, providing network services, monitoring hardware, or automating specific tasks.

Daemons are often initiated at system startup and continue running until the system is shut down. They play a critical role in automating and managing various system tasks, improving efficiency and reliability. A common naming convention for daemons is to end their names with the letter "d," making them easily recognizable, like "httpd" for a web server daemon or "sshd" for an SSH server daemon.

Each daemon process belongs to a process group, and multiple process groups form a "session." The session concept allows multiple jobs to be controlled through a single terminal, with one in the foreground and others running in the background. To become a daemon, a program typically follows several steps:

- It forks a child process, with the parent exiting and the child continuing.
- The child process calls setsid() to initiate a new session and detach itself from any controlling terminal.
- Set the umask to control the permissions of files and directories it creates.
- Change the process's current directory, typically to the rootdirectory (chdir('/')). There is a need for this because a daemon normally works until system shutdown, if the daemon's current working directory is on a file system other than the one containing '/', then that file system can't be unmounted.
- It closes all open file descriptors inherited from its parent, although some file descriptors may need to be kept open for specific operations.

Since daemons lack a controlling terminal, they frequently use the syslog facility to log error messages and other information to a central location. These logs are then processed by the syslogd daemon and distributed based on the syslogd.conf configuration file.

Moreover, daemons should be designed to handle signals properly. The SIGTERM signal should trigger an orderly shutdown, while the SIGHUP signal allows the daemon to reinitialize itself by rereading its configuration files and reopening log files.

In summary, daemons are vital background processes that facilitate the automation of various tasks and services in computer systems. They run independently of user sessions and contribute to system stability and efficiency.

6 Software Tools and COTS

6.1 Tools

6.1.1 Draw.io

Draw.io is a popular web-based diagramming tool that offers a user-friendly interface for creating a variety of diagrams, from flowcharts to mind maps. It allows for real-time collaboration, customization of shapes and styles, and seamless integration with cloud storage services. As a free and open-source tool, draw.io is widely used to design this all of this project diagrams.

6.2 GitHub

GitHub is a widely used web-based platform for version control and collaborative software development. It provides a repository for storing code, making it accessible to developers for sharing, collaborating, and tracking changes. GitHub's features include issue tracking, pull requests, and wikis, facilitating efficient teamwork. It plays a crucial role in open-source development and is a hub for the project developers to easily publish, update and manage code.

6.2.1 Buildroot

Buildroot is an open-source utility that simplifies the process of building embedded Linux systems. It automates the compilation and configuration of the Linux kernel, root filesystem, and various cross-compilation toolchains. Buildroot is a popular choice for developers working on embedded and IoT devices, as it streamlines the creation of custom, lightweight Linux distributions tailored to specific hardware and application requirements. It offers a vast library of pre-configured packages and a flexible configuration system, making it a valuable tool for creating efficient and optimized embedded Linux systems.

6.2.2 Visual Studio Code

Visual Studio Code (VS Code) is a widely used, free, and open-source code editor developed by Microsoft. It's renowned for its lightweight design, fast performance, and extensive extensibility through a wide range of plugins. VS Code supports multiple programming languages and offers features like code debugging, Git integration, and IntelliSense for code completion.

6.2.3 FlutterFlow

FlutterFlow is a visual development platform that simplifies the process of building mobile applications, including Android apps, by providing a low-code, drag-and-drop interface for creating user interfaces and app logic. It offers an intuitive way to design and develop mobile apps without writing extensive code. One notable advantage of FlutterFlow is its easy integration with Firebase, a popular backend-as-a-service platform by Google. Firebase integration in FlutterFlow allows developers to effortlessly connect their Android apps to Firebase services such as Firebase Authentication, Realtime Database, Firestore, and Cloud Functions. This streamlined integration facilitates user authentication, real-time data synchronization, and serverless computing, making it easier for developers to build robust and feature-rich Android applications.

6.3 COTS

6.3.1 POSIX Threads

POSIX Threads (Pthreads) are a standard for multithreading in Unix-based operating systems, enabling the creation and management of multiple threads within a single process for concurrent execution of tasks. They provide a portable and efficient way to handle parallelism, enhancing program performance by leveraging multiple CPU cores.

6.3.2 Alsamixer

Alsamixer is a command-line audio mixer tool for Linux-based systems, commonly used for configuring and adjusting sound settings. It provides a textual interface to control various aspects of audio playback and recording, including volume levels and input/output devices. Alsamixer is particularly helpful for managing sound on systems with ALSA (Advanced Linux Sound Architecture) as the underlying sound system, allowing users to fine-tune audio parameters. It will be used both for the captured audio for the livestream and for the output audio to the speakers.

6.3.3 FFmpeg

FFmpeg is a powerful, open-source multimedia framework and command-line tool used for manipulating, converting, and streaming audio and video content. It is renowned for its versatility and supports a vast range of multimedia formats, codecs, and containers. FFmpeg enables tasks like video and audio transcoding, format conversion, video editing, and streaming. Developed under the GNU Lesser General Public License, FFmpeg is widely used across various platforms, including Linux, macOS, and Windows. It provides a comprehensive set of libraries and a robust command-line interface for both novice and advanced users. FFmpeg's modular architecture allows developers to integrate its libraries into their applications, making it a fundamental building block for multimedia software. It also supports hardware acceleration for specific tasks, enhancing performance for video encoding and decoding. Specifically, it will be used for video and audio capture and to stream it via RTMP techonology to the server.

6.3.4 AWS

Amazon Web Services (AWS) is a leading cloud computing platform that offers a vast array of on-demand computing, storage, and networking services. It provides scalability, reliability, and flexibility for businesses and developers to host applications and data, making it a cornerstone of cloud infrastructure worldwide. The integration of AWS two services AWS Elemental MediaLive and Amazon Simple Storage Service (Amazon S3) enables a seamless process for receiving RTMP (Real-Time Messaging Protocol) streams and displaying them on Amazon CloudFront, AWS's content delivery network. AWS Elemental MediaLive serves as the live video processing component, transcoding and packaging incoming RTMP streams for distribution. These streams are then stored in Amazon S3, which acts as a durable and scalable repository. Finally, Amazon CloudFront, with its global network of edge locations, ensures low-latency and high-performance delivery of the content, making it readily accessible to viewers across the globe. This implementation offers a robust solution for delivering real-time video content to the application where the user will be presented the live-streamed and stored videos.

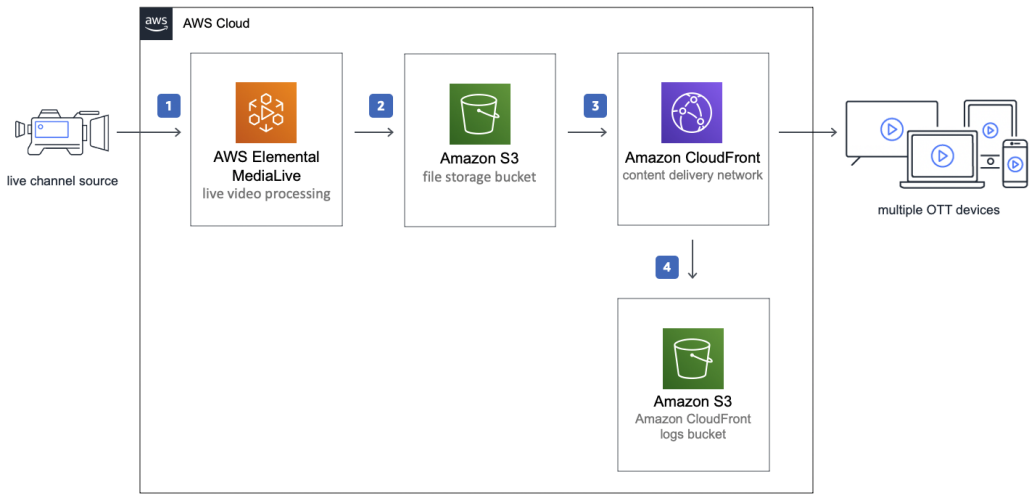


Figure 14: AWS Services Internal Design

6.3.5 RaspiCam API

The RaspiCam API, short for Raspberry Pi Camera Application Programming Interface, is a set of functions and tools designed to provide software developers with direct access to the Raspberry Pi Camera Module's capabilities. Developed by the Raspberry Pi Foundation, the RaspiCam API enables programmers to control and manipulate various aspects of the camera, including capturing still images and recording video. It offers extensive flexibility in configuring camera parameters like resolution, exposure, white balance, and more, making it a powerful tool for creating custom imaging applications on Raspberry Pi devices. With its straightforward C and C++ libraries, the RaspiCam API has become a popular choice for Raspberry Pi enthusiasts and developers looking to harness the full potential of the Raspberry Pi Camera Module in their projects, from security cameras to computer vision applications.

6.3.6 FireBase

Firebase is a comprehensive mobile and web application development platform by Google. It offers a wide range of services, including real-time database, authentication, cloud hosting, and analytics. Firebase simplifies the development process by providing developers with tools and infrastructure to create high-quality apps quickly. It is known for its ease of use, scalability, and real-time data synchronization, making it a popular choice for building modern, feature-rich applications. Firebase also provides robust security and cloud functions, facilitating seamless backend integration for app developers. This will be the main part for sending notification and log signals to the app from the house system.

7 Hardware Specification

In this subsection it will be displayed every hardware component this project will integrate, their specifications and connections will also be exemplified.

7.1 Raspberry pi 4 model B

The development board that will be used is the Raspberry pi 4 model B, as this board is a constraint of the project. The following picture shows the development board:

Figure 15: Raspberry pi 4 model B

Features:

- Processor:
  - Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memory:
  - 4GB ECC LPDDR4
- Connectivity:
  - 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless
  - LAN, Bluetooth 5.0, BLE

  - Gigabit Ethernet
  - 2 × USB 3.0 ports
  - 2 × USB 2.0 ports
- GPIO:
  - Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
- Video & Sound:
  - 2 × micro HDMI ports (up to 4Kp60 supported)
  - 2-lane MIPI DSI display port
  - 2-lane MIPI CSI camera port
  - 4-pole stereo audio and composite video port
- Multimedia:
  - H.265 (4Kp60 decode);
  - H.264 (1080p60 decode, 1080p30 encode);
  - OpenGL ES, 3.0 graphics
- SD card suport:
  - Micro SD card slot for loading operating system and data storage
- Input power:
  - 5V DC via USB-C connector (minimum 3A1)
  - 5V DC via GPIO header (minimum 3A1)
  - Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
- Environment:
  - Operating temperature 0–50°C

The next diagram shows the GPIO and other interfaces of the Raspberry pi 4, explained earlier and where they are located.

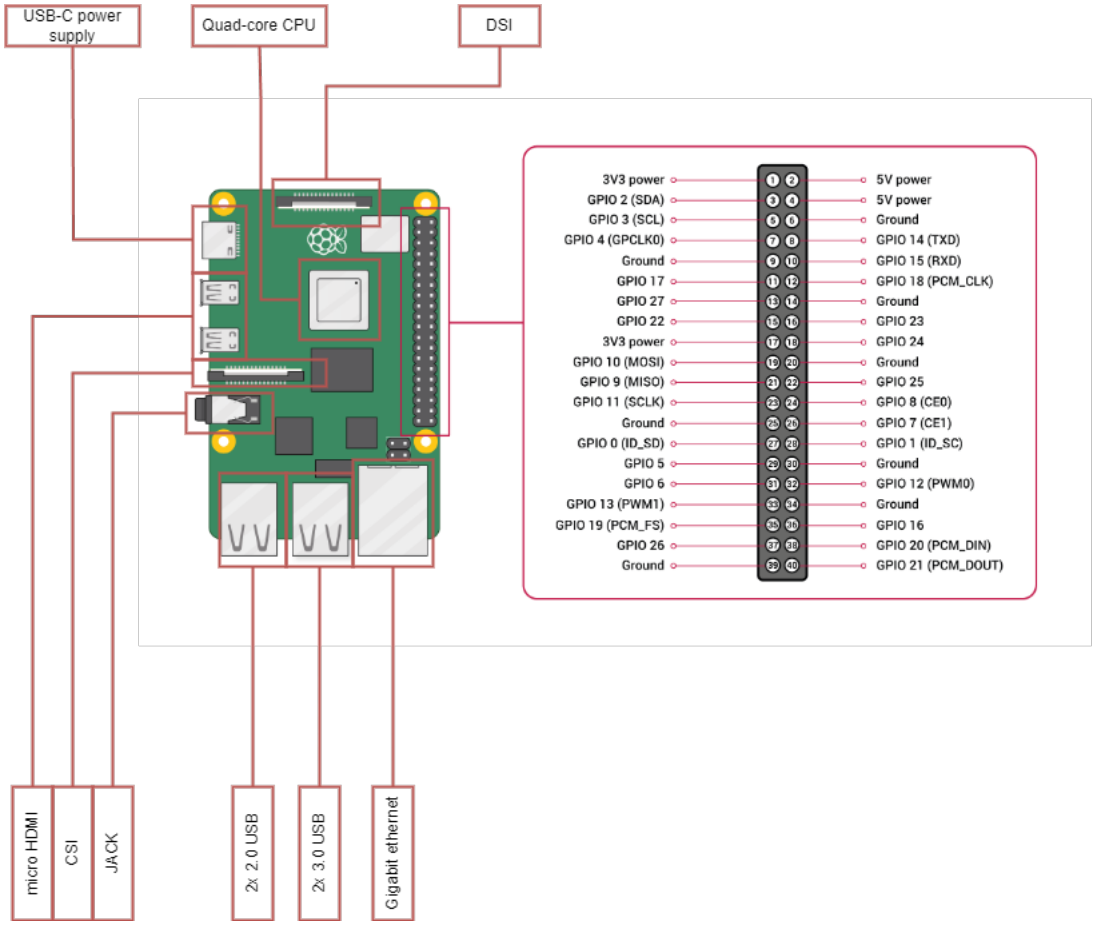


Figure 16: Diagram of Raspberry pi 4 interfaces and GPIO

7.2 Camera

One of the main functionalaties of this project its the ability to monitor the person in need, to enable that monitoring capabilities even with low light, so to meet this constraints the Raspberry Pi Night Vision Camera was chosen.

The Raspberry Pi Night Vision Camera is a key component of our project, designed to enable effective monitoring of individuals in low-light conditions. This versatile camera module is equipped with features to capture clear images and video in challenging lighting situations. Its notable features include upgradability for enhanced night vision capabilities, making it an ideal choice to meet our project's monitoring needs, even in the darkest environments.

Specifications:

- 5 megapixel OV5647 sensor

Figure 17: Camera with IR lights

- Focal Length : 3.6MM (adjustable)
- View Angle : 75.7 degree
- Sensor best resolution : 1080p

7.3 Motion Sensor

In our project, we've implemented an alternative approach to monitor individuals in a less intrusive manner by detecting movement in a room. To accomplish this, we rely on the Passive Infrared (PIR) sensor. This sensor is adept at identifying changes in infrared radiation within its field of view, making it an effective solution for motion detection. By using the PIR sensor, we can track movement within a room discreetly, without the need for direct physical contact or visual surveillance. This approach respects privacy and offers a non-intrusive means of monitoring, ensuring the well-being and safety of the individuals in question.

Figure 18: Motion Sensor

Specifications:

- Power Supply : 5V - 12V
- Range : 25cm - 10m
- view Angle : 360 degrees

7.4 Microphone

The chosen microphone is a vital component of our project as it allows us to capture audio from the person in care and transmit it to the user. This functionality is essential for remote monitoring, ensuring the safety and well-being of the individual. The microphone's flexible placement options also enable non-intrusive and optimal audio capture, making it a valuable addition to the project.

Figure 19: USB Microphone

Specifications:

- Power Supply: 5 V
- Sensitivity: -30 dB ± 3 dB
- Connection Port: USB 2.0
- Frequency Range : 20 Hz-16 KHz.

7.5 Push Button

The inclusion of this push button is paramount for our project, primarily due to its user-friendly design. Its larger size makes it easily noticeable and accessible, even for individuals with limited mobility or in stressful situations. The integrated illumination feature ensures visibility during nighttime or low-light conditions, enabling the person in need to locate and press it with ease in case of an emergency or when assistance is required. This user-centric design enhances the project's effectiveness in providing timely aid and support to those it serves.

Figure 20: Push Button

Specifications:

- Diameter: 10cm
- Incorporated LED Power Supply: 12V

7.6 Speaker

The integration of a speaker is a fundamental requirement for our project, as it plays a crucial role in facilitating communication between the caretaker and the person in need. This interactive element is essential for relaying important information, providing reassurance, or responding promptly to requests for assistance.

Figure 21: Mini Speaker

Specifications:

- Input Voltage: 2.2V-5V
- Impedance: 8ohm

7.7 Magnetic Sensor

The integration of a Magnetic Sensor into a project aimed at assisting individuals with Alzheimer's and dementia is of paramount importance. This technology plays a pivotal role in enhancing the safety, security, and overall quality of life for those living with these cognitive conditions. For individuals with Alzheimer's and dementia, maintaining a safe environment can be challenging. These conditions can lead to memory loss, disorientation, and the tendency to wander, which can result in potentially dangerous situations. Open doors, in particular, pose a significant risk, as they can allow these individuals to wander into unfamiliar or unsafe areas. The Magnetic Sensor, in this context, serves as a guardian, providing real-time monitoring of doors. When integrated into the project, it offers a simple yet effective solution to determine if a door is open or closed. This functionality can trigger an alert system, notifying caregivers, family members, or healthcare providers when a door is left open longer than it should be.

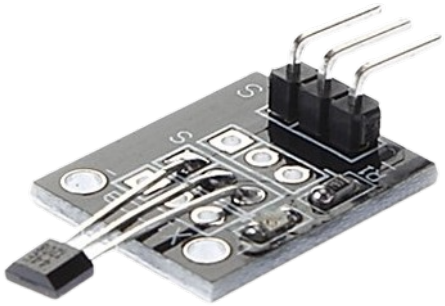


Figure 22: Magnetic Sensor

Specifications:

- Input Voltage: 2.2V-5V
- Impedance: 8ohm

Specifications:

- Input Voltage: 5V

7.8 Controlable Relay

The integration of a Controllable Relay into a caregiving system is an essential safety measure, particularly when caring for individuals who may be vulnerable, such as those with cognitive impairments or conditions that affect their judgment. This technology provides a means to remotely and securely control the operation of appliances like ovens and stoves, mitigating the risk of accidents and harm. For individuals in care who may struggle with memory loss, confusion, or other cognitive challenges, the misuse of kitchen appliances, especially ovens and stoves, can pose a significant danger. Cooking appliances left unattended or improperly operated can lead to fires, burns, and other hazards. The Controllable Relay offers a robust solution to this problem.

Figure 23: Relay

Specifications:

- Input Voltage: 5V
- Output Voltage: 230V

7.9 Hardware Connections

The following figure 24 shows the connections of this project sensors and actuators, the camera connects to CSI (Camera Serial Interface), the microphone to the USB port (its not the same microphone that will be used), button to 3v3 and GPIO4, the PIR sensor to 5v, GND and GPIO16 and finally the speaker to the audio jack.

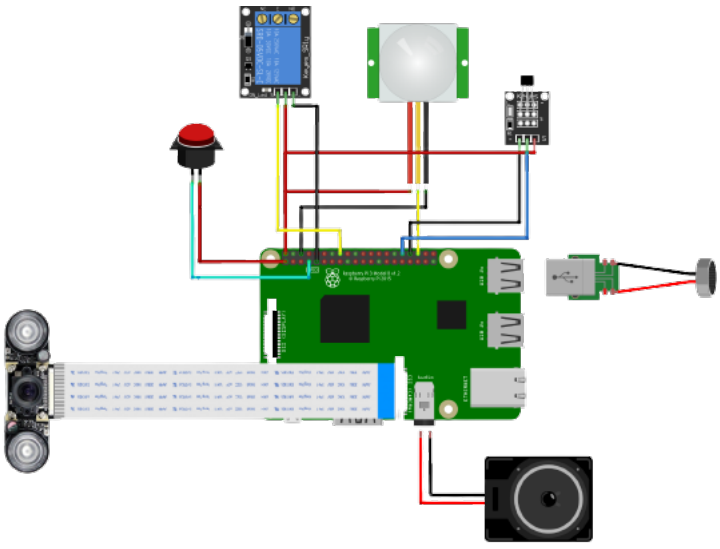


Figure 24: Hardware connections

8 Software Specification

Software specification, in the context of software development, refers to the process of defining in detail what a software system should do, how it should do it, and what the expected outcomes and behaviors are. It plays a vital role in the software development life-cycle, providing a clear and unambiguous description of a software project's requirements and functionality.

8.1 Task Division

This project is constituted by three major parts:

- **House System & Daemon Processes :** This parts are responsible to read and process the data received from the sensor, the House_system is responsible to process and send the inputs from camera and microphone process it and send it to the livestream, Raspberry also needs to receive data and audio from the database to then enable the bidirectional communication. The Daemon processes are responsible for the background processes like the motion detection, the pressing of the button and updating these sensors flags to the database to then notify the user.
- **Database & Livestream :** Database is responsible to save all the data from the sensors, audio from the user to the person in care and flags to notify the user. The livestream enables the live video and audio from the house where the person in care is and its also responsible to save and record the live video.
- **Mobile App :** The Mobile App its the interface to the user/caretaker, enables access to the livestream, sensors logs, recorded videos and display notifications. The design of the app must be simple and easy to use, to facilitate the monitoring and communication between the caretaker and person in care.

The next figure demonstrates a simple explanation of the main processes of every system and their connections and interactions with each other:

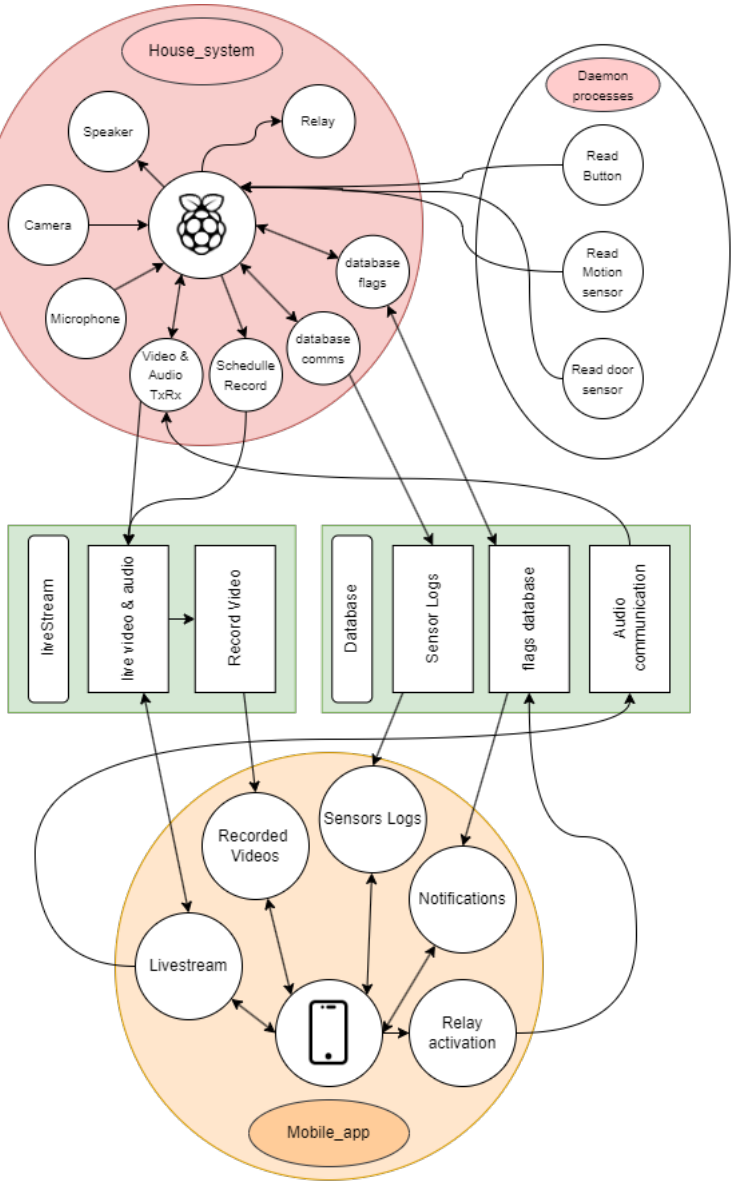


Figure 25: Tasks Overview

The system comprises three main components: the Local System, Linking Infrastructure, and the Mobile App. The Local System includes the House System and Daemon Processes, acting as the system's core. It manages sensor data and device control within the house, ensuring seamless operation.

The Linking Infrastructure connects the Local System to the broader network, utilizing AWS for livestreaming and recording and a Database for data storage. The Mobile App serves as the primary user interface, offering real-time data access and control over sensors

and controllers.

These components work in tandem to provide a robust system. The House System and Daemon Processes feed data to AWS and the Database for real-time monitoring and data storage. The Mobile App offers an intuitive interface for users to interact with the system, enabling control and access to data. This integrated system ensures efficient data flow and a user-centric experience.

8.2 Local System

8.2.1 Class Diagrams

A UML (Unified Modeling Language) class diagram is a graphical representation used in software engineering to visualize the structure and relationships of classes within a software system. It is a fundamental type of UML diagram and plays a crucial role in designing, planning, and documenting software applications. Class diagrams provide insights into the static structure of a system by depicting classes, their attributes, methods, and how they relate to one another.

CHouse_System

Figure 26 introduces a pivotal class at the heart of the local system, playing a crucial role in the system's core functionality. This central class is primarily responsible for the instantiation of vital class objects essential to the system's operation. These key components include:

- **Clivestream_ctrl** : Central to establishing bidirectional communication and video data storage, this class empowers live streaming capabilities and efficient video data management.
- **Cdatabase_sys** : Serving as a hub for data exchange, this class is pivotal in receiving and transmitting notifications and sensor data, ensuring the smooth flow of information across the system.

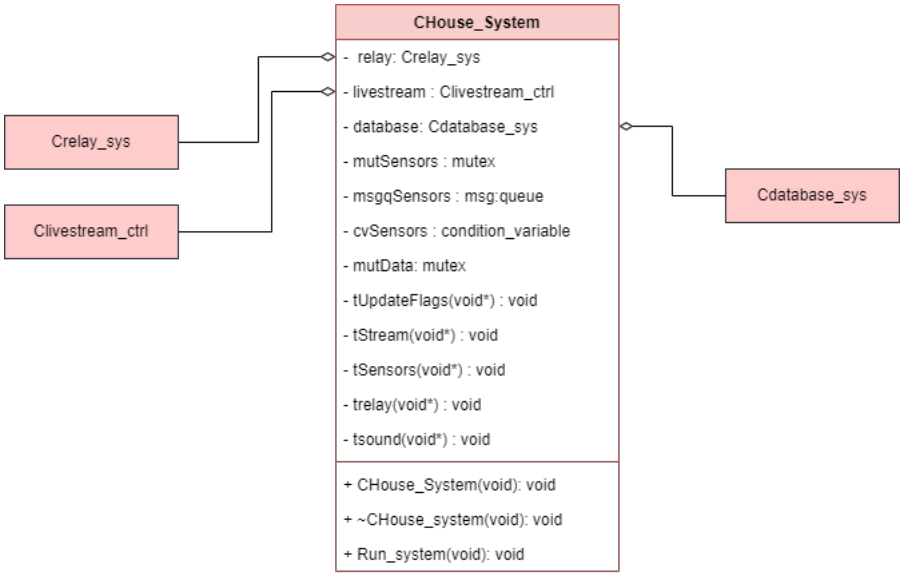


Figure 26: Class diagram: CHouse_system

CDsensors

Figure 27 presents the daemon class diagram within the Local system. This class plays a central role in initializing and coordinating several critical components, which are as follows:

- **Cmotion_sys** : This class not only controls the motion sensor but also oversees motion-related data, enabling the system to detect and respond to changes in the environment.
- **Cbutton_sys** : The class dedicated to managing the communication button ensures that users can effortlessly interact with the system. It provides the means for users to communicate their preferences or needs, serving as a vital input interface.
- **Cdoor_sys** : Responsible for the magnetic door sensor, this class is pivotal for security and access control.

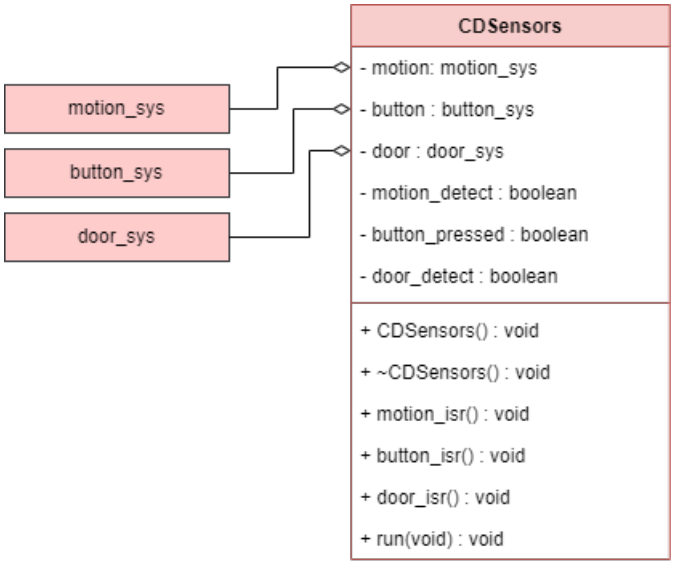


Figure 27: Class diagram: CDsensors

Class Clivestream_sys

The class diagram of the figure 77 shows the class Clivestream_sys. This class implements the following functions to control the livestream: start_livestream() and stop_livestream(), like the name says the functionality of the first is to start the livestream and the second stops the livestream. This class also possesses the function play_audio(), this function as the name states plays the audio from the user in the app to the person in care inside the house.

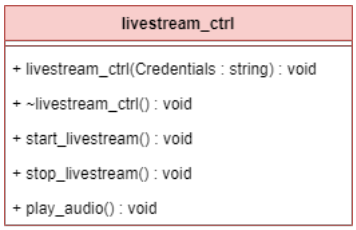


Figure 28: Class diagram: Clivestream_sys

Class Cmotion_sys

The class diagram of the figure 29 shows the class Cmotion_sys. This class creates a motion sensor object and implements the following functions to control: enable() and disable(). The first function enables the external interrupt that will act when the motion sensor detects movement and the other will disable the interrupt.

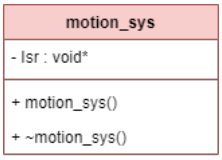


Figure 29: Class diagram: Cmotion_sys

Class Cbutton_sys

The class diagram of the figure 29 shows the class Cbutton_sys. This class creates a button object and implements the following functions to control: enable() and disable(). The first function enables the external interrupt that will act when the button is pressed and the other will disable the interrupt.

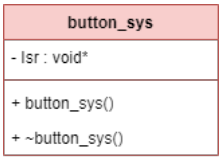


Figure 30: Class diagram: Cbutton_sys

Class Cdoor_sys

The class diagram of the following figure 31 portrays the class Cdoor_sys which is responsible for the creation of the door sensor object and implements its constructor and destructor.

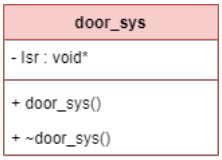


Figure 31: Class diagram: Cdoor_sys

Class Crelay_sys

The class diagram of the following figure 32 portrays the class Crelay_sys which is responsible for the creation of the relay object and implements the class constructers and destructores. This class also contains two functions activate_relay() and deactivate_relay()

the first one its used to activate the relay and interrupt the flow of electricity and the other functions restores it.

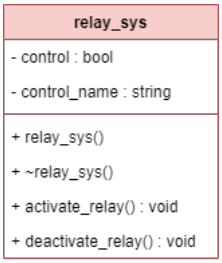


Figure 32: Class diagram: Crelay_sys

8.2.2 Task Overview

The main process is constitued by five tasks:

- **tStream** : this task is responsible to start and stop the live video and audio streaming.
- **tSensors** : this task receives messages sent by the daemon regarding the sensors values, the task is awaken when the signal SIGUSR1 sent from the daemon when exists new message to read.
- **tUpdateFlags** : the main purpose of this task is to synchronize the local flags with their counterparts in the database, if their value differ from the one in the house system their value is updated.
- **tRelay** : this task is responsible for the activation and deactivation of the relay interrupter when the user changes its flag on the database and subsequently triggers the relay.
- **tSound** : this thread is responsible for the creation of child process that will reproduce the audio received from the user.

The daemon,CDSensors, possesses tasks to read the value from the motion sensor, door sensor and button, like was stated above in the tSensors task, when any of the following tasks sends a command to the main process by message queue a signal, SIGUSR1, is used to alert the main process that a new message was sent.

- **motion_isr** : interrupt for the motion sensor. Always enable and executes when movement is detected, sending a command to the main process.

- **button_isr** : interrupt for when the button is pressed. Always enabled and executes when the button is pressed, sending a command to the main process.
- **door_isr** : interrupt for door sensor. When the door is opened the connection is broken and a signal is sent and an interrupt activated and processed to send to the main program.

8.2.3 Task Priority

The following diagram, shows the priority of the main process tasks. As was stated earlier the main process is composed by four tasks where the ones with higher level of priority are *tSensors* and *tUpdateFlags* then *tRelay* and finally with lowest priority the task *tLiveStream* and *tSound*.

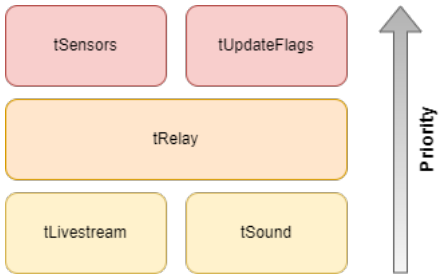


Figure 33: Task Priority

8.2.4 Initial Processes

Initial processes are responsible for launching and initializing the core functionality of this project. These processes are crucial for setting up the environment, loading necessary resources, and ensuring that the product is ready to perform its intended tasks.

Main Process

The Main Process serves as the linchpin for the program's seamless operation. It initiates the system's core functionality by creating the essential class objects within the CHouse_system constructor. These class objects, such as Cmotion_sys, Cbutton_sys, and Cdoor_sys, are fundamental for specific tasks within the system.

Once the CHouse_system and associated objects are initialized, the program is set in motion with the invocation of the "run()" command. In this phase, all the tasks, which are also created within the CHouse_system constructor, are enabled and started. This is achieved by utilizing the ".join()" approach, where each task is executed.

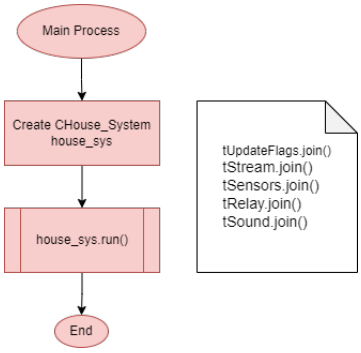


Figure 34: Main Process

The run() function is described alongside the chart in a note to better understand the work of the main process and which functions are initialized.

Daemon

Another important process that needs to be initialized in the start of the application is CDsensors which is explained in the figure 35. First the daemon is created(create_daemon()), then the message queue is opened to enable the communication with the main process. When receives a message that message will be the main process PID, after receiving the message a sensors object is created and the run() function is executed.

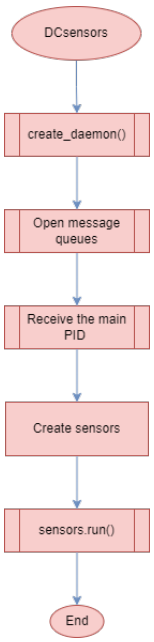


Figure 35: Daemon Process

8.2.5 Flowcharts

A flowchart is a visual representation of a process or system using symbols and lines to illustrate the sequence of steps, decisions, and actions within that process. Flowcharts are a valuable tool for visually mapping out complex processes or algorithms, making them easier to understand, analyze, and communicate.

CHouse_system

The following figure 36 represents CHouse_system class which is the main class and is responsible to create all the objects that will be need, all the synchronization tools needed and the creation of the tasks: tUpdateFlags, tStream, tSensors, tSound and tRelay.

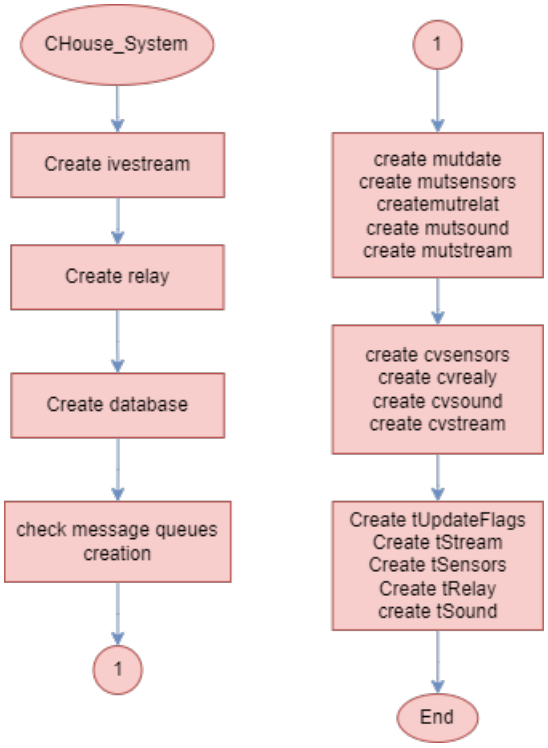


Figure 36: CHouse_system Flowchart

tUpdateFlags

The task responsible to update the flags in the database is represented in the figure 37. The task when the SIGUSR1 is detected checks if the flags stored in the database differ from the ones in the local system they are updated in the database otherwise the task does nothing.

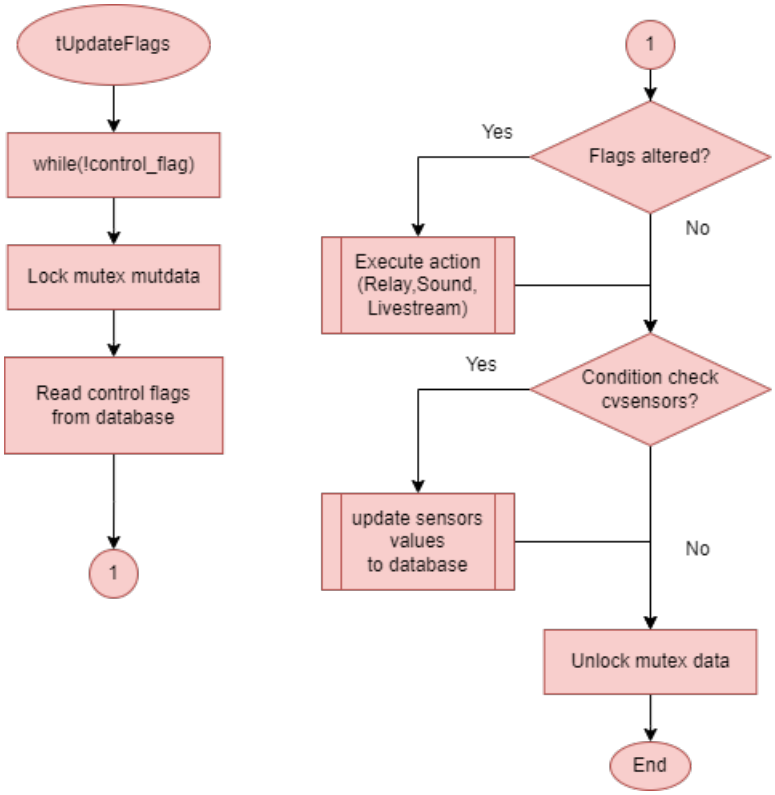


Figure 37: task tUpdateFlags flowchart

tStream

The task responsible for the live audio and video streaming is represented in the following figure 38. The livestream will be available 24/7 to monitor the person in need, the stream starts the same time as the task is created, if there is a record flag the live video starts being recorded until this flag stops, the stream only stops streaming if a command is sent to stop streaming or if any error is detected in the livestream and a flag is flagged.

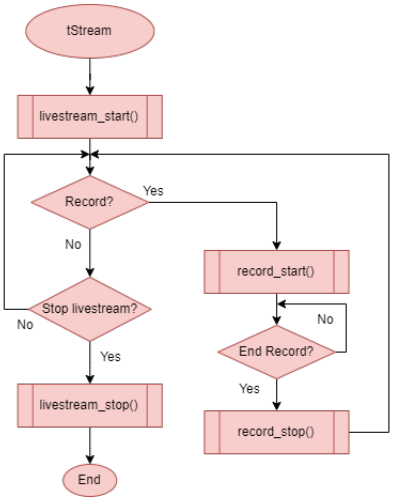


Figure 38: task tStream flowchart

tSensors

This task is responsible for the controlling and receiving messages from the daemon sensors, this task is represented in following figure 39. When no message is in the message queue the task goes to sleep and when cvSensor is notified the task awakens. This happens if SIGUSR1 is activated, after the condition variable is triggered the message is read from the message queue, if this message corresponds to the movement detection it triggers the flag that controls the status of the PIR sensor or if the button is pressed the same happens but the flag that controls the button status is the one that is triggered, or if is the door sensor the one which is triggered the door flag changes. On both this cases the database flags must be updated so there is a need to intertwine both this tasks and if the button is pressed there is also a need to send a notification to the caretaker.

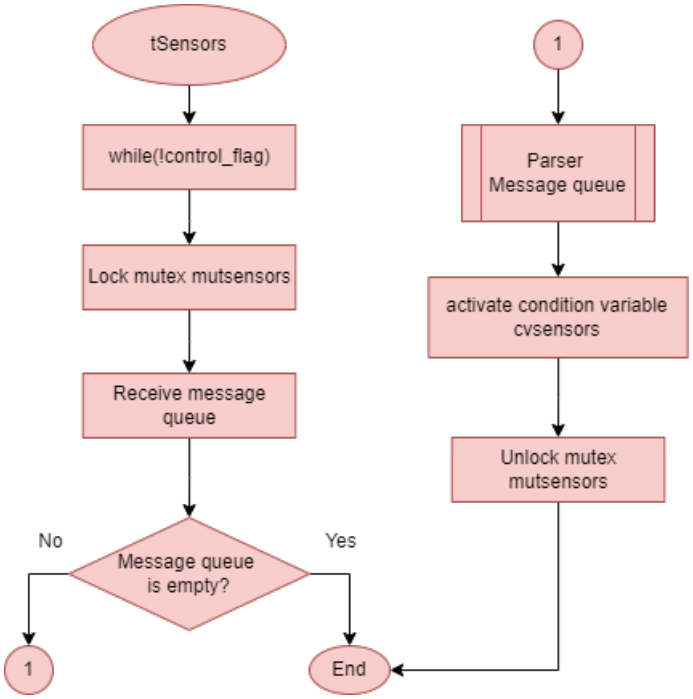


Figure 39: task tSensors flowchart

tRelay

The relay task, as illustrated in Figure 40, assumes a critical role in ensuring the precise control of the relay system. This task continually monitors the relay flag activation and patiently awaits any changes in its status. The flag's state is dynamically updated through interactions with the database, primarily in response to a user's request to either activate or deactivate the relay system.

In essence, this task acts as the bridge between the user's intent and the relay system's operation. By consistently examining the relay flag and promptly responding to changes, it guarantees that the relay system behaves in accordance with the user's commands, contributing to the overall responsiveness and functionality of the system.

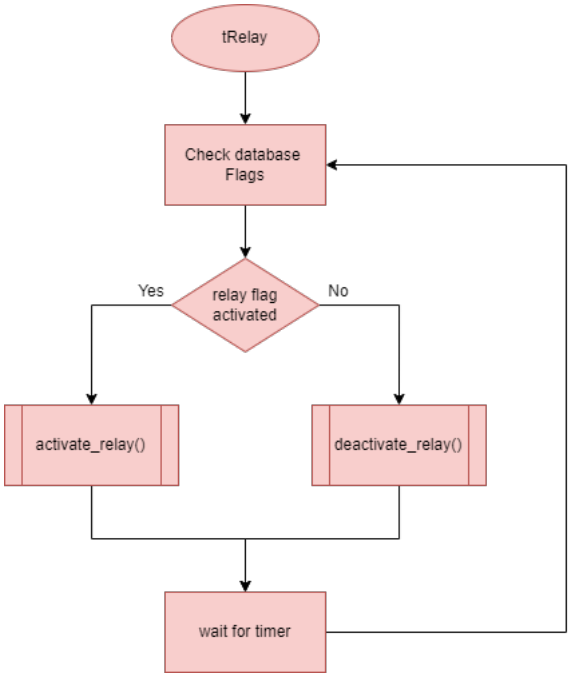


Figure 40: task tRelay flowchart

tSound

The final thread implemented in this project is the tSound, referenced in Figure 41. The primary objective of this thread is to play the sound stored in the database, allowing the user to communicate verbally with the person in care. This sound playback functionality enhances the bidirectional communication aspect of the system, providing a crucial means for remote interaction between the user and the individual under care.

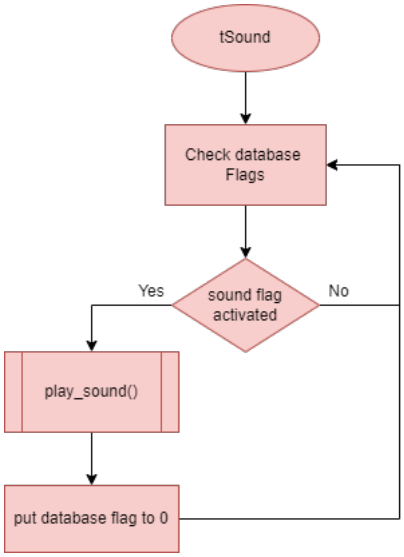


Figure 41: task tSound flowchart

DSensors methods

In Figure 37, the motion_isr method is showcased, and it is activated when motion is detected. This method serves the purpose of relaying information to the main process. Additionally, in Figure 37 also introduces the button_isr and door_isr method, both this methods behave the same way as the motion method but the first one notifies if the button is pressed and the other notifies if the door is opened. All this process notify the main system.

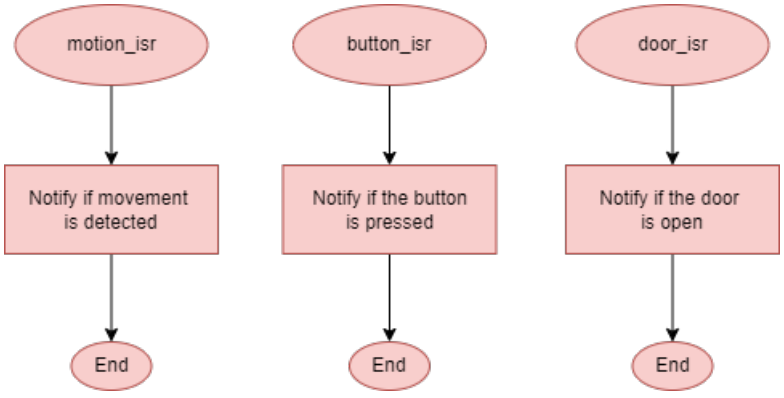


Figure 42: Flowcharts of the sensors interrupts

button_sys methods

The following picture 43 shows both the constructor, on the left, and the destuctor, on the right of the button_sys class. The constructor is responsible to insert the correct device driver to control the button and enable the interrupt to detect when this button is pressed. The destructor makes the opposite from the constructor, disables the previous enabled interrupt and removes the device driver.

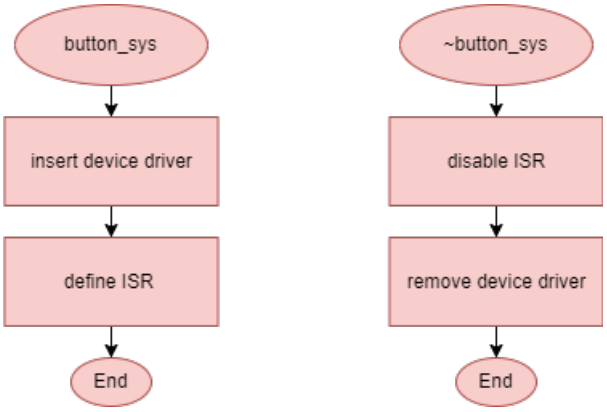


Figure 43: Flowcharts of both Constructor and Destructor of button_sys class

door_sys methods

The following picture 44 shows both the constructor, on the left, and the destuctor, on the right of the door_sys class. The constructor is responsible to insert the correct device driver to control the door sensor and enable the interrupt to trigger when the door is detected. The destructor makes the opposite from the constructor, disables the previous enabled interrupt and removes the device driver.

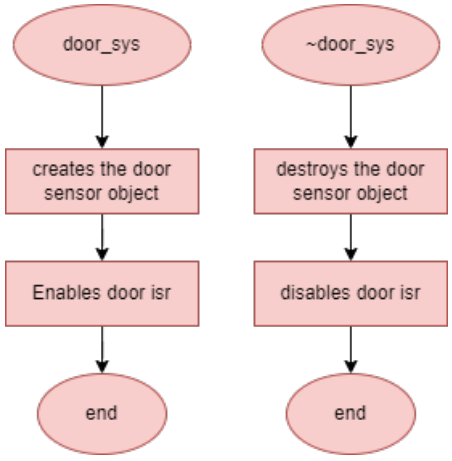


Figure 44: Flowcharts of both Constructor and Destructor of door_sys class

motion_sys methods

The following picture 45 shows both the constructor, on the left, and the destuctor, on the right of the motion_sys class. The constructor is responsible to insert the correct device driver to control the motion sensor and enable the interrupt to trigger when movement is detected. The destructor makes the opposite from the constructor, disables the previous enabled interrupt and removes the device driver.

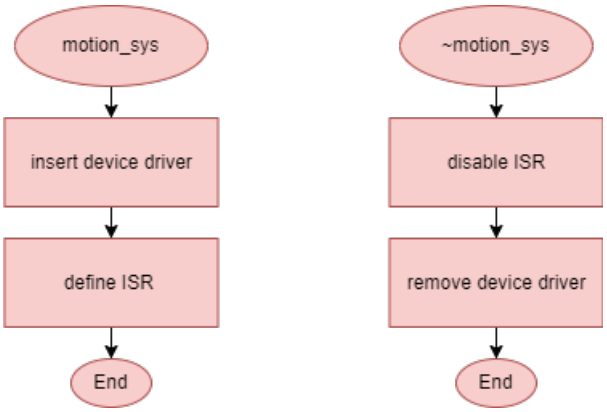


Figure 45: Flowcharts of both Constructor and Destructor of motion_sys class

livestream_ctrl methods

The picture below 46 displays the flowcharts that compose the livestream_ctrl class, the constructor(), start() and stop(). The first is responsible to correctly start the AWS (Amazon Web Services) server, where the livestream will be running, the start function calls the ffmpeg to create video and audio files to then send to the server via RTMP(Real Time Messaging Protocol), the stop function purpose is the opposite from the previous, stops the ffmpeg and the livestream of audio and video.

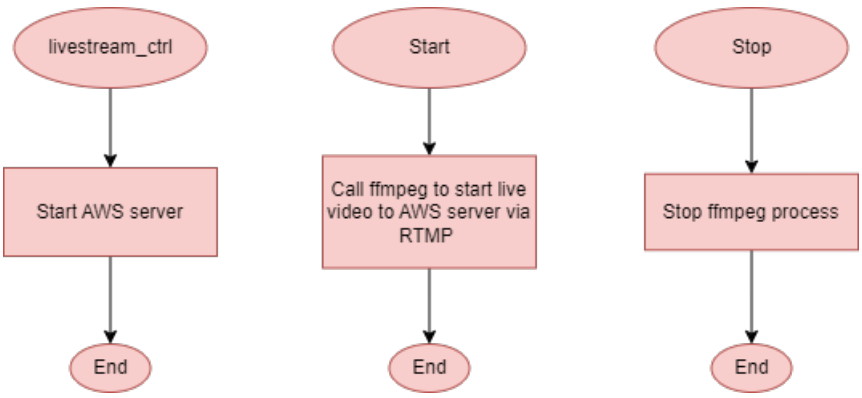


Figure 46: Flowcharts of Constructor , Start and Stop functions

The following picture 47 illustrates the flowchart of the function play_audio(). This function plays a pivotal role in managing the audio playback process within the system, contributing to the seamless communication functionality between the user and the person in care.

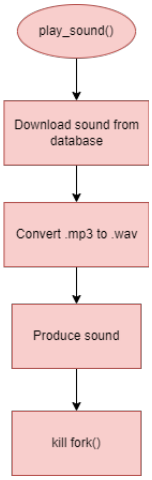


Figure 47: Flowcharts of Constructor , Start and Stop functions

database_sys methods

The picture below 48 displays the flowcharts that compose the database_sys class. This class is composed by a constructor, a send_data and receive_data functions. The constructor is responsible to establish communication with the database and get the correct urls for each folder, the send_data functions ensures the connection to the database and sends the data from the controller to the specified location in the database, the receive_data function also needs to ensure the connection to the database and then sends data from the database to the controller.

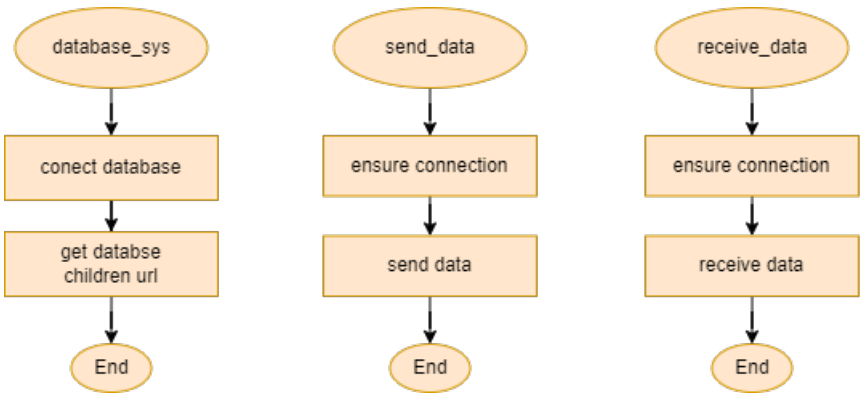


Figure 48: Flowcharts of the database_sys Constructor, send_data and receive_data

relay_sys methods

The picture below 49 displays the flowcharts of the functions that compose the relay_sys class. This class is made by a constructor, destructor, activate_relay and deactivate_relay function, the two last functions are responsible for the activation and deactivation of the relay when the user wishes. the constructor and destructor create and eliminate the relay object.

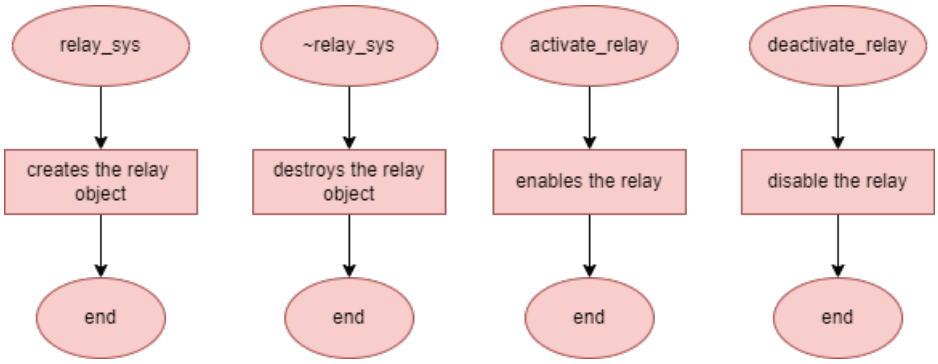


Figure 49: Flowcharts of the functions from the class relay_sys

8.3 Remote System

The remote system is composed by 2 major components the mobile app and the database.

8.3.1 Database

Entity Relationship

An Entity Relationship Diagram or ERD is a visual map that illustrates how different data elements are connected in a database, making it easier to design, communicate, and manage data effectively. The next diagram shows this project database diagram.

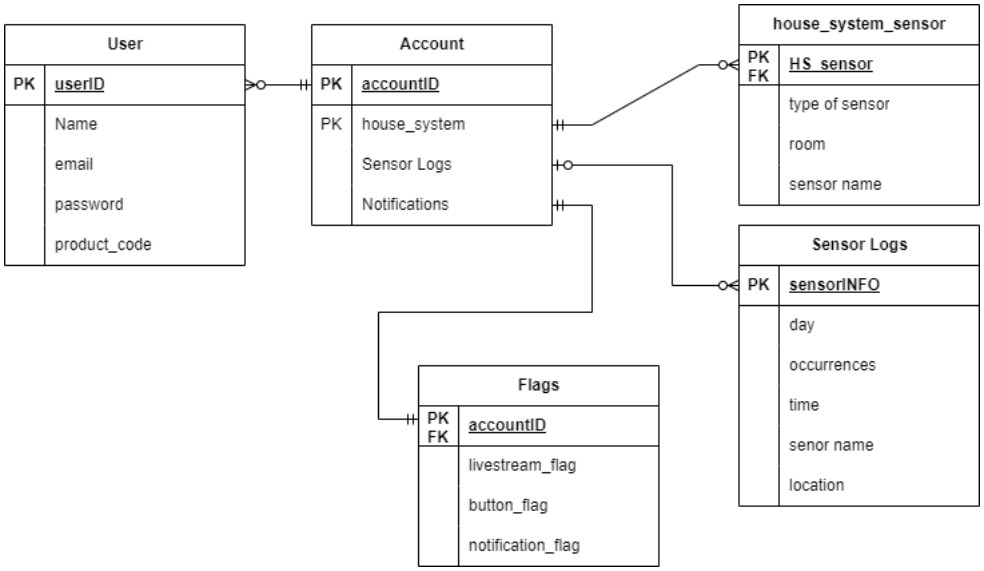


Figure 50: Entity Relationship Diagram

As it can be seen in the diagram this database its composed of 5 entities: User, Account, Flags, Sensor Logs and House System Sensor. This database can have multiple users but the user can only access one account. This account saves every user , saves also Sensors logs and house sensors registers. The database also saves the flags that enable and/or notify the local or remote systems.

8.3.2 Mobile App Functionalities

The table below offers a high-level overview of the app's functionalities. It distills complex functions into a simplified format for easy reference and understanding. This abstraction enables users to quickly grasp the app's core capabilities without delving into intricate details.

App Functionalities
Play livestream
Play recorded video
Register account
Login into account
Logout of account
Activate Relay
Verify Sensors
Notify the caretaker in case of unexpected events

8.3.3 Mobile Application Flowcharts

In this flowchart, the user needs to login or register. The login is verified in the database and the register is verified in order to not double log-in. After the register, the user goes back to log-in. After the log-in is successful, the user is show the Home Page.

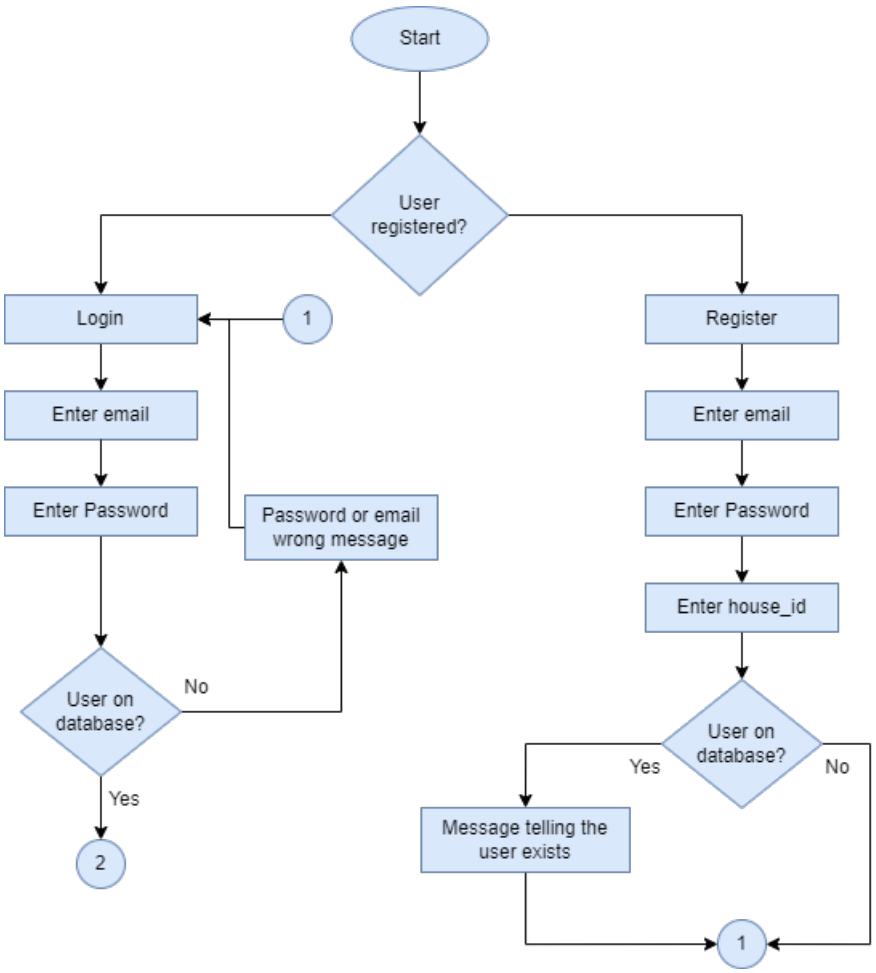


Figure 51: Flowchart Login

In the homepage, the user can access the functionalities. The functionalities are Livestream, Recorded Videos, Sensor Logs, Configuration and Logout. At any point, while recording the audio, there is the option to stop sending it. If the audio is successfully recorded it is sent to the database, otherwise it is discarded. Recorded Videos option displays the recorded videos when pressed. Sensor Logs display the sensor logs with the time stamps and showing which was the activated sensor. Logout, send the user back to the Login Menu.

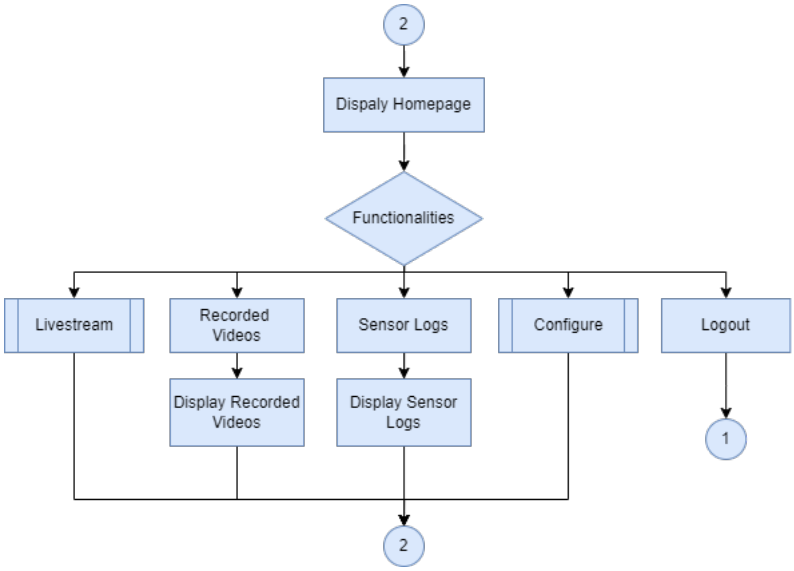


Figure 52: Flowchart Homepage

Livestream enables to see the Livestream in real time and allows the option to send audio. Additionally, users have the convenience of easily returning to the home screen or main menu, ensuring a seamless and user-friendly navigation experience.

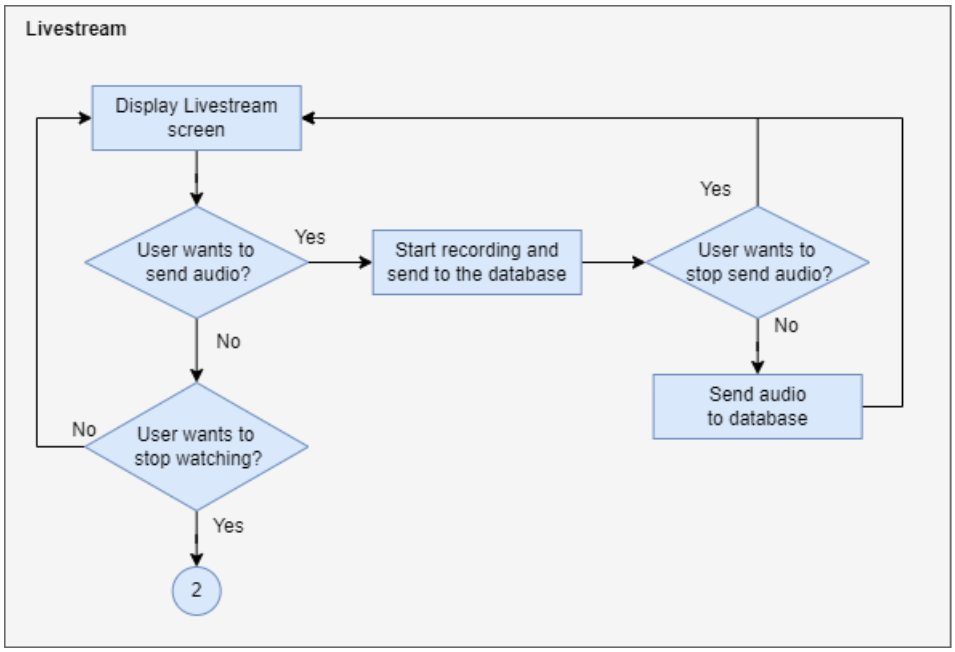


Figure 53: Flowchart Livestream page

Configuration enables to set the timers of the recording, to add a new user, to configure sensors and to change credentials. Configuring sensor allows to enable or disable the sensors, changing credentials allows to change user password and add user allows to create a new user to an already configured device.

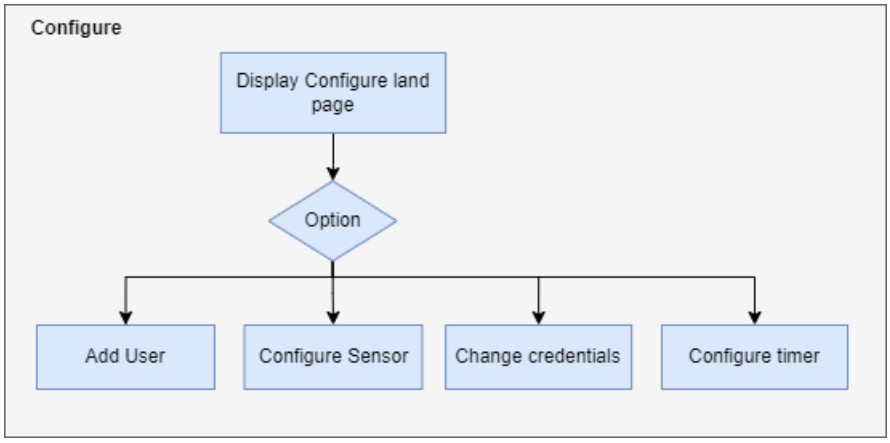


Figure 54: Flowchart Configure page

8.3.4 Mobile App Design

The mobile application serves as the primary interface connecting the caretaker and the person in need, representing the heart of our project. A well-crafted design is paramount to ensure a seamless and enriching experience for users. The chosen design strikes a balance between simplicity and modernity, offering a user-friendly interface packed with robust functionalities. In the following section, we'll present images showcasing this thoughtfully designed app, which harmonizes ease of use with a wealth of capabilities, enhancing the overall quality of care and communication within our project.

The picture 55 shows the login page on the left and the register page on the right, in the first the user can put is email and its password after checking the user credentials and if they are correct the user goes to the Homepage (picture 56), if the credentials are incorrect a message is presented saying that and allows to access the register page by pressing the register word down on the app. The register page allows a new user to register the only difference from a normal register is the necessity to define the house ID, which is a code presented and given to every House system to give another layer of security.

Figure 55: Login and Register Page

The picture below (picture 56) shows the Livestream Page and Home Page which is responsible to allow the caretaker to access functions to monitor and help the the person in care. The user can access the notifications, sensor logs, livestream page and configurations page.

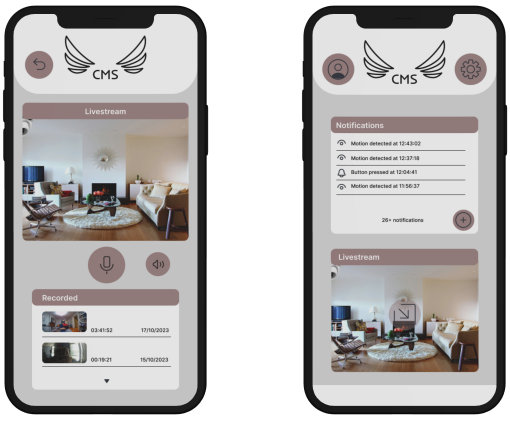


Figure 56: Livestream and Home Page

9 Test Plan

9.1 Hardware Test Cases

The following table represents the test cases for all the hardware that will be used as well as the expected result. In further phases of this project, there will be the addition of the real result.

Test Cases	Expected Result	Real Result
	Motion Sensor	
Movement in the sensor	SIGNAL pin high (5V)	----------
No Movement in the sensor	SIGNAL pin low (0V)	----------
	Button	
Press button	Output pin high (5V)	----------
	Microphone	
Record audio	Audio recorded equals to the one inputted	----------
	Speaker	
Play audio	Audio played correctly	----------
	Camera	
Capture image	Frames Captured correctly	----------
Light mode switching	Changes back and forth to night mode automatically	----------
	Magnetic Sensor	
Magnet detected	Output pin high (5V)	----------
	Relay	
Relay activated	Output voltage (230V)	----------

Table 4: Hardware test cases.

9.2 Software Test Cases

This comprehensive table presents a range of software test cases, each describing an action or operation that the system should perform. The "Expected Result" column outlines the desired outcomes, while the "Real Result" column is intended for recording the actual results once the testing process is conducted. These tests are crucial for ensuring the system's functionality, reliability, and performance in both the local and remote contexts.

Test Case	Expected Result	Real Result
	Local System	
Send message to queue	Message written correctly	---------
Read message from queue	Message read correctly	---------
Send notification	Database flag updated	---------
Process sensor trigger	Distinguish which sensor was triggered	---------
Process video and audio	Correctly put video and audio data together	---------
Live stream	Send video and audio to server successfully	---------
	Remote System	
Play livestream	Successfully retrieve livestream from server	---------
Play recorded video	Successfully retrieve recorded video from database	---------
Register account	Update database with new account	---------
Login into account	Retrieve login credentials from database and compare them	---------
Logout of account	Logout valid	---------
Enable/Disable Relay	Update Relay Flag on database	---------

Table 5: Software test cases.

9.3 Integration Test Cases

The following table provides an overview of various integration test cases and their expected results in a system. These tests are crucial for ensuring that the system functions as intended when different components are integrated. The "Expected Result" column specifies what should happen during each test, and the "Real Result" column, which is currently left empty, is where the actual outcomes of the tests will be recorded during the testing process.

Test Cases	Expected Result	Real Result
Press Button	Notify the Caretaker User	------
Motion Detected	Notify the Caretaker User	------
Door Open	Notify the Caretaker User	------
Control Relay	Enable/Disable Relay	------
Register Account Sucessfully	Show Successful Message	------
Configure Sensor	Enable/Disable each sensor notification	------
Sensor Logs	Display sensor logs	------
Click on "Live Stream"	Display Live Stream	------
Click on "Mic Button"	Stream Audio; Activate Speaker	------
Click on "Recorded Video"	Display Recorded Video	------

Table 6: Integration Test Cases.

9.4 Dry Run

In the following table, a dry run of the system's core is presented. This dry run involves a comprehensive testing of the main sensors that form an integral part of the system. Additionally, there is a configuration test designed to let the user remotely enable or disable each sensor notification. This dry run is a crucial step in the system's development or validation process, as it helps identify any potential issues or discrepancies in the system's behavior. To better acknowledge the tests, check the flowcharts on figure 37, 39 and figure 54.

Line	Motion Detection	Button	SMotion Enabled	SButton Enabled	MsgqSensors	Signal	Send to Database	Notification
1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	1	0	0	0	0	0
4	0	0	1	1	0	0	0	0
5	0	1	0	0	1	1	1	0
6	0	1	0	1	1	1	1	1
7	0	1	1	0	1	1	1	0
8	0	1	1	1	1	1	1	1
9	1	0	0	0	1	1	1	0
10	1	0	0	1	1	1	1	1
11	1	0	1	0	1	1	1	1
12	1	0	1	1	1	1	1	1
13	1	1	0	0	1	1	1	0
14	1	1	0	1	1	1	1	1
15	1	1	1	0	1	1	1	1
16	1	1	1	1	1	1	1	1

Table 7: Core Dry Run.

10 Gantt Diagram

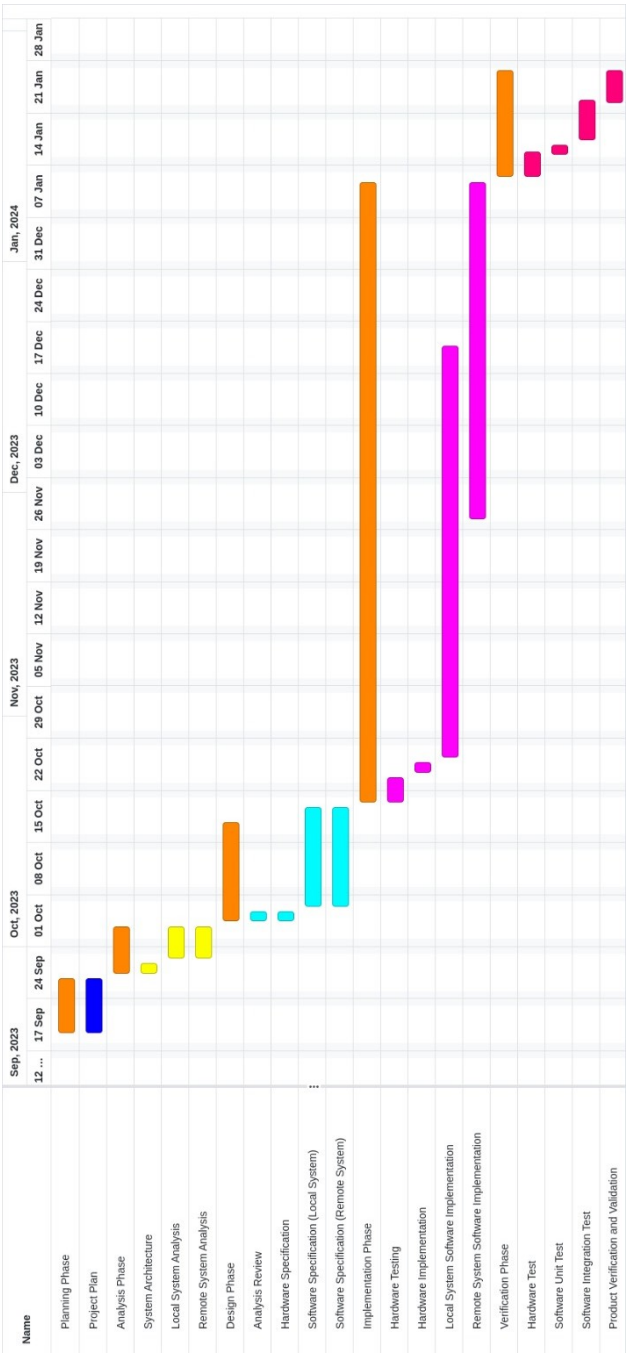


Figure 57: Gantt Diagram

11 Implementation

The implementation phase is a crucial step where we turn our project ideas into a practical solution. Using an embedded Linux OS created through Buildroot, we're integrating sensors, a button, and a relay to demonstrate the capabilities and responsiveness of this system. The goal is to write code and create a product that ensures these components work together seamlessly, allowing effective monitoring and interaction in the monitored environment.

A key feature is the real-time communication through a caregiver app. This app enables remote observation and engagement with those under care. Testing will ensure the system works well, with a focus on sensor accuracy, interaction responsiveness, livestream reliability and bidirectional communication.

11.1 Buildroot

Buildroot serves as a valuable build system for developers working on embedded Linux projects, streamlining the creation of a functional embedded Linux system with minimal effort. Essentially, it is a collection of makefiles that automate the entire process of acquiring, configuring, compiling, and installing various software packages necessary for the embedded system. This tool efficiently manages dependencies and addresses cross-compiling challenges for a wide range of platforms, making the development process more accessible.

A distinctive feature of Buildroot is its ability to automatically generate the essential cross-compilation toolchain, construct a root filesystem, compile the Linux kernel image, and produce a bootloader tailored for the specific embedded system. This comprehensive functionality simplifies the traditionally intricate tasks involved in setting up an embedded Linux environment, making the development process more efficient and less error-prone.

```
$ mkdir buildroot
$ cd buildroot
$ wget https://buildroot.org/downloads/buildroot2021.08.tar.gz
$ tar -xzf buildroot-2021.08.tar.gz
$ cd buildroot-2021.08
```

The previous listings explained how to download and install Buildroot. The next will portray how to do the base configurations that will be needed for the future configurations.

```
$ make raspberrypi4_defconfig
$ make menuconfig
$ make
```

The 3 commands shown before create the basic configurations for the board that will be used, in this case the Raspberry Pi4 as this board is a constraint of this project, after 'make menuconfig' to enable necessary configurations for the project and finally 'make' to create the image that will be passed to the Raspberry.

Toolchain Configuration

The Buildroot toolchain simplifies the process of cross-compiling software for embedded platforms. It includes essential components like a compiler, assembler, linker, C library (libc), header files, and debugging tools. By automating configuration within the Buildroot framework, the toolchain optimizes the compiled software for specific embedded platforms. Developers can compile code on a more powerful host machine, ensuring compatibility with the target embedded system.

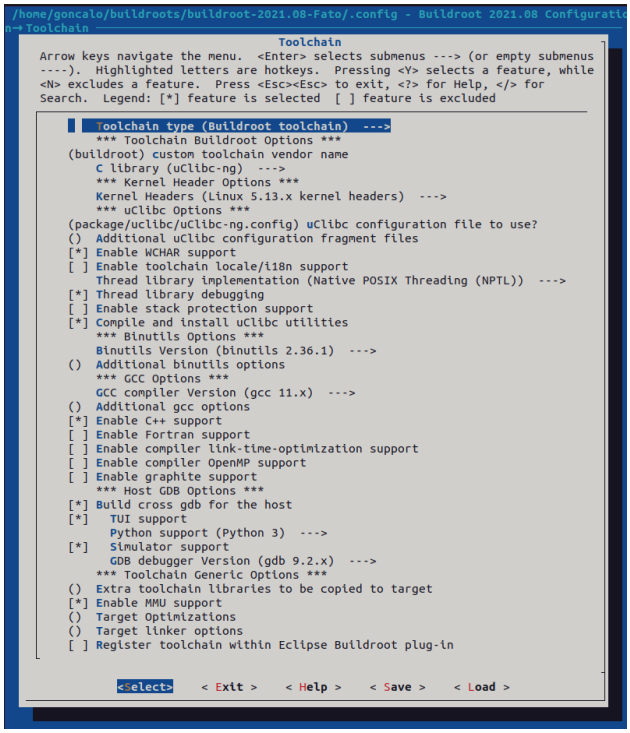


Figure 58: Buildroot Toolchain Menu

The earlier image illustrates the project's Buildroot toolchain, highlighting the presence of uClibc-ng—a compact alternative to the glibc C library, tailored for efficiency. The image also indicates the inclusion of debugging configurations and the incorporation of Python 3 support.

System Configuration Configuration

The "System Configuration" is a critical phase in the Buildroot build system where settings are tailored to create a specific embedded Linux system. During this phase, key decisions are made regarding the system's architecture, toolchain, package selection, filesystem details, kernel options, and other essential parameters. These configurations act as a blueprint for the Buildroot build process, defining the characteristics of the final embedded system. The detailed choices made in this section significantly influence the outcome, determining the overall functionality and components of the embedded system.

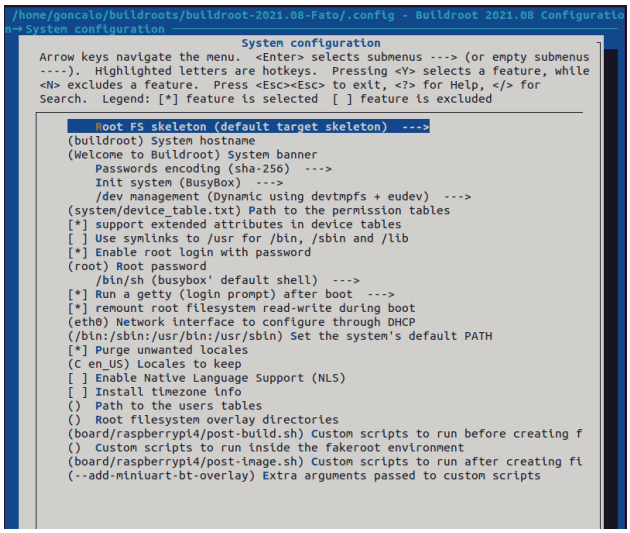


Figure 59: Buildroot System Configuration Menu

The depicted System Configuration of this project emphasizes the "/dev management" section, where the selection of "Dynamic using devtmpfs + eudev" was made. This choice is motivated by the advantages of "devtmpfs," which dynamically establishes the "/dev" directory during boot, allowing the kernel to autonomously create and manage device nodes. Additionally, "eudev" serves as a dynamic device management daemon that seamlessly complements "devtmpfs," enabling the system to adapt to changes by dynamically handling devices as they are connected or disconnected. This configuration was opted for its efficiency and adaptability in scenarios where the device configuration may vary.

Filesystem Images Configuration

The Buildroot Filesystem Images is responsible to encapsulate the entire software environment for the desired target device. The images created contain necessary components, binaries, libraries and other critical configurations to run the embedded Linux system.

The following images shows this project Filesystem Image configuration.

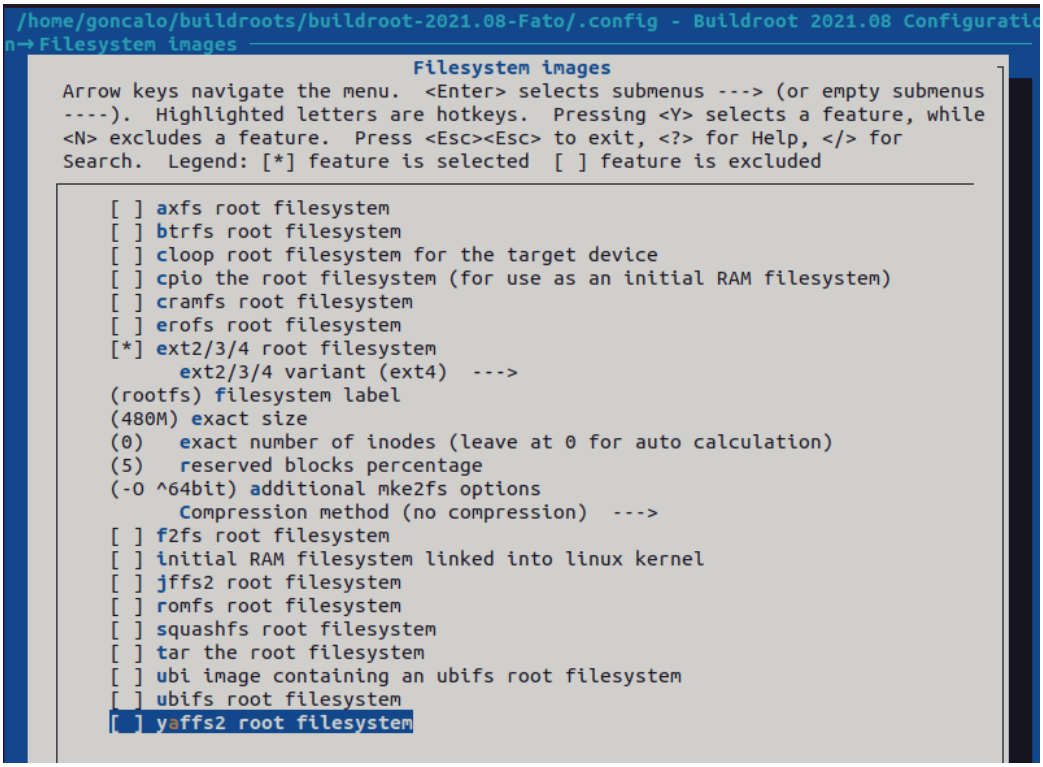


Figure 60: Buildroot Filesystem Image Menu

In the Figure 60 needs to be highlighted the need to expand the root filesystem to 480MB to accommodate the necessary configurations.

11.2 Buildroot Configuration

To accommodate the specific requirements of this project, adjustments were made within the Buildroot configurations. These alterations encompassed various settings crucial for building the embedded Linux system according to the project's specifications.

Under Target Packages some alterations were needed to be made:

11.2.1 Audio and video applications

alsa-utils

The ALSA Utilities package contains various utilities which are useful for controlling your sound card.

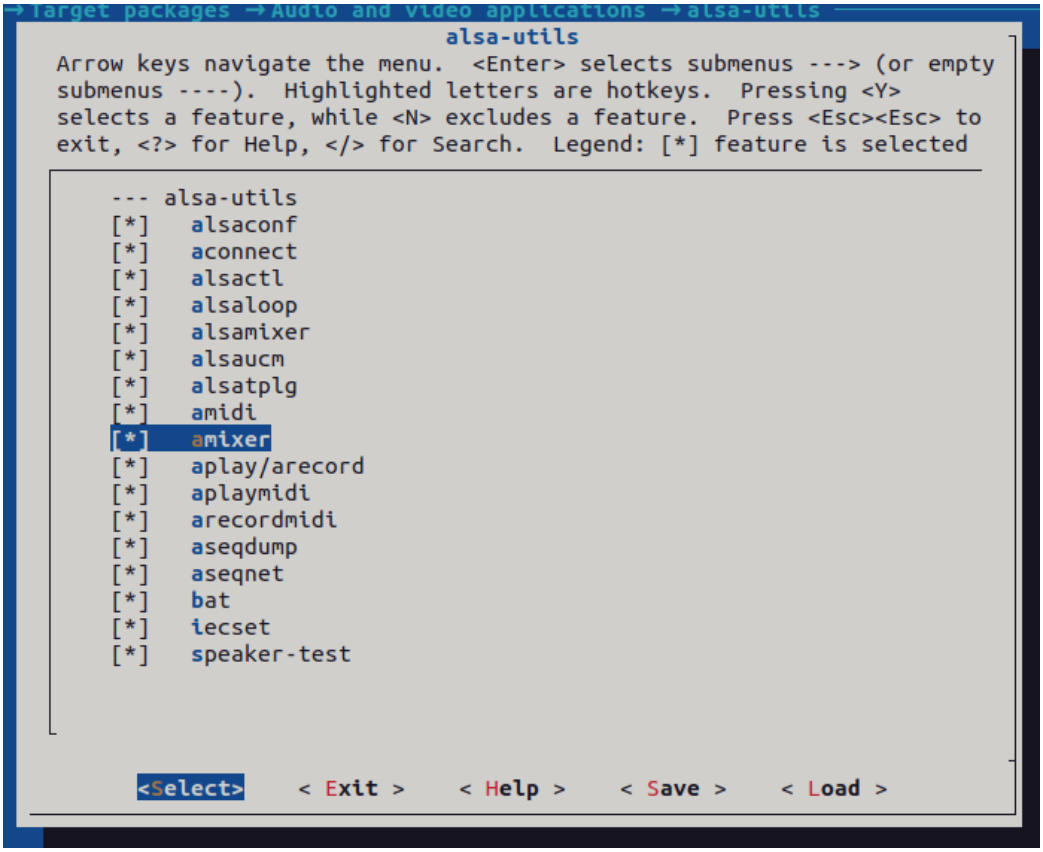


Figure 61: Alsa-utils Configuration

The figure referenced as 61 illustrates the configurations undertaken to enable the recognition and utilization of the sound device driver along with its capabilities. These adjustments in settings were crucial to ensure the proper integration and functionality of the audio subsystem within the embedded system.

ffmpeg

FFmpeg serves as the primary multimedia framework for this project, possessing the remarkable ability to handle decoding, encoding, transcoding, multiplexing, demultiplexing, streaming, filtering, and playback of diverse multimedia content. This robust framework is instrumental in managing various audio and video formats, making it a pivotal component for processing and manipulating multimedia data within this project.

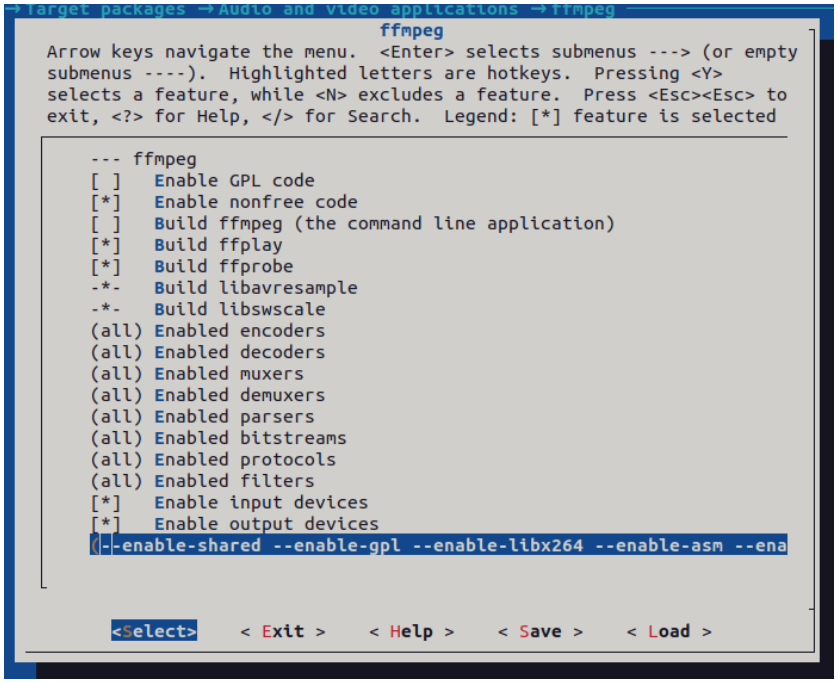


Figure 62: ffmpeg Configuration

To correctly configure the ffmpeg some enables were manually added, in the following configuration: "–enable-shared –enable-gpl –enable-libx264 –enable-asm –enable-inline-asm –enable-alsa –enable-librtmp –enable-libopenh264".

Inside the Audio and video applications other configurations were signaled:

- **v4l2grab** : Utility for grabbing JPEGs form V4L2 devices. This tool is similar to v4l2grab available from libv4l contrib directory, but provides additional features such as JPEG output.
- **v4l2loopback** : This module allows you to create "virtual video devices". Normal (v4l2) applications will read these devices as if they were ordinary video devices, but the video will not be read from e.g. a capture card but instead it is generated by another application.

11.2.2 Debugging, profiling and benchmark

Within this menu, the addition of "gdb" introduces in-system debugging capabilities or cross-debugging functionalities between the Raspberry Pi and the computer. This inclusion facilitates efficient debugging processes, allowing developers to analyze and troubleshoot code seamlessly within the embedded system or across the Raspberry Pi and the connected computer.

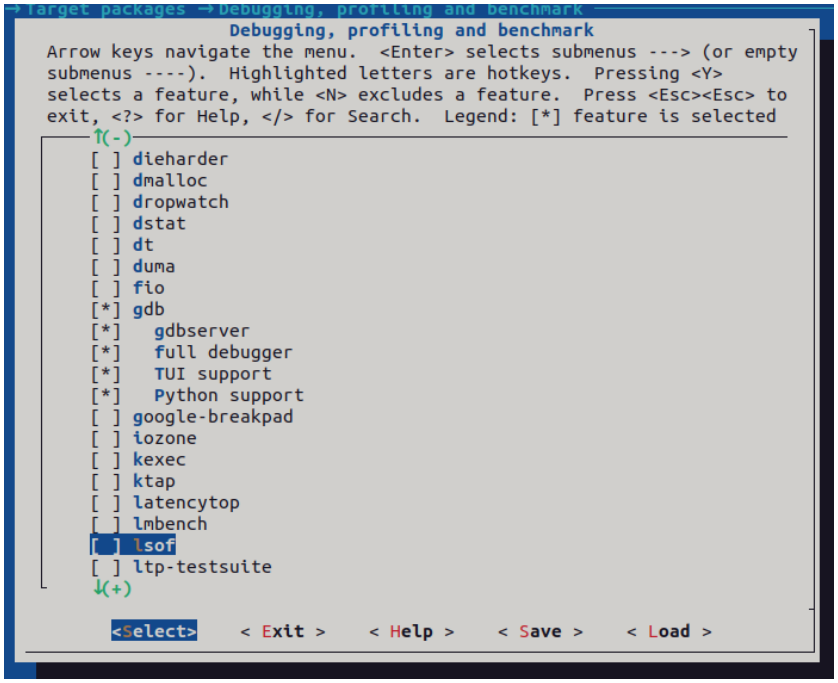


Figure 63: gdb Configuration

11.2.3 Hardware handling

In managing the essential hardware components, specific options were introduced to address these requirements:

- **eudev**
  - **enable hwdb installation** : Enables hardware database installation to /etc/udev/hwdb.bin
- **rpi-userland** : Raspberry Pi Userland contains the necessary library to use the VideoCore driver. Includes source for the ARM side code to interface to: EGL, mmal, GLESv2, vcos, openmaxil, vchiq_arm, bcm_host, WFC, OpenVG.
- **usbutils** : USB enumeration utilities.

Firmware

In this section, three crucial additions were made to address hardware-related functionalities shown in the following figure:

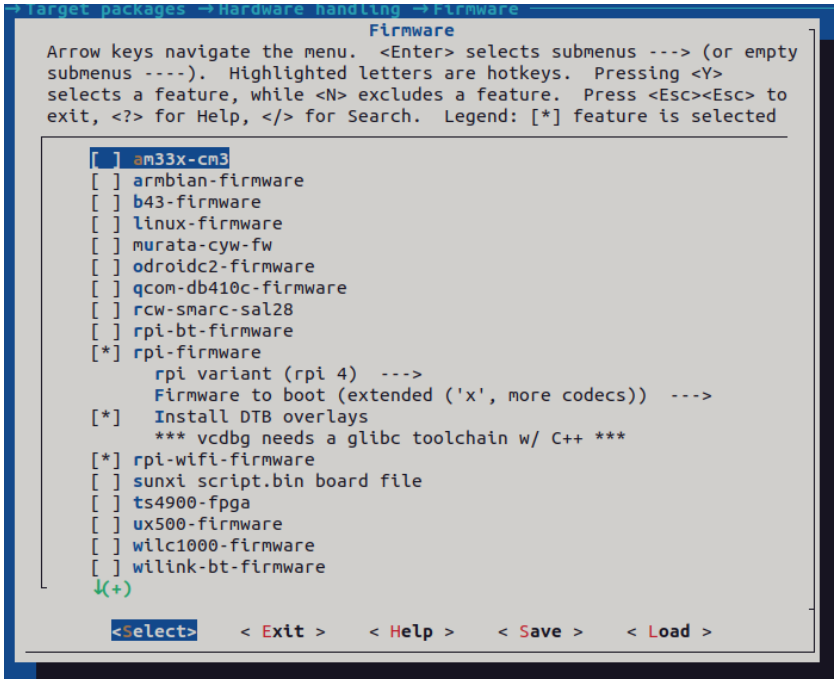


Figure 64: Firmware Configuration

- **rpi-firmware**: This addition involves the inclusion of "rpi-firmware," which encompasses firmware components specific to the Raspberry Pi. These firmware files are essential for enabling and optimizing various hardware features on the Raspberry Pi platform.
- **Install DTB Overlays**: The inclusion of "Install DTB overlays" pertains to Device Tree Binary (DTB) overlays. These overlays allow dynamic customization of the device tree, enabling the configuration of hardware peripherals and interfaces based on specific project requirements.
- **rpi-wifi-firmware**: The introduction of "rpi-wifi-firmware" signifies the inclusion of firmware necessary for handling wireless networking functionalities on the Raspberry Pi. This addition ensures proper wireless communication support, contributing to the comprehensive hardware handling capabilities of the system.

11.2.4 Interpreter languages and scripting

In the context of this project, a crucial requirement involves communicating and transferring data to a Firebase database. To fulfill this need, the project configuration necessitated the addition of a Python interpreter, augmented with extra core functionalities and external Python modules.

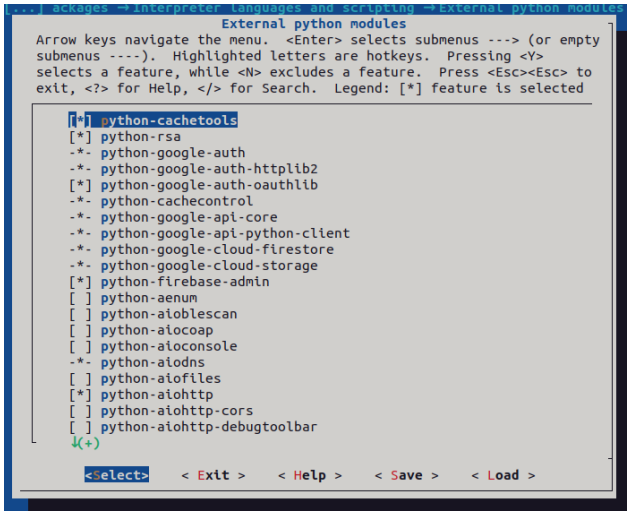


Figure 65: Python external modules Configuration

The presented figure 65 illustrates the configuration settings for external Python modules within the project implementation. Each of these modules was externally added, accompanied by corresponding .mk and Config.in files.

11.2.5 Networking applications

In order to seamlessly integrate live video streaming and enable wireless connectivity within this project, strategic configurations within the networking applications become imperative. This section delves into the specific networking applications that play a pivotal role in shaping the project's networking landscape, ensuring it aligns with the requirements of robust data transmission and wireless communication.

- **dhcp (ISC)** : DHCP relay agent from the ISC DHCP distribution.
- **dhcpcd** ; An RFC2131 compliant DHCP client
- **dropbear** : A small SSH 2 server designed for small memory environments.
- **ifupdown** : A small SSH 2 server designed for small memory environments.
- **ifupdown scripts** : Set of scripts used by ifupdown (either the standalone one, or the busybox one) to bring network up, or tear it down.
- **iptables** : Linux kernel firewall, NAT, and packet mangling tools.
- **iwd** : iNet Wireless daemon (iwd).
- **nginx** : nginx is an HTTP and reverse proxy server, as well as a mail proxy server.

- **ntp** : Network Time Protocol suite/programs.
- **wget** : Network utility to retrieve files from http, https and ftp.
- **wireless tools** : A collection of tools to configure wireless lan cards.
- **wpa-supplicant** : WPA supplicant for secure wireless networks.
- **wpan-tools** : Userspace tools for Linux IEEE 802.15.4 stack.

These added packages form the backbone of network communication in this project, emphasizing both the stream capabilities and the mechanism for establishing Wi-Fi connectivity. The subsequent images provide a visual representation of the construction and configuration of these packages, offering insights into their integration within the project's framework. Together this images showcases the project's capacity for seamless live video streaming and efficient Wi-Fi connectivity.

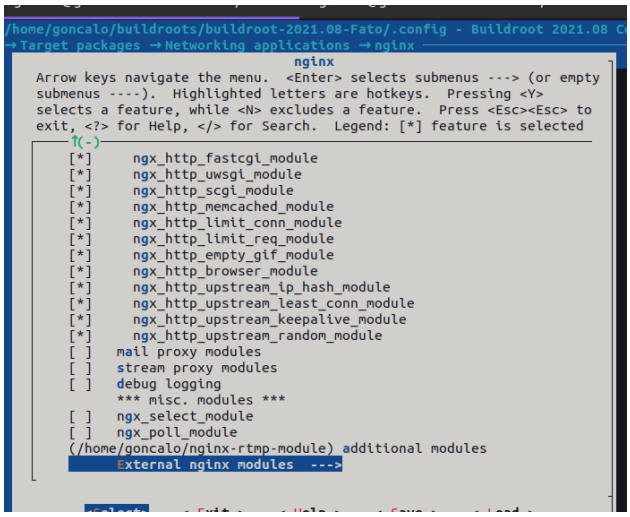


Figure 66: Nginx Configuration

Figure 66 illustrates the configuration of the Nginx module, a critical component responsible for facilitating RTMP communication in this project, enabling the execution of live streaming. Notably, it's essential to highlight the imperative need for adding an external RTMP module to Nginx. This addition is crucial, considering that the standard Buildroot configuration lacks this option. The external RTMP module augments the functionality of Nginx, ensuring robust support for RTMP communication and, consequently, the seamless execution of live streaming within the project.

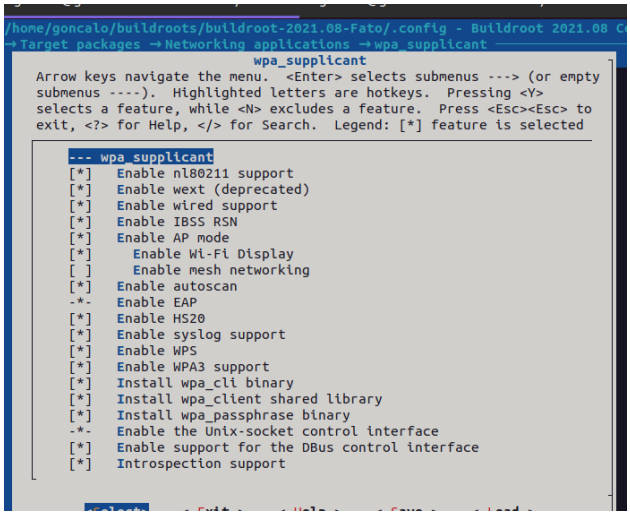


Figure 67: Wpa-supplicant Configuration

In Figure 67, the depicted options delineate the essential configurations required to establish a Wi-Fi connection within the project.

11.3 System Initialization

To optimize the boot process and ensure a seamless initialization within the project, several configurations were implemented. A pivotal aspect involved the creation of a new shell file, orchestrating the execution of a predefined set of commands. This file streamlines the execution of various tasks during boot, contributing to the efficient startup of the system.

Moreover, a tailored wpa_supplicant.conf file was introduced to facilitate the configuration of Wi-Fi settings. This file encapsulates essential parameters, including the SSID and password, streamlining the process of establishing a reliable and secure Wi-Fi connection within the project.

In addition to these configurations, specific attention was given to Nginx, a critical component for RTMP communication in the project. The nginx.conf file, which holds the configurations for Nginx, was meticulously adjusted to align with the project's requirements.

11.3.1 Init.d

To enhance the initialization process of the main program during system boot, a custom shell script named S90localinit.sh has been meticulously crafted. Shell init scripts serve the crucial role of executing automatically as part of the system's startup routine, orchestrating the essential conditions required for the seamless launch of the primary program.

The provided script is situated within the init.d directory, a conventional repository for system initialization scripts in Unix-like operating systems. This directory is dedicated

to scripts that govern critical system processes and services during the boot sequence. Notably, scripts prefixed with "Snn," such as S90cms.sh, are designed to initiate specific system services during startup, with the assigned two-digit number determining the order of execution.

```
#install device drivers
insmod /CMS/ddriver/button.ko
insmod /CMS/ddriver/relay.ko
insmod /CMS/ddriver/magnetic.ko
insmod /CMS/ddriver/motion.ko

#start wpa_supplicant
wpa_supplicant -i wlan0 -c /etc/wpa_supplicant.conf

sleep 15

#define priorities to wifi connection
sudo ip route del default via 192.168.140.216 dev wlan0
sudo ip route add default via 192.168.140.216 dev wlan0 metric
    100

sleep 5

#start main process
/CMS/mainlocal.elf

#start daemon
/CMS/daemon.elf
```

Listing 1: Custom shell script S90localinit.sh

This script encompasses critical commands such as loading device drivers, initiating the wpa_supplicant for wireless connectivity, defining WiFi connection priorities, and launching the main process and daemon. Each command plays a pivotal role in configuring the system for optimal performance during startup.

To save this command as shell sript the following command was need **chmod +x S90localinit.sh**, this command makes the shell script executable by changing its permissions.

11.3.2 CMakeLists.txt

In this project, the 'CMakeLists.txt' file assumes a pivotal role as a configuration blueprint within the CMake build system for our C++ development endeavors. This file acts as a comprehensive guide, dictating the intricacies of the project's build process, including the compilation and linking stages. By encapsulating critical instructions such as the definition of source files, desired C++ standards, and compiler options, it allows us to generate

platform-specific build scripts. This versatility is invaluable, ensuring cross-platform compatibility and enabling our development team to seamlessly manage and scale the project's build settings. The following listing demonstrate this project CmakeLists:

```
cmake_minimum_required(VERSION 3.22.1)
project(CMS_proj LANGUAGES CXX)

# Compilers
set(CMAKE_CXX_COMPILER "/home/goncalo/buildroots/buildroot
    -2021.08-Fato/output/host/bin/arm-buildroot-linux-
    uclibcgnueabihf-g++")
set(CMAKE_CC_COMPILER "/home/goncalo/buildroots/buildroot
    -2021.08-Fato/output/host/bin/arm-buildroot-linux-
    uclibcgnueabihf-gcc")

# Bin files to bin dir
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}/
    bin")

include_directories(
${CMAKE_CURRENT_SOURCE_DIR}/main/inc/home/goncalo/buildroots/
    buildroot-2021.08-Fato/output/host/arm-buildroot-linux-
    uclibcgnueabihf/sysroot/usr/include/python3.9
)

set(SOURCES
main/main.cpp
main/src/house_system.cpp
main/src/database_sys.cpp
main/src/relay_sys.cpp
main/src/livestream_ctrl.cpp
)

# Main
add_executable(mainlocal.elf ${SOURCES})
target_compile_features(mainlocal.elf PRIVATE cxx_std_11)

# Linking Libraries
target_link_libraries(mainlocal.elf PRIVATE /home/goncalo/
    buildroots/buildroot-2021.08-Fato/output/host/arm-buildroot-
    linux-uclibcgnueabihf/sysroot/usr/lib/libpython3.9.so -
    lpthread -lrt)

#Daemon Configuration

set(DSOURCES
daemon/button_sys.cpp
daemon/door_sys.cpp
daemon/motion_sys.cpp
```

```
)

# Daemon process
add_executable(daemon.elf daemon/maind.cpp daemon/Cdaemon.cpp
    daemon/data_errors.cpp ${DSOURCES}) #${daemon_SRCS}

target_include_directories(daemon.elf PRIVATE
${CMAKE_CURRENT_SOURCE_DIR}/daemon)
```

Listing 2: This project CMakeLists.txt

Upon examination of the 'CMakeLists.txt' file, its evident that the project, named 'CMS_proj,' is configured for the ARM architecture, employing the arm-buildroot-linux-uclibcgnueabihf toolchain. This script establishes the project's fundamental parameters, specifying the minimum required CMake version, designating it as a C++-based project, and configuring the compiler paths to seamlessly integrate with the ARM toolchain.

The file is organized and simple, defining the binary output directory and specifying essential include directories within the project. The project's structure is thoughtfully outlined, featuring well-organized source files catering to both the main and daemon processes. The 'mainlocal.elf' executable, tailored for the main process, exhibits adherence to modern C++ standards, specifically C++11. It links against crucial dependencies such as 'libpython3.9.so,' pthread, and rt. Noteworthy is the foresight to craft a separate 'daemon.elf' executable for the daemon process, complete with its unique set of source files and include directories.

This 'CMakeLists.txt' configuration attests to a systematic and well-considered approach to project organization and compilation for the ARM architecture.

11.3.3 Wpa_supplicant.conf

The wpa_supplicant.conf is a configuration file used by the wpa_supplicant program, which is a widely used open-source software implementation of the WPA (Wi-Fi Protected Access) protocol. This file is commonly employed in Linux and other Unix-like operating systems to provide necessary parameters for connecting to secured Wi-Fi networks.

```
    ctrl_interface=/var/run/wpa_supplicant
    ap_scan=1

    network={
      ssid="OPPOR6"
      psk="oppo12345"
      key_mgmt=WPA-PSK
    }
```

Listing 3: Contents of wpa_supplicant.conf

The presented listing reveals the contents of the program within the wpa_supplicant.conf file, responsible for orchestrating the Wi-Fi connection. This code articulates the specifications for the Wi-Fi connection, including the SSID (Service Set Identifier) and its corresponding password. These parameters are essential for configuring the project's wireless connectivity, ensuring the secure and accurate establishment of the Wi-Fi connection.

11.3.4 Nginx.conf

The nginx.conf file is the command center for configuring the Nginx web server, a powerful tool for hosting websites and managing internet traffic. In this file, we define how the server should behave, what requests it should handle, and how it interacts with other parts of a website. It's like a set of instructions that tell Nginx how to do its job. We'll explore the important parts of this file, such as setting up the server, managing URLs, ensuring secure connections, controlling access, and handling errors.

```
1    rtmp {
2        server {
3                listen 1935;
4                chunk_size 4096;
5
6                application live {
7                        live on;
8                  allow publish 127.0.0.1;
9
10               }
11       }
12   }
```

Listing 4: Contents added to the nginx.conf

The provided listing delineates a crucial addition to the nginx.conf file, a pivotal configuration for enabling RTMP processing within Nginx. This specific code segment introduces an application named "live," serving as the designated destination for streaming content. By incorporating this code into nginx.conf, Nginx is empowered to handle RTMP communication, allowing for the efficient processing and distribution of live streams within the specified "live" application.

11.4 Local System

In this section, we'll take a close look at how we've built and designed four important classes in our system: House_system.h, livestream_ctrl.h, database_sys.h, and relay_sys.h. These classes are like the backbone of our system, each doing something special to make sure everything runs smoothly. We'll break down how we made them, explaining the choices we've made, the methods we've used, and the special things each class can do.

11.4.1 House_system

The primary role of this class, as outlined in the software specification, is to serve as the central controller within the project. It functions as a master orchestrator, seamlessly managing the flow of processes by effectively controlling the multithreading aspects of the program. In essence, it acts as the nexus that receives crucial information from both the daemon process and the database. Subsequently, based on the received data, this class intelligently triggers the appropriate outputs, ensuring the precise execution of the program's functionality. Its purpose is to maintain the overall fluency and coherence of the system's operations, serving as a vital component in the seamless integration of various components and processes.

```
#ifndef HOUSE_SYSTEM_H
#define HOUSE_SYSTEM_H

#include <iostream>
#include <pthread.h>
#include <sched.h>
#include <errno.h>
#include <unistd.h>
#include <mqueue.h>
#include <signal.h>

//subsystems Classes
#include "relay_sys.h"
#include "livestream_ctrl.h"
#include "database_sys.h"

//message queue organization
#define MQ_NAME "/MSGQUEUE_SENSORS"
#define MQ_MAXSIZE 50
#define MQ_MAXMSGS 5

struct sinp_flags{
    bool motion;
    bool door;
    bool button;
};

struct control_flags{
    bool relay;
    bool sound;
    bool live;
};

class houseSystem{
    private:
```

```
36
37      relay_sys relay;
38      livestream_ctrl livestream;
39      database_sys database;
40
41      //Mutexes
42      pthread_mutex_t mutdata;    //mutex for tupdateflags
43      pthread_mutex_t mutrelay;   //mutex for trelay
44      pthread_mutex_t mutsensors; //mutex for tsensors
45      pthread_mutex_t mutsound;   //mutex for tsound
46      pthread_mutex_t mutstream;  //mutex for tlivestream
47
48      //Condition variables
49      pthread_cond_t cvsensors;   //condition var for tsensors
50      pthread_cond_t cvrelay;     //condition var for trelay
51      pthread_cond_t cvsound;     //condition var for tsound
52      pthread_cond_t cvstream;    //condition var for tstream
53
54      //FLAGS
55      sinp_flags house_sen;
56      sinp_flags database_sen;
57      control_flags house_periph;
58      control_flags database_periph;
59
60      bool sensors = 0;
61      bool control_flag = 0;
62
63      //message queue organization
64      struct mq_attr attr_msg;
65      mqd_t msgqueue;
66      char data[MQ_MAXSIZE];
67      unsigned int prio;
68
69      pthread_t thread1, thread2, thread3, thread4, thread5;
70
71      //thread functions
72      static void *tupdateFlags(void* );
73      static void *tstream(void* );
74      static void *tsensors(void* );
75      static void *trelay(void* );
76      static void *tsound(void* );
77
78      static void sigHandler(int sig);
79
80      public:
81
82      houseSystem();              // create all the tasks and
    necessities that will be needed
```

```
83      ~houseSystem();              //destroy all the tasks and
        necessities
84
85      void run();
86  };
87
88  #endif
```

Listing 5: House_System header filer

Upon examination of this class, the initial section reveals essential includes and the creation of two crucial structs: "sinp_flags" (sensor input flags) and "control_flags." These structs serve as containers for storing current values obtained from both sensors, managed by the daemon process ("sinp_flags"), and flags from the database under the caretaker's control ("control_flags").

Within the class, there's meticulous handling of synchronization mechanisms such as mutexes, condition variables, and a message queue. These elements are indispensable for the accurate implementation and control of threads, as well as for facilitating communication with the daemon process through the message queue.

Further into the class, initialization of other essential classes vital for the project's proper functioning takes place. Thread functions are aptly named, encapsulating various components necessary for seamless execution. These components are confined within the private section of the class, emphasizing encapsulation. In the public domain, the class constructor and destructor are provided, along with the "run()" function. The latter serves as the entry point for executing the program, orchestrating the interplay of threads and ensuring the project's coherent operation.

11.4.2 Database_sys

The database_sys class serves as a crucial interface bridging the main process and the chosen database, "Firebase." Its primary purpose is to facilitate seamless communication and data transition between these two components. The following listings highlight key functions and their usage. It's worth noting that this class also acts as an interpreter between C++ and Python, as "Firebase" is not inherently ideal for direct implementation in C++.

```
1  void database_sys::send_data(const std::string& path, const std
       ::string& key, bool value) {
2
3      if (this->pModule != nullptr) {
4          PyObject* pSendData = PyObject_GetAttrString(this->
       pModule, "set_operation");
5
```

```
6         if (pSendData != nullptr && PyCallable_Check(pSendData))
      {
7             PyObject* pArgs = PyTuple_Pack(3, Py_BuildValue("s",
      path.c_str()), Py_BuildValue("s", key.c_str()),
     Py_BuildValue("O", value ? Py_True : Py_False));
8             PyObject_CallObject(pSendData, pArgs);
9
10            Py_DECREF(pArgs);
11            Py_DECREF(pSendData);
12        } else {
13            fprintf(stderr, "Error: The 'set_operation' function
      is not callable.\n");
14        }
15    }
16 }
```

Listing 6: database_sys function to send data to the database

The preceding code snippet, in the listing 6, illustrates the class code responsible for invoking a Python function to transmit data to the database. Subsequently, the following section presents the corresponding Python function that is invoked by the aforementioned C++ class for the purpose of data transmission.

```
1 def set_operation(ref, key, value):
2
3     refi = db.reference(ref)
4     refi.child(key).set(value)
```

Listing 7: py_database.py function to send data

11.4.3 Livestream_ctrl

The core function of this class is to produce live video and audio streams, allowing the caretaker to engage in verbal communication with the person under their care. The upcoming listing will elucidate the process of enabling the livestream, providing a detailed explanation for each segment of the command it invokes. This comprehensive breakdown aims to clarify the intricate components involved in fulfilling the primary objective of the class.

```
1
2 void livestream_ctrl::start_livestream(){
3
4     pid_t pid = fork();
5     if(pid == 0)
6     {
```

```
7         char *args[] = {"ffmpeg", "-re", "-f", "v4l2", "-
      video_size", "320x240", "-thread_queue_size", "16384", "-
      framerate", "70", "-i", "/dev/video0", "-f", "alsa", "-ac", "
      1", "-thread_queue_size", "1024", "-ar", "44100", "-i", "
      plughw:1,0", "-f", "flv", "rtmp://localhost/live", "-c:a", "
      libfdk_aac", "-b:a", "128k", "-c:v", "libx264", "-b:v", "1600
      k", "-qp", "0", "-preset", "ultrafast", "-filter:v", "fps=fps
      =30", "-tune", "zerolatency", "-x264opts", "keyint=50", "-g",
       "25", "-pix_fmt", "yuv420p", NULL};
8
9         execvp("ffmpeg",args);
10
11    } else {
12        // Parent process
13        int status;
14        waitpid(pid, &status, WNOHANG);
15    }
16 }
```

Listing 8: function start_livestream()

The ffmpeg command that is called, captures video from a webcam (/dev/video0) and mono audio from an ALSA audio source (plughw:0,0), encoding them with libx264 for video and libfdk_aac for audio. The video is resized to 320x240 pixels, has a frame rate of 70 fps, and is streamed to an RTMP server at rtmp://localhost/live. The audio is set to mono with a bitrate of 128 kbps. The video encoding utilizes a target bitrate of 1600 kbps, lossless quality (qp 0), ultrafast preset for fast encoding, and additional settings for reduced latency (zerolatency), keyframe interval (keyint=50), GOP size (g 25), and pixel format (yuv420p). The resulting stream is optimized for low latency streaming with specified video and audio quality parameters.

11.5 Daemon

In this section, we will delve into the implementation of the Daemon process, elucidating its functionality and the various classes it encompasses. Furthermore, we will explore the intricate interactions between the main process and the daemon process, shedding light on the seamless collaboration between these components.

11.5.1 Cdaemon

The central component of the daemon process is the ”Cdaemon” class. This class serves as a container for all other essential classes, including those dedicated to handling button, motion, and door sensors. Additionally, two auxiliary files play pivotal roles in this process: ”data_errors,” responsible for relaying crucial process information to a ’.log’ file, and ”interrupt.h,” which acts as a wrapper for interrupt functions.

```
Cdaemon::Cdaemon():
  motion(isr_control),
  button(isr_control),
  door(isr_control)
{

  mq_unlink(MQ_NAME);

    //message queues initialization attributes
    attr.mq_flags = 0;
    attr.mq_maxmsg = MQ_MAXMSGS;
    attr.mq_msgsize = MQ_MAXSIZE;
    attr.mq_curmsgs = 0;

    //create msg queue
    msgqueue = mq_open(MQ_NAME, O_CREAT | O_WRONLY, 0666, &attr)
   ;   //create the msg queue with write only
    if(msgqueue == (mqd_t)-1){
    logError("mq_open");
    mq_unlink(MQ_NAME);
  }

  thisPtr = this;

}
```

Listing 9: Constructor of Cdaemon class

The code snippet in Listing 9 illustrates the implementation of the class constructor. Notably, the constructor initiates by assigning interruption handlers to the sensors, specifically utilizing the isr_control function. Subsequently, the constructor proceeds to establish message queues that facilitate communication with the main process. This initialization process ensures that the daemon process is well-equipped to handle sensor interruptions and maintain effective communication channels with the main process.

```
    void Cdaemon::isr_control(int control, siginfo_t *info, void
    *unused) {
    std::string msg;  // Adjust the size as needed

    switch (control) {
        case SIGUSR2:
      logError("[SIGNAL] Sigusr2 triggered button ISR");
      thisPtr->button_flag = 1;
            send_message = 1;
            break;

        case SIGUSR1:
      logError("[SIGNAL] Sigusr1 triggered door/motion ISR");
```

```
13
14      if(info->si_value.sival_int == 40){
15        msg = "[VALUE] sensor Motion with value: " + std::
    to_string(info->si_value.sival_int) + '\0';
16        logError(msg);
17        thisPtr->motion_flag = 1;
18      }
19      if(info->si_value.sival_int == 30){
20        msg = "[VALUE] sensor Magnet with value: " + std::
    to_string(info->si_value.sival_int) + '\0';
21        logError(msg);
22        thisPtr->door_flag = 1;
23      }
24            send_message = 1;
25            break;
26
27        default:
28            send_message = 0;
29            logError("[ERROR] Signal was undefined");
30            break;
31    }
32 }
```

Listing 10: isr_control function

The preceding listing provides insight into the operation of the interruption handler. This handler efficiently captures signals activated by the device drivers of each sensor. By interpreting the transmitted values, the process discerns which specific sensor was triggered, enabling precise identification and response to the activated sensor.

11.5.2 Button, Motion, Magnetic

In this section, we will explore the implementation of sensor classes responsible for managing their corresponding device drivers. Given the structural similarities among these device drivers, the implementations of these sensor classes share common features. The following snippets will illustrate how these sensor classes interact both with their respective device drivers and the main daemon class.

```
1 motion_sys::motion_sys(ISR isr){
2
3     sigemptyset(&sig.sa_mask);
4     handler = isr;
5
6 }
```

Listing 11: motion constructor

In Listing 11, the motion class constructor is depicted, where the ISR handler is defined as an input parameter. Additionally, the signal is cleared within the constructor through the utilization of sigemptyset(&sig.sa_mask).

```
    void motion_sys::enable(){

    dev_str = "/dev/" + (std::string)name;
    pid_t pid;
    uid_t pidu;
    previous_val = false;
    //open and enable device driver

    device = open(dev_str.c_str(), O_RDWR);      //open the
   specific device driver

    if(device < 0)
        logError("[ERROR] Device Driver NOT found");

    pid = getpid();

    if(ioctl(device, IOCTL_PID, &pid))
  {
    close(device);
    logError("[ERROR] Failed system call. Closing device driver"
   );
  }

    sig.sa_sigaction = handler;
    sig.sa_flags = SA_SIGINFO;
    if (sigaction(SIGUSR1_HANDLER, &sig, NULL) == -1) {
         logError("[ERROR] Failed to set up signal handler");
    }

}
```

Listing 12: motion enable function

The provided code (Listing 12) showcases the "enable" function in the motion class, which is responsible for initializing and starting the associated device driver. Here's a simplified explanation:

In this function, the motion class prepares to use its device driver by constructing the file path, opening the driver, and checking for errors. It then retrieves the process ID and communicates this information to the device driver. The function also sets up a signal handler to respond to specific signals triggered by the driver.

11.6 Device Drivers

In this section, we'll explore the implementation of the device driver for the motion sensor. Since the structure is quite similar to that of the button and magnetic sensor drivers, we'll focus on highlighting the main differences. Additionally, we'll discuss the key distinctions between the motion sensor device driver implementation and the relay device driver. The motion sensor relies on reading values triggered by the sensor, while the relay is activated by a flag under the user's control.

Reiterating, the primary distinction between the motion device driver and the relay device driver lies in their mechanisms of activation. The motion sensor utilizes interrupts to signal user-level processes, responding dynamically to sensor-triggered events. On the contrary, the relay device is activated through explicit user control, where the caretaker can manage its state as needed. This fundamental difference in activation methods caters to the unique functionalities and requirements of each device within the system.

```
1  static irqreturn_t irq_handler(int irq, void *dev_id)
2  {
3    int pinVal = gpio_get_value(pinNum);
4
5  //Debounce mechanisms
6  #ifdef EN_DEBOUNCE
7    unsigned long diff = jiffies - old_jiffie;
8
9    if (diff < 200)
10   {
11     return IRQ_HANDLED;
12   }
13
14   old_jiffie = jiffies;
15 #endif
16
17   printk(KERN_INFO "[MOTION] Interruption handler: PIN -> %d.\n"
       , pinVal);
18
19   info.si_signo = SIGH;
20   info.si_code = SI_QUEUE;
21   info.si_int = 40;
22
23   task = pid_task(find_pid_ns(pid, &init_pid_ns), PIDTYPE_PID);
24
25   if(task != NULL)
26     send_sig_info(SIGH, &info, task);
27
28   return IRQ_HANDLED;
29 }
```

Listing 13: motion device driver interrupt handler

In Listing 13, the interrupt handler plays a vital role in activating the 'SIGUSR1' signal, subsequently received by the daemon process, as demonstrated in Listing 10. Importantly, the handler also communicates an integer value (such as '40') to the daemon process, serving as an identifier for this specific motion sensor.

This identification mechanism is crucial due to the potential shared use of the 'SIGUSR1' signal among multiple sensors, such as motion and magnetic sensors. The unique integer value allows the daemon process to discern the signal's source when handling multiple simultaneous signals.

Additionally, a debounce mechanism is incorporated within the driver implementation, as seen in the provided code snippet. This mechanism helps prevent unintended rapid triggering of interrupts. When the 'EN_DEBOUNCE' flag is enabled, the interrupt handling function calculates the time difference between the current and previous jiffies (timestamps). If the difference is less than 200 milliseconds, the function promptly returns 'IRQ_HANDLED', effectively suppressing rapid interruptions.

Looking forward, to facilitate the real-time integration of more sensors and support the dynamic addition of new sensors, the device driver will undergo extensions. This will involve enhancing the initialization function to receive and process identification numbers effectively, laying the groundwork for seamless incorporation of additional sensors in future iterations.

```
1  static int __init motion_driver_init(void)
2  {
3    if ((alloc_chrdev_region(&dev, 0, 1, DEVICE_NAME)) < 0)
4    {
5      printk(KERN_INFO "[MOTION] Cannot allocate major number\n");
6        return -1;
7    }
8
9    /*Creating cdev structure*/
10   cdev_init(&c_dev, &fops);
11
12   /*Adding character device to the system*/
13   if((cdev_add(&c_dev,dev,1)) < 0)
14   {
15     printk(KERN_INFO "[MOTION] Cannot add the device to the
       system\n");
16     unregister_chrdev_region(dev,1);
17   }
18
19   /*Creating struct class*/
20   if((dev_class = class_create(THIS_MODULE, CLASS_NAME)) == NULL
       )
21   {
22     printk(KERN_INFO "[MOTION] Cannot create the struct class\n"
       );
```

```
23      unregister_chrdev_region(dev,1);
24    }
25
26    /*Creating device*/
27    if((device_create(dev_class, NULL, dev, NULL, DEVICE_NAME)) ==
        NULL)
28    {
29      printk(KERN_INFO "[MOTION] Cannot create the Device\n");
30      class_destroy(dev_class);
31      unregister_chrdev_region(dev,1);
32    }
33
34    irqNumber = gpio_to_irq(pinNum);
35
36    if (request_irq(irqNumber, irq_handler, IRQF_TRIGGER_RISING,
       DEVICE_NAME, (void *)(irq_handler)))
37    {
38      printk(KERN_INFO "[MOTION] Cannot register IRQ\n");
39      free_irq(irqNumber,(void *)(irq_handler));
40      class_destroy(dev_class);
41      unregister_chrdev_region(dev,1);
42    }
43
44    s_pGpioRegisters = (struct GpioRegisters *)ioremap(GPIO_BASE,
       sizeof(struct GpioRegisters));
45    SetGPIOFunction(s_pGpioRegisters, pinNum, GPIO_INPUT);
46
47    return 0;
48  }
```

Listing 14: motion device driver initialization

The initialization function in Listing 14 serves as the entry point for setting up the motion sensor device driver. The process begins by allocating a region of character device numbers and initializing the character device structure ('c_dev'). This is essential for the device to be recognized and interacted with by the Linux kernel.

Following that, the function creates a device class and a corresponding device. The device class helps organize and manage devices within the kernel, while the device is a representation of the motion sensor in the kernel's sysfs (system file system).

The subsequent part of the initialization focuses on configuring the interrupt handling for the motion sensor. It involves converting a GPIO pin to its corresponding IRQ number and requesting an interrupt line. The interrupt line is associated with the specified IRQ handler ('irq_handler'). This setup allows the motion sensor to generate interrupts and notify the system when motion is detected.

Finally, the initialization function configures the GPIO pin for the motion sensor. It involves mapping physical addresses to virtual addresses for input/output operations and setting the GPIO pin as an input. This step ensures that the system can communicate with

the motion sensor through GPIO signals.

11.7 Database

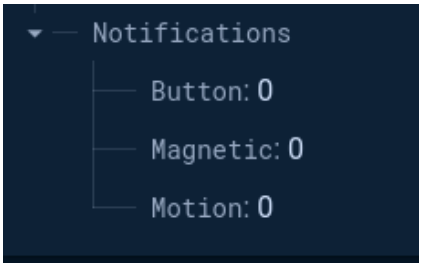
As previously mentioned, the chosen database for this project is Google's Firebase, renowned for its robust and scalable backend infrastructure that greatly streamlines application development and deployment processes. Firebase's real-time NoSQL database stands out as a cornerstone, enabling seamless data synchronization across diverse devices and clients in real-time. This feature perfectly aligns with the project's needs for dynamic and collaborative data management, ensuring that changes made by one user are instantly reflected across all connected devices.

Beyond the real-time database, Firebase offers a diverse array of services, each contributing to the platform's versatility and comprehensive support for application development. Some key services include:

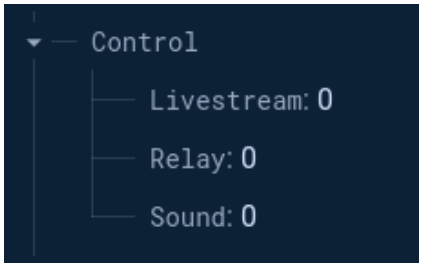
- **Authentication Services**: Firebase simplifies user authentication, providing secure mechanisms for user sign-ins and identity management.
- **Cloud Functions**: Developers can leverage serverless cloud functions to execute server-side logic without the need to manage underlying server infrastructure.
- **Cloud Storage**: Firebase facilitates efficient file storage in the cloud, making it seamless for applications to manage and retrieve user-uploaded files.
- **Hosting**: Firebase Hosting offers a straightforward solution for deploying web applications, complete with features such as content delivery network (CDN) delivery and SSL support.

The serverless architecture of the platform alleviates the burden of intricate backend management, providing our project with the freedom to focus on crafting and enhancing application features. Firebase's seamless integration of services, coupled with its user-friendly interface, positions it as an ideal solution for our project's needs. Specifically tailored for rapid development, real-time data synchronization, and scalable backend services, Firebase aligns perfectly with the demands of our project, ensuring efficiency and flexibility throughout the development process. After this explanation of "why" was this database chosen for this project in the following images will be shown how the data is displayed inside the user friendly UI of the Firebase:

The images presented earlier provide a visual representation of the designated "folders" containing both control flags and notification flags. Within the Control flags, figure 68b category, there are three distinct flags, namely "sound," "video," and "relay." The "sound" flag becomes active when a user initiates an audio interaction with the caregiver, while the "video" flag is triggered upon the user accessing the livestream feature in the application.



(a) Firebase Notifications folder



(b) Firebase Control folder

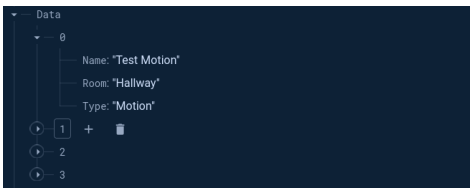
Figure 68: Firebase flags folders

The "relay" flag is designed to simulate user control over various objects within the local system, allowing for interaction with appliances or devices like stoves and lightbulbs. These graphical representations visually guide the understanding of the folders and their associated control flags, facilitating a clearer comprehension of the user-centric functionalities within the system.

In contrast, the Notification flags in figure 68a, showcased in the images, serve as mechanisms through which the Local System communicates information to the user. The "motion," "button," and "door" flags encapsulate specific events. The "motion" flag, for instance, symbolizes the activation of motion sensors, indicating the detection of movement within the environment. The "button" flag, a central component of the project's alert system, enables the person in care to initiate communication with the user. Additionally, the "door" flag provides insights into the state of a door, signaling whether it is open or closed. The visual representation of these notification flags aids in elucidating their respective functionalities, enhancing the overall understanding of the communication dynamics within the system.

The following images will show the other "folders" that exist in this project:

(a) Firebase Logs folder



(b) Firebase Data folder

Figure 69: Firebase Logs and Data folders

The images provided earlier offer insights into the "Data" folder, figure 69b, serving as a repository for comprehensive information pertaining to sensors and other peripherals accessible to the user or transmitting information to the user. Within this folder, there are

currently four controllers represented: motion, button, door, and relay. Each controller is characterized by distinct attributes such as location, name, and type, facilitating a clear identification of the purpose of each peripheral. This organization within the "Data" folder ensures an orderly arrangement of sensor-related data, enabling efficient access and management for users interacting with the system.

In addition to the "Data" folder, the images showcase the "Logs" folder, figure 69a, an integral component containing a historical record of sensor triggers. Each time a sensor is activated, regardless of the triggering reason, a corresponding message is logged in this repository. The logs provide valuable information including the date and the specific sensor that was triggered. This systematic recording of sensor events in the "Logs" folder not only serves as an audit trail but also facilitates retrospective analysis, aiding in troubleshooting, system optimization, and ensuring a comprehensive overview of the system's historical sensor activities.

12 Verification

In this section, the verification process of the project will be outlined. These tests were conducted to validate the operational modes and ensure the seamless integration of components within the project. To streamline information, the subsequent subsections are categorized into different tests, each serving a distinct purpose. This systematic approach to verification aims to validate the functionality, reliability, and overall performance of the project components, providing a comprehensive understanding of its capabilities and adherence to project specifications.

12.1 Local System

In the forthcoming subsection, we will delve into the verification processes for the project components. The Local system, elucidated earlier, serves as the project's cornerstone, functioning as the primary controller, intermediary between the User and the person in need, and the livestream server. It is crucial to comprehensively test and validate these functionalities to ensure the system's reliability and effectiveness. It is worth noting that while the current implementation entails the Local system acting as the livestream server, future iterations may explore alternative implementations for enhanced efficiency and scalability.

During this subsection, a visual representation will be provided through images demonstrating the correct functioning and activation of the implemented modules in the project. These images serve as tangible evidence of the system's performance, showcasing the successful execution of various functionalities. The visual verification aids in validating the seamless integration of components and ensures that each module operates as intended, contributing to the overall efficacy of the project.

12.1.1 Sensors & Relay

The sensors are composed by a motion sensor, a magnetic sensor and a button. As was referenced earlier every sensor and relay have its own device driver. To guarantee if they work as expected, they were tested by a simple code using the gpiod library that uses a simple for loop that will turn of and on the relay for a few times and each time will read the values from the sensors, the following listing is snippet of the code that was created for this first stage:

```
    while(1){

        if(gpiod_line_get_value(sensor[1]))
            printf("\n -> ON");
        else
            printf("\n -> OFF");

            if(prev_clk != gpiod_line_get_value(sensor[1])){
       //every time that the clock changes value

                if(prev_clk == 0 && gpiod_line_get_value(sensor
[1]) == 1){

                    printf("\n -> RISING EDGE CLK");
                    printf("\nValue Read from MASTER: %d, %d",
gpiod_line_get_value(sensor[2]),gpio_pin[2]);
                    prev_clk = gpiod_line_get_value(sensor[1]);

                }else if(prev_clk == 1 && gpiod_line_get_value(
sensor[1]) == 0){

                    printf("\n -> FALLING EDGE CLK");
                    printf("\nValue Read from MASTER: %d",
gpiod_line_get_value(sensor[2]));
                    prev_clk = gpiod_line_get_value(sensor[1]);

                }
            }
    }
```

Listing 15: Sensor & relay test example

After the tests were made the device drivers were implemented as it was explained in the subsection 11.6. To install and check the correct working of the device drivers the following commands were needed:

```
# insmod button.ko
# insmod relay.ko
```

```
# insmod magnetic.ko
# insmod motion.ko
# lsmod
# dmesg
```

Listing 16: Commands used to install and test device drivers

The initial four commands facilitate the installation of the new .ko files, referred to as loadable kernel modules. These modules, akin to object files, serve to extend the Linux kernel of the distribution, essentially constituting the newly created device drivers. Subsequent to their installation, the efficacy of the process is confirmed by executing the 'lsmod' command, as illustrated in Figure 70. This command validates the presence and activation of the loadable kernel modules within the system.

Following the module installation, a critical check is performed by executing the 'dmesg' command, as showcased in Figure 71. This command allows for the examination of kernel messages, ensuring the absence of errors during the initialization process.

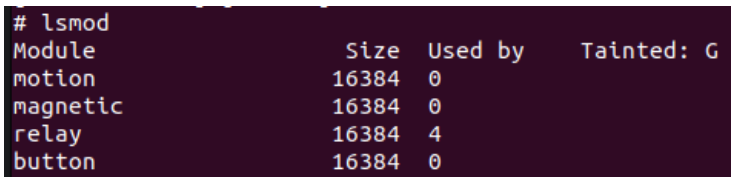


Figure 70: lsmod command

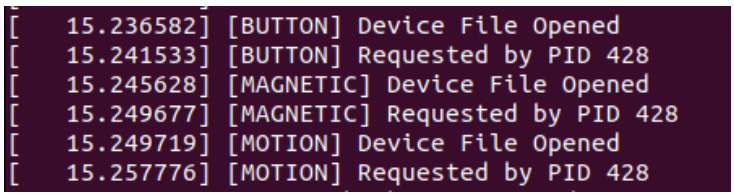


Figure 71: dmesg command device drivers initialization message

After successfully creating the respective device drivers, several tests were conducted to verify their correct implementation. The captured image, obtained using the 'dmesg' command, showcase the device drivers sending messages upon the occurrence of an interrupt, specifically when a rising edge is detected. The visual representation in Figure 72 provides a glimpse into the successful activation of the interrupts, affirming the proper functionality of the implemented device drivers.

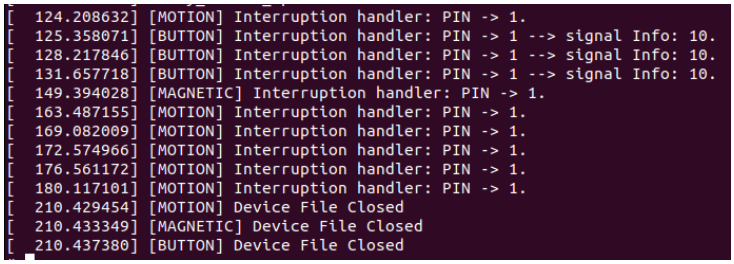


Figure 72: dmesg command device drivers IRQ handled

12.1.2 Camera

Before testing the camera it was needed to first test if the Raspberry pi 4B detects the camera otherwise the connection or the camera could be faulty. The following command was used to test this:

```
1    # vcgencmd get_camera
```

Listing 17: Command to test if Raspberry recognizes the camera

The command will return the following string "supported=X detected=X", the X represents an intenger number bigger or equal to 0. For the camera to be implemented both of this values must be one or bigger, as expected the values were one for both, represented in the figure 74.

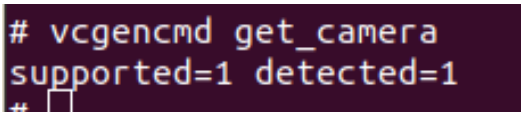


Figure 73: Command: vcgencmd get_camera

Subsequent to the module installation and verification, tests were conducted to assess the camera functionality. However, an issue arose during the testing phase where the night vision component of the camera proved challenging to produce. Consequently, for this prototype, the night vision functionality was regrettably discarded. It is imperative to note that while this feature was omitted in the current iteration, future updates and iterations of the project will focus on addressing and implementing night vision capabilities to enhance the overall functionality of the system. The following figure was taken the raspberry pi camera and some configurations were made to take the be

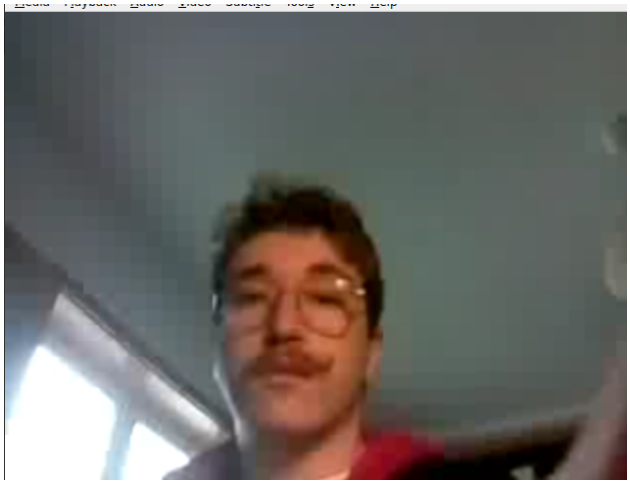


Figure 74: Test camera picture

12.1.3 Microphone & Speaker

As the primary focus of this project lies in the livestream and bidirectional communication functionalities, the sound department plays a crucial role. To address this, alsa-utils was employed to manage sound production and recording. The following commands were executed to identify the existing drivers for each sound component.

```
# arecord -l
# aplay -l
```

Listing 18: Alsa-utils commands

The initial command displays the existing drivers capable of recording sound, as depicted in Figure 75.

```
# arecord -l
**** List of CAPTURE Hardware Devices ****
card 0: Sound [Corsair HS45 Surround USB Sound], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: Device [USB PnP Sound Device], device 0: USB Audio [USB Audio]
  Subdevices: 0/1
  Subdevice #0: subdevice #0
```

Figure 75: Command arecord -l

Subsequently, another command is executed to showcase the drivers available for sound production, illustrated in Figure 76.

```
# aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: Sound [Corsair HS45 Surround USB Sound], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

Figure 76: Command aplay -l

12.1.4 Livestream

As previously highlighted, the livestream constitutes an important aspect of this project. To validate its functionality, a specific command, demonstrated during the implementation section under the 'livestream_ctrl' subsection, was employed. This command, outlined for video streaming, is as follows:

```
ffmpeg -f v4l2 -video_size 640x480 -framerate 30 -i /dev/video0
    -c:v libx264 -preset ultrafast -b:v 1000k -f flv rtmp://
    localhost/live
```

Listing 19: Livestream Test Command

This command successfully streamed live video from the camera to the localhost IP. The live video feed was then captured by the VLC media player for verification of correct functionality. However, due to prolonged processing times for frame transmission, it became necessary to reduce the image quality. Subsequently, a modified command was implemented to address this issue while also incorporating audio streaming. The ensuing images aim to showcase the results of these tests, highlighting the optimized video and audio streaming functionalities achieved.

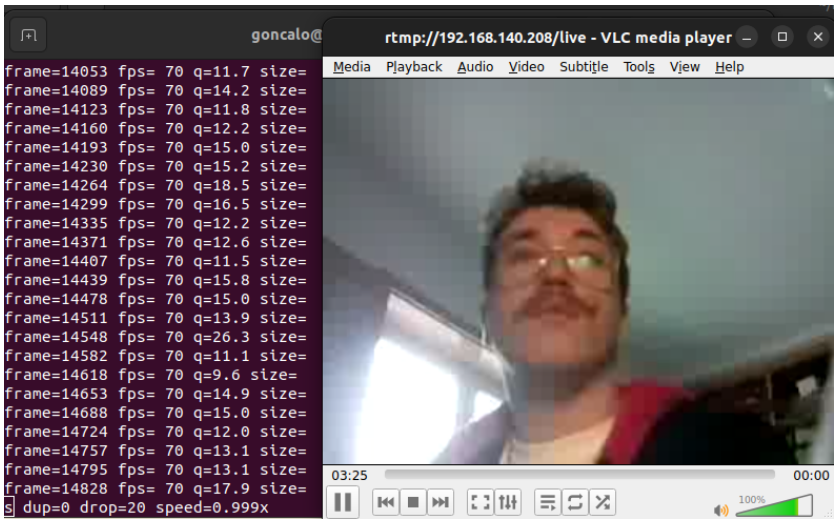


Figure 77: Test live audio and video

12.2 Test Cases

In this section, the defined Test Cases from the Design stage will undergo comprehensive scrutiny to ensure that all planned elements have been successfully implemented. Each test case serves as a critical benchmark, validating the adherence of the project to the initially outlined specifications and requirements. The subsequent evaluation aims to affirm the system's functionality, reliability, and conformance to the predetermined design parameters.

12.2.1 Hardware Test Cases

Here are the results of the hardware test cases. While many objectives were successfully achieved, it is important to note that one test case could not be implemented due to hardware malfunctioning. The following table provide insights into the system's hardware performance..

Test Cases	Expected Result	Real Result
	Motion Sensor	
Movement in the sensor	SIGNAL pin high (5V)	Correct
No Movement in the sensor	SIGNAL pin low (0V)	Correct
	Button	
Press button	Output pin high (5V)	Correct
	Microphone	
Record audio	Audio recorded equals to the one inputted	Correct
	Speaker	
Play audio	Audio played correctly	Correct
	Camera	
Capture image	Frames Captured correctly	Correct
Light mode switching	Changes back and forth to night mode automatically	Unable
	Magnetic Sensor	
Magnet detected	Output pin high (5V)	Correct
	Relay	
Relay activated	Output voltage (230V)	Correct

Table 8: Hardware test cases Results

12.2.2 Software Test Cases

In this stage, the results of the software test cases are presented. The majority of the test cases were successfully achieved. Instances where the test cases were not marked as successful were primarily due to the fact that while the functionality worked, the final implementation was deemed suboptimal. The following outcomes provide a comprehensive overview of the software test cases, showcasing areas of success and highlighting opportunities for refinement in the final implementation.

Test Case	Expected Result	Real Result
	Local System	
Send message to queue	Message written correctly	Working
Read message from queue	Message read correctly	Working
Send notification	Database flag updated	Working
Process sensor trigger	Distinguish which sensor was triggered	Working
Process video and audio	Correctly put video and audio data together	Working
Live stream	Send video and audio to server successfully	Working
	Remote System	
Play livestream	Successfully retrieve livestream from server	Working
Play recorded video	Successfully retrieve recorded video from database	Not Working
Register account	Update database with new account	Working
Login into account	Retrieve login credentials from database and compare them	Working
Logout of account	Logout valid	Working
Enable/Disable Relay	Update Relay Flag on database	Working

Table 9: Software test cases Results

12.2.3 Integration Test Cases

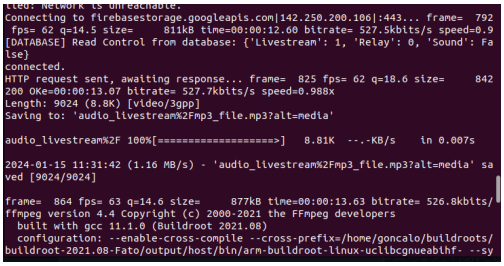
Almost all Integration Tests were achieved, underscoring the seamless integration and collaboration of individual components within the system. However, it is noted that the 'Record Video' test case remains unimplemented. While the majority of integration objectives were met, the absence of video recording functionality serves as a potential avenue for future enhancements. The results of the integration testing reinforce the robustness and cohesiveness of the system, setting a solid foundation for future developments and improvements.

Test Cases	Expected Result	Real Result
Press Button	Notify the Caretaker User	Working
Motion Detected	Notify the Caretaker User	Working
Door Open	Notify the Caretaker User	Working
Control Relay	Enable/Disable Relay	Working
Register Account Sucessfully	Show Successful Message	Working
Configure Sensor	Enable/Disable each sensor notification	Working
Sensor Logs	Display sensor logs	Working
Click on "Live Stream"	Display Live Stream	Working
Click on "Mic Button"	Stream Audio; Activate Speaker	Working
Click on "Recorded Video"	Display Recorded Video	Not Working

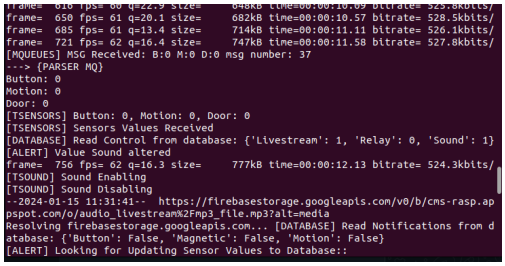
Table 10: Integration Test Cases Results

The integration test results, as depicted in the preceding table, are complemented by a visual representation in Figures and 78 79. These images encapsulate pivotal moments in the project's operational flow, offering a real-time glimpse into its functionality. The figures chronicle the initialization process, sound reception and playback, as well as concurrent video and audio streaming—all harmoniously interacting with the database.

Figure 78: Threads creation and testing



(a) Sound received



(b) Frames prints

Figure 79: Serial prints showcasing the working of this project

This visual narrative aligns with the observed outcomes in the integration test, affirming the successful collaboration of diverse components. Together, these elements furnish a comprehensive understanding of the project's robust and synchronized functioning during the integration phase.

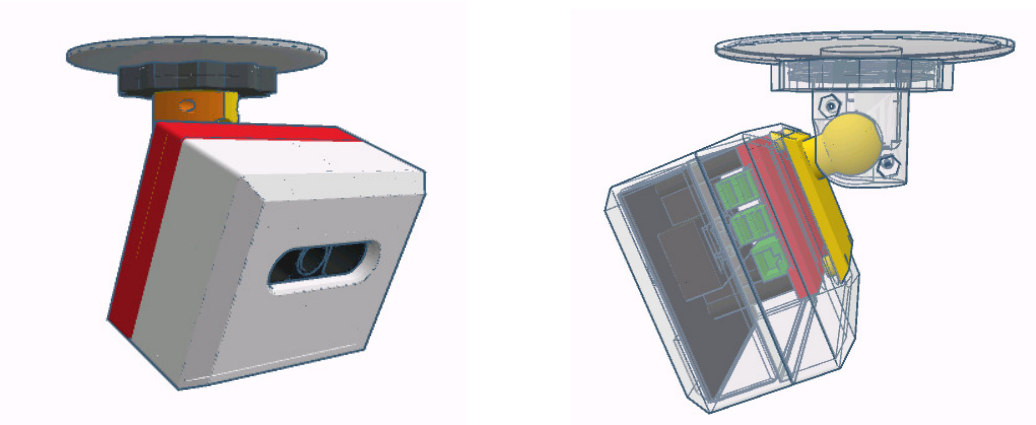
13 Final Product

In this section, the final prototype of the project is presented, encompassing both the layout of the local system and the design of the accompanying mobile application. It's important to note that these representations are subject to future alterations to better accommodate necessary upgrades and refinements. The local system is comprised of two main components: the camera component and the button. Currently, both the button and relay are housed within the same container, but future iterations will involve their separation, with the relay being appropriately positioned. These modifications aim to enhance the overall functionality and aesthetics of the local system, ensuring its adaptability to evolving project requirements and technological advancements.

13.1 Local System

In this subsection, the anticipated design envisions the placement of the Raspberry Pi 4 B, camera, motion sensor, microphone, and speaker in proximity, forming a central hub. This configuration is designed to represent a unified location for core components, facilitating a cohesive and integrated system. In contrast, the button, magnetic sensor, and relay are positioned outside this central hub. This intentional arrangement simulates a real-world scenario where components like the button and sensors may be distributed throughout a house based on their specific functionalities and use cases. The objective is to create a design that aligns with practical and functional considerations within a domestic setting.

The subsequent images depict simulation renders of the envisioned output for the main component, showcasing the spatial arrangement of the Raspberry Pi 4 B, camera, motion sensor, microphone, and speaker. These renders offer a visual representation of the desired configuration, emphasizing the cohesive integration of essential components within a central hub. This simulation aims to provide a glimpse into the anticipated design and layout, fostering a clear understanding of the spatial relationships between key elements in the system.



(a) Overview of the design

(b) Side view with internals

Figure 80: Possible indoor final design

This design, acquired from Thingiverse user stmorgan, is licensed under Creative Commons - Attribution. All design contributions and credit for this particular configuration go to the original creator. This design serves as a foundational reference for the spatial organization of components in the envisioned system, contributing to the overall aesthetic and functional considerations.

In envisioning an outdoor variant for the project, the design evolves to meet the demands of the external environment. Unlike the indoor design, the outdoor setup is tailored for robustness and durability, equipped to withstand varying weather conditions and external elements. The structure is more expansive, accommodating additional components and featuring enhanced infrared (IR) LED capabilities to ensure optimal visibility during nighttime.

The outdoor design is strategically crafted to house all necessary components securely. Its larger footprint allows for the integration of a more substantial power supply, offering extended operational endurance. The heightened robustness of the structure aims to safeguard the embedded system from environmental challenges, ensuring uninterrupted functionality.

Moreover, the increased number of IR LEDs contributes to improved nighttime surveillance, enhancing the system's ability to monitor and ensure the well-being of the person in care.

(a) Overview of the design (b) Side view

Figure 81: Possible outdoor final design

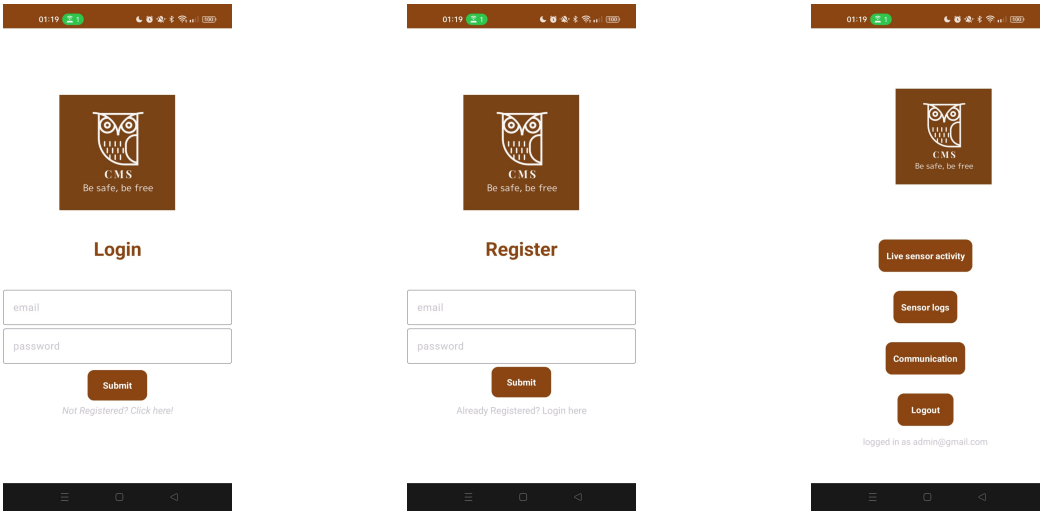
This design, credited to Thingiverse user mperkins905, is licensed under Creative Commons - Attribution. All design contributions and credit for this particular configuration go to the original creator.

This dual-design approach demonstrates adaptability, catering to both indoor and outdoor scenarios, and reflects the project's commitment to providing comprehensive and flexible solutions for those in need.

13.2 Remote System

The final design of the remote system diverged significantly from the initially envisioned layout. While certain aspects deviated, many of the anticipated capabilities were successfully incorporated. It is crucial to recognize that this prototype's final design represents a stepping stone, and future iterations are foreseen to feature enhanced design elements and a more user-friendly interface. The continuous improvement and refinement of the system's design are integral aspects of the project's evolution and ongoing development.

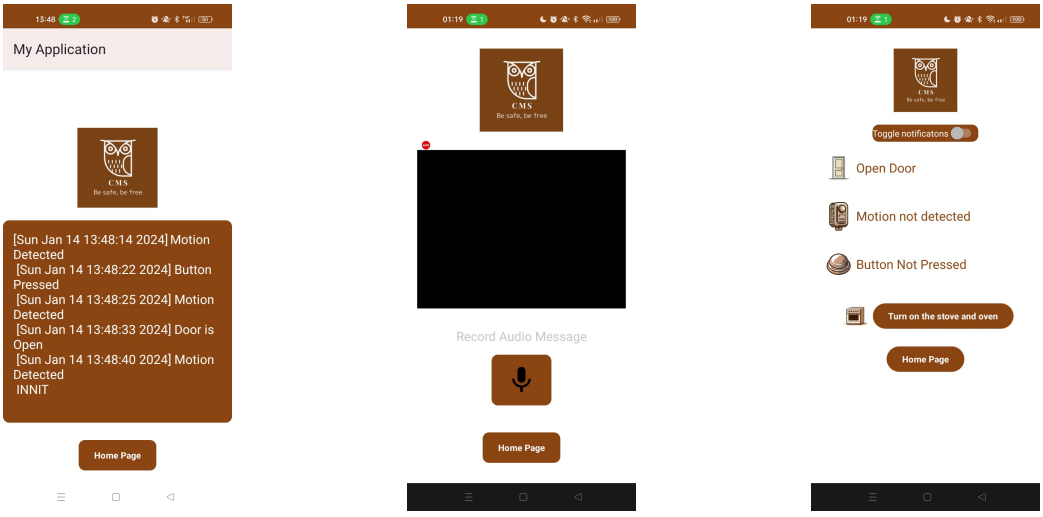
The following Figures show the User Login (Figure 82a), Register (Figure 82b) and Landing (Figure 82c) pages:



(a) Login page (b) Register page (c) Landing page

Figure 82: Final prototype Login, Register and Landing Pages

The figure 82 illustrates the Login, Register, and Landing pages, marking the initial interaction points for users upon accessing the app. Ensuring a positive user experience, these pages embody simplicity, user-friendliness, and intuitive design. As users progress beyond these introductory pages, they encounter the Logs, Livestream, and Notifications pages, each serving distinct purposes:



(a) Logs page (b) Livestream page (c) Notifications page

Figure 83: Final prototype for Logs, Livestream and Notifications pages

The Logs page (figure 83a) meticulously displays information about triggered sensors throughout the project's lifespan. Moving to the Livestream page (figure 83b), users are presented with a window for livestream viewing and a microphone icon to facilitate communication with the person in care. Lastly, the Notifications page (figure 83c) offers real-time updates on sensor statuses, notification toggles, and potential control over household appliances or rooms in future iterations.

The design philosophy emphasizes a harmonious blend of simplicity and functionality, ensuring an accessible and effective user interface.

14 Conclusion

14.1 Conclusion

The primary goal of this project was to create an effective system, titled "Communication and Monitoring System for people with Alzheimer and Dementia," designed to assist individuals dealing with Alzheimer's and Dementia. The core functionalities included the ability to livestream both video and audio, leveraging multiple sensors to establish a comprehensive perception of the monitored environment. Additionally, the system aimed to facilitate communication with the individual in care, while also providing control over various home appliances and objects that could potentially pose harm to the person in need. The overarching objective was to ensure user-friendly accessibility through a mobile application, enabling remote control and monitoring of the local system from any location. This holistic approach sought to enhance the overall well-being and safety of individuals dealing with Alzheimer's and Dementia, fostering a secure and supportive living environment.

This project played a pivotal role in advancing our capabilities in the design and programming of embedded systems. It provided a valuable learning experience, particularly in the areas of developing device drivers, implementing Posix Threads, and utilizing Message queues—all within the buildroot system. The intricacies involved in managing complexity within the project underscored the importance of careful planning and systematic implementation, as evident in the adopted waterfall model. Each stage of the project was meticulously defined and designed to simplify future work, emphasizing the significance of structured workflows.

One key takeaway from this endeavor was the recognition that every project involves trade-offs. Balancing various aspects, such as functionality, resource utilization, and user interface design, required thoughtful consideration and decision-making. As engineers, this project emphasized the importance of taking these trade-offs into account and making informed choices to meet the project objectives effectively.

In conclusion, the project not only enriched our technical skills but also instilled valuable insights into project management and decision-making processes.

14.2 Future Work

In the context of future improvements, several key objectives that could not be implemented in the current version of the project are earmarked for prioritized implementation. These include enhancing night-vision capabilities, enabling the recording of live video and audio, and introducing modularity for seamlessly integrating new modules like cameras and motion sensors. To accommodate these advancements, the remote system will undergo refinement to support the integration of these new functionalities. Additionally, enhancing communication channels and providing clearer insights into device driver information will be integral aspects of the remote system's improvement. Another notable consideration is the implementation of person detection to monitor and identify potential concerns such as someone falling or crawling. Simple yet effective measures like scream detection and voice commands will be explored, allowing the person in need to alert the caretaker or automatically call emergency services if immediate assistance is required. These proposed enhancements aim to further elevate the system's capabilities in providing comprehensive care and support for individuals with Alzheimer's and dementia.

References

[1] A vision-based home security system using opencv on raspberry pi 3. 2019.

[2] Ms. Renuka Chuimurkar. Smart surveillance security monitoring system usingraspberry pi and pir sensor. *International Journal of Scientific Engineering and Applied Science (IJSEAS)*, 2016.

[3] EmbeTronicx. Sending signal from linux device driver to user space.

[4] ffmpeg. Capture / webcam.

[5] Alex Garnett. How to set up a video streaming server using nginx-rtmp on ubuntu 20.04.

[6] David Emanuel Ribeiro Gaspar. Raspberry pi: a smart video monitoring platform, 2014.

[7] globenewswire. Home security market. *https://www.globenewswire.com/en/news-release/2023/01/24/2594779/0/en/Home- Security-System-Market-Size-to-Touch-USD-106-3-Billion-By-2030-As-Per-Acumen-Research- and-Consulting.html*.

[8] Google. Firebase documents.

[9] Video surveillance using raspberry pi architecture. 2015.

[10] Alessandro Rubini Jonathan Corbet and Greg Kroah-Hartman. *Linux Device Drivers*. O'Reilly, 2005.

[11] Tamas Karoly. Exchange data between device drivers and user applications. Technical report, Friedrich-Alexander-Universit ¨at Erlangen-Nurnberg, 2015.

[12] Michael Kerrisk. *The Linux Programming Interface*. William Pollock, 2010.

[13] mordorintelligence. Industry reports. *https://www.mordorintelligence.com/pt/industry-reports/smart-home-security-market*.

[14] Multitasking and Cconcurrency.pdf by Professor Adriano Tavares.

[15] nytimes. Best home security system. *https://www.nytimes.com/wirecutter/reviews/the-best-home-security-system*.

[16] OECD. Health glance 2017. *https://www.oecd-ilibrary.org/social-issues-migration-health/health-at-a-glance-2017/dementia-prevalence_health_glance-2017-76-en*.

[17] Oscar Perez. Shell, init files, variables and expansions, 2022.

[18] David Plowman. Raspberry pi camera module: Still image capture, 2023.

[19] PThreads Programming.pdf by Professor Adriano Tavares.

[20] Rolf van Gelder. Raspberry pi – getting audio working.