Spring 2024

Smart Dog Door

Andrew Shetler
aas183@uakron.edu

William Boissoneault
whb12@uakron.edu

Jacob Stump
jgs71@uakron.edu

Benjamin Charlson
bjc122@uakron.edu

Follow this and additional works at: https://ideaexchange.uakron.edu/honors_research_projects

Part of the Computer Engineering Commons, and the Electrical and Computer Engineering Commons

Please take a moment to share how this work helps you through this survey. Your feedback will be important as we plan further development of our repository.

Recommended Citation
Shetler, Andrew; Boissoneault, William; Stump, Jacob; and Charlson, Benjamin, "Smart Dog Door" (2024). Williams Honors College, Honors Research Projects. 1825. https://ideaexchange.uakron.edu/honors_research_projects/1825

This Dissertation/Thesis is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.
Honors Research Project

Smart Dog Door

Individual Student Contributions

William Boissoneault

My contribution to this project is to design and implement a solenoid based locking system. Upon setting out, the goal was to have an energy efficient locking system and sensor for accurate locking and unlocking. To have the locking arm actuated by the microcontroller, I collaborated with Jacob Stump to ensure that my designed circuitry was compatible with his selected board. Through iteration, an NMOS driving circuit is used to allow for electrical isolation from the high current solenoid and the microcontroller. Also, the microcontroller has a control loop through a reed switch actuated by a magnet in the door to allow the microcontroller to know when the door is in place and prepared to be locked again. Finally, once I learned how to use KiCad to draft PCB’s, I took on a supportive role to Benjamin Charlson in ordering and properly exporting gerber and drill files for fabrication.

Benjamin Charlson (Honors)

My contribution to the project was to deliver battery power to the microcontrollers and locking mechanism and to design a system to recharge the batteries. I calculated the power consumption of the microcontrollers and locking mechanism, and from that calculated the number of batteries needed to supply the system power for a couple of days before needing to be recharged. To create a design using rechargeable batteries, a boost converter was needed to boost the voltage of the batteries from 3.7V to 5V. The boost converter design includes a 555 timer,
which outputs to a gate driver, which then outputs to the boost converter. A breadboard design was created and tested, and then a printed circuit board was designed, tested, and soldered. Additionally, the lithium-ion batteries used in the design are recharged using a battery recharging circuit, which was designed and tested on a prototype board, and implemented on a final PCB design. These boards are mounted in a housing that attaches to the door. I worked with Will on the design and troubleshooting of the battery recharging circuit. I also worked with Will on the implementation of housing the electronic components for the final design, as well as connecting power to the solenoid locking mechanism. I worked with Jacob on ensuring that the correct voltage and power would be supplied to the microcontrollers, as well as the physical connection of the power to the microcontrollers.

Andrew Shetler (Honors)

My contribution to the project is to design and implement a mobile application and database. The database will act as an intermediary for communication between the dog door and the mobile application. That is, the dog door will store information in the database to be accessed by the application and the application will store the information in the database to be accessed by the dog door. The database will be an SQL database. Specifically, an Azure SQL database will be used to store data between the mobile application and the dog door. Images saved will be stored in Azure Blob storage with links to the images stored in the SQL database. This design will allow the information in database can be accessed from anywhere. The mobile application will be used to view information in the database stored from the dog door, such as information relating to pet activity through the dog door. The application will also allow the user to configure locking restrictions for the pet door. The mobile application is broken up into the client side and backend. The client side will be developed with the .NET MAUI framework, which uses C# and
XAML. The backend will be a Web API that will be developed using the ASP.NET framework as the core and the Entity Framework for communication with the database. The Web API will be hosted on Azure. The intended operating system for the mobile application will be Android.

Jacob Stump

My contribution is to act as a computer engineer and Project Manager for this senior design project. For my subsystem, I am working on embedded firmware and communication with the application. Our microcontroller, the ESP32-CAM, utilizes protocols with the locking mechanism and the camera to display and update what the viewer sees in the mobile application. It is my responsibility to process all that information given to the microcontroller and send this information to an API that will show this information in an easy-to-read manner for the application to see. I have designed much of the data acquiring protocols, and I have written a code that connects to our API using the HTTP protocol. As the project manager, I maintain the project timeline, order parts when needed, schedule meetings with our Faculty Advisor, and oversee the budget.
Smart Dog Door

Project Design Report

Design Team Member 1: William Boissoneault

Design Team Member 2: Benjamin Charlson

Design Team Member 3: Andrew Shetler

Design Team Member 4: Jacob Stump

Dr. Osama Alkhateeb

Date Submitted: 12/03/2023
0. Abstract

With modern life getting busier and more complex, many have turned to autonomous assistance for taking care of pets. The objective of this project is to design and prototype a device and application that will allow only certain dogs to enter and exit through a dog door while making the owners aware of their dog's activity. The dog door will utilize wireless communication from the dog to the door to control the status of the door’s lock and update the database that will send updates to the application. The application will keep track of when the dog enters and exits the door and present a photo of the dog entering and exiting the door. The locking system will keep the door securely shut unless a pet is entering or exiting, which will ensure that unwanted animals will not enter the home.

The key features are:

- RFID collar detection
- A working mobile application
- Communication with mobile application through server
- Configurable operation of product
- Image confirmation sent from the door to the server and then to the application

1. Problem Statement

1.1 Need

According to the American Veterinary Medical Association (AVMA), 45 percent of U.S. households own a dog (Larkin, 2021). Many times, throughout the day, dogs want to go outside to do their business, play, or explore. Often, this process involves having someone physically take
their dog outside themselves or using a dog door; however, both methods have their drawbacks. Someone taking their dog out pulls them away from what they are doing. A dog using a dog door leaves an opening for unwanted creatures to enter the home and an inability to see what the dog is doing. Therefore, a device, which would allow only certain dogs to enter and exit a house and would allow pet owners to view the dog’s activity, would allow dogs to have freedom to do as they wish while saving their owner the hassle of having to take the dogs out.

1.2 Objective

The objective of this project is to design and prototype a device and application that will allow only certain dogs to enter and exit through a dog door while making the owners aware of their dog’s activity. The dog door will utilize wireless communication from the dog to the door to control the status of the door’s lock. The application will keep track of when the dog enters and exits the door, show a photo of the dog entering and exiting the door, and record photo for future viewing.

1.3 Background

The premise of the project is to create a smart dog door and an accompanying application that will allow a pet owner to remotely control the dog door and view the dog’s activity. The basic theory behind the dog door involves automatic identification technology, an electrically actuated locking system, remote application-based controls of the dog door, and image sharing.
The dog door will be able to recognize several tags to accommodate more than one pet. To detect a specific dog and control the locking status of the dog door, automatic identification technology will be built into the door. This technology can distinguish between different pets based on a unique signature specific to each dog in the system. Radio Frequency Identification (RFID) is a standard in the industry. According to David Benedetti and Gaia Maselli, “RFID is considered a key technology for identification of many objects in several smart environments” (Benedetti, 2022, p. 1). An RFID tag uses radio waves to transmit a unique signal that a system can use to identify a specific user (Benedetti, 2022). In summary, RFID is a widely used concept that has limitless potential in its applications.

To autonomously lock and unlock a door based on a dog’s unique signature, an electromechanical system of some sort is required. Magnetic door stripes—which are commonly used in industrial lock-out doors—and simple solenoid-spring systems are two ways of doing this. For our application’s needs, a solenoid and spring-loaded system seems ideal as it reduces the amount of power required for locking and unlocking the door. If the system is to be battery operated, power efficiency is imperative.

Pull type solenoids are, simply, magnet wire coils surrounding a ferromagnetic arm (Burkowski, 2019, section 2). By utilizing this physical phenomenon, a spring-loaded pull solenoid could be implemented to create a normally locked system that unlocks if current flows through the solenoid’s magnet wire. Thus, the average power usage would decrease as power only needs to be delivered in short intervals.
A mobile application will be created to control various features of the dog door. The application will have features, such as locking the door so it cannot be used by a dog, controlling the time of day the dog is allowed passage through the dog door, and a visual depiction of the dog. Image processing and sending capabilities will be accomplished by the door. The ability to view images sent from the door will be built into the application. The images will then be sent to a server to be stored. On the data streaming side of things, the application connects to a server, and creates a request for accessing the server. When the server accepts the application’s request, the server directs the application to the image files which can then be sent back to the application. According to Dr. Shiddiqi, Dr. Pratama, and Dr. Ciptaningtyas of the Department of Informatics Faculty of Information Technology at Sepuluh Nopember Institute of Technology, “[w]hen the client request to server, the server will direct its response by sending streams of video files in sequence, and the client can immediately play the video stream.” (Shiddiqi, 2010, p.294). As a result, streaming videos is successfully achieved.

Smart dog doors are an industry being explored and implemented by a variety of companies. Many of the current smart dog doors use wireless short-range communication to control the dog’s passage through the dog door. RFID is the main method of controlling a dog’s access to pass the door. Researchers at Mapua University in the Philippines created a smart pet care mobile application where one of the features included was a smart dog door. The researchers stated, “Animal identification using biometrics has been the traditional use radio frequency identification (RFID) in major research issue animal management.” (Luayon, 2019, p. 428) To control the locking system, they used a sensor attached to the pet’s collar. The locking mechanism
is controlled by a servo motor. US Patent 8539715B2 also talks about using RFID to control the pet door lock. This patent specially references “reading a sub-dermal RFID implant” (Hill, 2013).

AS

The smart pet care mobile application also can control the pet door’s lock and give a report of how many times a pet has entered and exited the pet door. The mobile application could also view a live feed of the pet from a camera that was separate from the dog door.

AS

Another pet monitoring system was made by researchers at St. John’s University, Taiwan in 2011. Their pet door had a selective latch to enable or disable a pet’s passage through the door, an animal detector to detect a dog seeking passage through the door, motion sensors to detect if the dog moved through the door, a clock that allows the setting of a time schedule or set up of predefined pass-through conditions for the pets, and environmental sensors to monitor the weather and allow a pet owner to control the pet door’s lock based on outside weather conditions (Own, 2011, p.473).

AS

Many limitations from the patents shown above involve power. To have a system that automatically checks for RFID tags and autonomously unlocks a door, a constant power draw is required. As a result, rechargeable and replaceable batteries can be common for systems like these. It would also be just as inconvenient or more challenging to have a wall connection to these devices.

WB and JS
Another limitation from some patents involves RFID. For an RFID signal to be successfully transmitted, the tag must be in reader range. If the tag is too far away from the scanner, the signal is too weak for the RFID reader to detect the tag, the reader cannot do anything until it receives a stronger signal from the tag. Another issue with using RFID technology is data collision. According to Dr. Qian Zhihong, a professor of Communications and Information Systems at Jilin University, and Dr. Wang Xue, a lecturer in the department of communication engineering at Jilin University, “In the RFID systems, data collision is inevitable when the reader sends a communication request, and multiple tags respond with simultaneous data transmission. Data collision is prone to causing problems such as: identification delay, spectrum resource waste, a decreased system throughput rate, etc.” (Zhihong, 2014, p. 44).

In conclusion, RFID can be beneficial in its use, but there are limitations, such as signal strength and data collision, that must be overcome to maximize its capabilities.

Some automatic animal doors on the market use an electrical motor to open the system. This poses the problem of the motor and its associated components deteriorating over time. Higher end motors would reduce the wear and tear over time, but these would be difficult to sell because pet owners would be unlikely to buy a more expensive system.

There is a plethora of different autonomous pet related systems, with many commonalities and many differences. A common trait of these systems is that a battery is used instead of a direct connection to a power receptacle. In kind, it is likely this proposed project will have a battery as its source of power. As pet owners know, animals can be prone to chewing on electrical cables.
Therefore, it would be safe and proper to have a rechargeable system rather than a direct connection to an AC outlet as owners can be present and supervise during charging cycles. Also, a wired connection could present itself as a tripping hazard.

WB

Another widespread practice in these devices is using RFID for identifying a pet. This proposed system would also have this commonality with other pieces of equipment already on the market. By giving a pet a collar that has a unique RFID tag, it makes it simple for the system to recognize that a pet is present and unlock the door in response.

While there is much crossover for pet doors, this proposed system will undoubtably have differing features. Chief among them is that this system will have video streaming capabilities so that a pet’s owner can supervise their animal without being physically present. The camera for the video streaming will be built into the door instead of using a separate camera system. This was accomplished by researchers at Mapua University with their mobile application (Luayon, 2019). With this feature, an owner will have the peace of mind knowing that their pet is safe.

AS and WB and BC

One widespread practice in this market is to use mechanical motors to open and close the pet door. This proposed system, however, will use an electromechanical locking system. This would limit wear of the system, limit power consumption, and free up power for more demanding systems, such as video streaming capabilities or RFID scanning.

WB and BC
Patent US9157269 was accepted on October 13, 2015, and the inventor is Richard Brown. This patent describes and depicts a programmable and smartphone-controlled pet door. One interesting aspect of the device described in the patent is the ability for an owner to program the door to open only at certain times or for a certain number of times before locking out completely. Not only this, but many varied sizes of this application are mentioned, such as doors designed for use at a zoo.

WB

Patent US8302348 was accepted on November 6, 2012, and the inventor is Crosby S. Noyes. This patent is for a sliding pet door that can easily be installed and only allows specified pets through the door. The patent was accepted on November 6, 2012, by Crosby S. Noyes. Pets are identified using radio frequency identification (RFID). There is a device on the casing of the door that can be configured to regulate how a pet can travel through the door. Multiple pets can be registered, and certain pets can be allowed according to the conditions set. This is done by restricting individual pets to travelling only in or out of the door and restricting what times the pet door can be unlocked. There is also a platform and ramp for the pet to stand on if the door is mounted above floor level. The pet door is opened and closed using gears and motors that move a sliding door up and down or left and right, depending on how the pet door is mounted. There is also a sensor that detects the current position of the pet door.

BC

1.4 Marking Requirements (JS)

1. This system should allow its user to monitor its registered animal’s status remotely.

2. This system should control a registered animal’s ability to go through the door.
3. This system should set different modes that allow automatic lock/unlocking.

4. This system should allow multiple animals to be registered into the system.

5. This system should prevent unregistered animals from getting into a house.

2 Engineering Analysis

2.1 Server and Application (AS)

The theory of operation for the server is to act as an intermediary between the application and dog door. The server will store information from the dog door to be accessed by the application and the application to be accessed by the dog door. The server is broken up into two parts one is a database, and one is storage for the images used in the system. The data needed to be stored by the database to satisfy the marketing requirements can be broken up into three distinct categories. The categories are pet personal information, pet activity information, and door locking restriction information. These three categories handle all information, besides images, needed by the application and dog door.

The pet personal information category handles information directly associated with a single pet. The first part of information this category needs to store is each pet’s id. The pet’s id is from the UID from RFID tag assigned to the given pet. The next information needed for this category is a pet name and pet image link. This information will be input from the application and stored in the database to personalize each pet’s profile in the application. The image itself is stored in dedicated image storage while a link to the image is stored is in the database. Lastly, the information on whether the pet was last detected inside or outside the door needs to be stored to easily display in the mobile application whether a pet is inside or outside.
The pet activity information category holds information on each time a pet went through the dog door. The first part of information is the pet ID of the pet that went through the door, which is needed to mark what pet went through the door. The second part of the information is whether the pet went from inside to outside or inside to outside. Next, a timestamp of when the dog went through the door is needed to inform the pet owner of what time the dog accessed the door. Lastly, a link to an image of the pet, where the image file is stored in the server in dedicated image storage, going through the door is needed to show the pet owner in the application the dog’s activity.

The locking restriction information category holds information on the restrictions on access to the dog door the pet owner set in place from the application. The pet owner from the application can set certain times of day the pet door can be used and whether the dog is always locked, unlocked, or operating normally. To facilitate this functionality the database will need to hold information on the time the locking restriction starts, the time the locking restriction stops, and the door is locked or unlocked between the locking restriction start and stop time.

Since the information in the database is broken up into three relational categories, a relational database can hold the data. The images will be stored in Blob storage. Blob storage is storage that is aimed at large unstructured data, which in our case is the pet activity images and pet profile images. The server also needs to be accessible to the application from anywhere. Since the pet owner will sometimes not be home when the dog door is used, the dog door needs to be able to communicate with the server and the application to the server from anywhere. The requirement of the server needing to be accessible from anywhere lends itself to the use of a cloud server. Cloud servers are hosted on the internet, which provides
access from anywhere. An Azure server will be used for this project. The database used for this project is the Azure SQL Database project. Each of the three categories of information will be stored in a different database table in the Azure SQL database. The blob storage for this project will be Azure Blob Storage.

The purpose of the application is to let the owner view the dog door's activity and set locking restrictions for it. The application will be a mobile application because a mobile application more easily allows a pet owner to view their pets’ activities from anywhere. The application will be made up of a client side and a backend. The client side will be what the pet owner will interact with. It will display all the information on their pets and the dog door from the cloud server, and it will allow the pet owner to input the pet personal information and locking restrictions. The backend of the application will communicate with the database to get information to display and set information in the database. The application will also notify the pet owner of new pet activity in the database through notifications.

The mobile application will need to achieve seven distinctive features to meet the marketing and engineering requirements. The first requirement is that the application will allow the user to request to add a new pet into the database. This feature will be achieved by the app adding a blank row into the pet information database table that can be populated when a new RFID tag is detected. The second feature is that the application will allow for deletion of pets in the database. Upon request the application will delete all information from the database and Blob storage regarding the requested pet. The third feature is the application will display the pet activity. This activity will include the times a pet went through the door and an image associated with each of these various times.
The fourth feature is the application will display whether the pet is currently outside or inside based on the latest activity for the selected pet. The fifth feature is the pet owner can set locking restrictions for the dog door. These restrictions include certain times when the pet door can or cannot be unlocked, and whether the pet door is always locked or unlocked. The sixth feature is that the pet owner can update personal information on each pet in the application. The personal information includes a name for each pet and an image for each pet. The seventh feature is that the application will notify the pet owner of recent activity of the dog door.

To easily accommodate the access to the server and to host the backend of the application, Azure App Services will be used. Azure app services will enable the application to be able to connect to the database whenever the user is at as long as they have an internet connection. The backend will be ASP.NET Core Web API will use Entity Framework for facilitating the exchange of information with the database. For the backend, the C# programming language will be used. The client side of the application will use the .NET MAUI framework that utilizes the C# programming language and XAML. The application will use the MVVM software architecture. This architecture was selected because it will easily allow the pages of the application to dynamically change based on the information in the database. MVVM also allows for clear separation between application logic and the application UI, which allows for the application to more easily be tested and maintained (Stonis 2022). The mobile application will be developed using Visual Studio as it has tools built in to facilitate development. The target operating system will be Android.
2.2 Power System (BC)

The power system must supply both the proper voltage and the proper current to the locking system and the microprocessor. In addition, the system will use battery power so that the dog door will not need to be plugged into an outlet. The battery power supply will need to be rechargeable so that the user does not have to constantly change out the batteries.

Both the locking system and the microprocessor require 5V. Due to the selection of batteries on the market, a 5V rechargeable battery was not available. Instead, 3.7V batteries will be used, and that voltage will be stepped up to 5V using a boost converter, which is an efficient way to step up voltages. The efficiency of the boost converter was found to be 81% when calculating efficiency using equation 1.

\[
n = \frac{P_{out}}{P_{in}} \cdot 100
\]

Pout is the output power of the converter, Pin is the input power of the converter, and n is the efficiency of the converter.

The battery power supply will need to last long enough to last for at least 2 days in case the owner is not home for a night or two. Thus, multiple batteries in parallel will be used to increase the total amp hours produced by the power supply. Each battery provides 2.6Ah of current.

The power supply must power the microprocessor and the solenoid. The solenoid takes 5V and uses 11W of power, so it pulls 2.2A when active, which can be calculated using equation 2, where P is power, I is current, and V is voltage.

\[
P = IV
\]
The solenoid is normally off and only turns on when opening the dog door, so ideally it would be off most of the day, unless the user sets the door to be on for a given time. Throughout the entire day, the door will not be open for more than an hour, which would amount to 2.2Ah. The ESP32-CAM has different modes, which are deep sleep, awake without flash, and awake with flash, and have different current draws in each mode. For a given day, the ESP32 will often be in deep sleep and awake without flash. Table 1 shows how many amp hours each mode will take in each day, if the ESP32 is in that mode for the given amount of time. The amp hours can be calculated by multiplying the current by the number of hours in that mode, as shown in equation 3. \( I \) is current, \( t \) is the time in hours.

\[
[\text{Ah}] = I \cdot t
\]

(3)

Table 1: ESP32-CAM Current draw and amp hours based on duration in each mode.

<table>
<thead>
<tr>
<th>ESP32-CAM</th>
<th>Deep Sleep</th>
<th>Without Flash</th>
<th>With Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>6 mA</td>
<td>180 mA</td>
<td>310 mA</td>
</tr>
<tr>
<td>Hours</td>
<td>12</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Amp hours</td>
<td>72 mAh</td>
<td>1980 mAh</td>
<td>310 mAh</td>
</tr>
<tr>
<td>Total</td>
<td>2362 mAh</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus, both ESP32s together would take 4.724 Ah. Combined with the solenoid, this adds to 6.924 Ah for a single day.

The input current in amp hours can be calculated by equation 4. \( I_{in} \) is the input current in amp hours, \( V_{out} \) is the output voltage of the boost converter, \( I_{out} \) is the output current to the boost converter, \( V_{in} \) is the input voltage, and \( n \) is the efficiency of the boost converter.

\[
I_{in} = \frac{V_{out} \cdot I_{out}}{V_{in} \cdot n}
\]

(4)
Since the boost converter has 81% efficiency, an input voltage of 3.7V, an output voltage of 5V, and an output current of 6.924 A, the input amp hours needed would be 11.55 Ah for one day.

Then, the number of batteries for a single day can be calculated using equation 5, where $B$ is the number of batteries needed for a single day of power, $I_{in}$ is the input current in amp hours needed (found in equation 4), and $I_{battery}$ is the current in amp hours that a single battery supplies, which is 2.6Ah.

$$B = \frac{I_{in}}{I_{battery}}$$ (5)

Thus, approximately 4.4 batteries would be needed to provide a single day of power. For two days, this amounts to 9 batteries at a minimum. Therefore, using 10 batteries for the system is sufficient.

For the boost converter to output 5V, the gate of the MOSFET needs to be driven with the proper frequency and duty cycle, as determined by the capacitors and resistors used in the 555-timer circuit. The duty cycle needed is determined by equation 6, where $D$ is the desired duty cycle, $V_{in}$ is the input voltage of the boost converter, and $V_{out}$ is the output voltage of the boost converter.

$$D = (1 - \frac{V_{in}}{V_{out}}) \cdot 100$$ (6)

Given that the input voltage is 3.7V and the output voltage is 5V, the desired duty cycle is 26%. 
2.3 Locking system (WB)

The locking module is required to be reliable and energy efficient. If the door is not able to reliably lock itself again or drains the battery array rapidly, the entire point of the project is defeated. However, the first hurdle to overcome is power efficiency.

It is imperative that no matter what voltage the solenoid selected uses, it should pull a reasonable amount of power infrequently. As it stands, a 5V or 12V pull-solenoid pulls similar amounts of power (13 and 11 respectively) but the major difference between the two is the amount of current draw. With the 5V solenoid pulling a bit over 2 amps of current, this should be feasible as the system should, ideally, only draw current infrequently.

Another consideration for the solenoid was not just its power consumption/current draw, but also the amount of continuous usage the device can handle. Through research, it was found that there are varying types of solenoids that can only be actuated for different amounts of time. While it is not necessary to have a solenoid that can continuously operate, the pricing difference is negligible.

Finally, to ensure reliable re-latching/locking, a permanent magnet on the door and a reed switch control loop to the processor can be implemented. Presently, the design of the dog door has permanent magnets on the frame with metal attachments on the door to reduce the amount of swaying after a dog has already entered or exited. This should speed up the door closing, thus reducing the amount of time the solenoid consumes power.
2.4 Microprocessor (JS)

The microprocessor is split up into three main parts: the microprocessor, the radio frequency module, and the database module. To begin, the microprocessor that will be used is an ESP32-CAM. Originally, the team wanted to use the PIC24 series. In terms of communication protocols, the PIC24 would have been a perfect choice for us. However, it became apparent quickly that it would be incredibly difficult to be able to store an image on it and send it into a database. The normal flash memory for the PIC24 is around 50Kb. The average digital image is about 500Kb making it incredibly difficult to store an image. Ideally, we want to store an image in the first place because in the instance that our image does not get sent correctly or at all into the database, we want to be able to still have that image to retransmit until it is successful. From there, a Raspberry Pi was used as well. However, a huge drawback to using a Raspberry Pi is that it needs anywhere from 5W to 12.5W of power which is enormous. So, the discovery of the ESP32-CAM was the breakthrough that was needed.

The ESP32-CAM has everything that is needed to successfully complete the project. The board has the capabilities to communicate with the database using Wi-Fi. The company even advertises that the ESP32-CAM can upload pictures over Wi-Fi. There is 32 Mb SPI default flash memory which is more than enough for the microprocessor to store and send an image to the database. There are also both UART and I2C communication protocols that allow for the microprocessor to communicate with the RFID module. Furthermore, there will be at most about 0.5mA pulled to be used. With the standard 5V being pulled, about 2.5W will be pulled per microprocessor which is significantly lower than the Raspberry Pi. Lastly, the microprocessor is also incredibly cheap sitting at $8.92 per board including the
microprocessor. As a result, the ESP32-CAM provides the project with the best of both worlds in terms of communication with adjacent modules and power consumption.

The next part of the subsystem involved the radio frequency module. For this project, an RFID tag and reader were chosen as our communication between the pet and the doggie door. For the RFID tag we will be using a 28161-ND tag. It has a communication range of about 1 meter which will go perfectly to meet our engineering requirements. The RFID reader used is the Pepper C1-000394. It will communicate with the microprocessor using the UART communication protocol. To power this module, a 5V output from the ESP32-CAM will be used. Attached to the RFID reader is the RFID NFC Flex ANT1356 40×40-100. This will serve as the antenna. While there is a built-in antenna to the RFID reader, the antenna will be able to receive a signal from further away than what the RFID reader allows. The RFID tag, RFID reader, and RFID antenna will operate at 13.56MHz, which is a frequency that will let us get the needed extended range.

The Pepper C1 has a bit of functionality that will also be useful in completing this project. For starters, the Pepper C1 can whitelist RFID UIDs. By doing this, it can help with preventing unregistered RFID signals from entering the database. This will allow for a more secure system overall. To continue, the Pepper C1 can transmit information to a server. That will be set up to speed up the time it takes to update and store information in the database. Finally, the Pepper C1 has an LED that can change colors depending on if the RFID UID is whitelisted. Those colors can be changed based on an RGB value and can be utilized to show if a tag is registered or not when we build the dog door.

The final part of this subsystem is the transferring of data between the microprocessor and the database. Ideally, the microprocessor will send the needed
information to the blob storage over Wi-Fi. To make sure the image is sent successfully, the microprocessor will store the image taken temporarily as a precaution to make sure no data is lost when transmitting. To store the photo onto the database, we will attempt to send the photo and store it in blob storage on the Azure server that Drew is setting up. The unique ID, UID, from the RFID reader will be sent to the Azure database for storage and eventually be sent over to the application.

By using the parts listed above for this subsystem, it is both doable and plausible to not only successfully communicate within the subsystem, but to every other subsystem as well.

### 3 Engineering Requirements Specification

<table>
<thead>
<tr>
<th>Marketing Requirements</th>
<th>Engineering Requirements</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>The dog door should open within 3 seconds to allow a pet through to the door.</td>
<td>3 seconds should be adequate time for the system to send and receive a signal to open the door</td>
</tr>
<tr>
<td>1, 2, 3, 4, 5</td>
<td>A database will be used to store information, pet ID, pet name, in/out status, and pet image, on registered pets, information on pet activity, pet ID, in/out, time, and image, and store information on locking registration, times of restricted door access.</td>
<td>This is important for consistent and proper performance from the door. This will allow for debugging and configurability.</td>
</tr>
<tr>
<td>2, 5</td>
<td>The locking mechanism should be strong enough to prevent a 30lbs unwanted animal from going through the door.</td>
<td>30lbs is about the weight of a coyote and greater than the average weight of the raccoon, skunk, and other unwanted pets.</td>
</tr>
<tr>
<td>1, 4</td>
<td>A remote application will be used to monitor the status of registered pets, which includes whether the pet is inside or</td>
<td>A remote application will allow the pet owner to view the up-to-date pet activity anywhere</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>outside and an image of the pet passing through the door.</td>
<td>remotely with a device with internet access</td>
</tr>
<tr>
<td>2, 3</td>
<td>A remote application will be used to control whether the door is always locked or unlocked, as well as certain times of day that the door is restricted.</td>
<td>A remote-control application will allow the user to specify when their pets can and cannot use the door.</td>
</tr>
<tr>
<td>2, 4, 5</td>
<td>A remote application will be used to initiate pairing new pets through communication with the pet door via the database.</td>
<td>A pairing application allows for multiple pets to be registered for communication with the door.</td>
</tr>
<tr>
<td>4, 5</td>
<td>A remote application will be used to delete unwanted pets in the database.</td>
<td>Pets already in the system that no longer need access must be able to be deleted.</td>
</tr>
<tr>
<td>1</td>
<td>A remote application will allow the user to input their pet names and an image for each pet to be stored in the database.</td>
<td>The addition of a pet name and pet image will more easily allow the user of the application to distinguish between their pets.</td>
</tr>
<tr>
<td>1</td>
<td>A remote application will notify the user through push notifications within 1 minute of the pet passing through the door.</td>
<td>1 minute will allow enough time for the database to be updated will the needed information while still notifying the user close to the event.</td>
</tr>
<tr>
<td>1</td>
<td>The remote application should check for updated pet activity information in the database every 30 seconds.</td>
<td>The 30 seconds will allow the user to be notified and view new pet activity close to the event.</td>
</tr>
<tr>
<td>2, 5</td>
<td>Wireless communication between a module on the dog door and a module on the pet will be used to get an ID from the pet to unlock the dog door. The pet should be detected if within 3 feet.</td>
<td>Communication between the dog and door modules allows for the door to unlock only when a dog with a registered module is near the door.</td>
</tr>
<tr>
<td>2, 3</td>
<td>A spring-loaded solenoid peg will unlock in the presence of a registered pet. This module will also have a sensor that detects if the door is in place for locking. Potentially, a cover</td>
<td>This system needs to be power efficient and reliable, so a control loop that can detect if the door is in place to be locked is required as well as a spring to passively keep the door locked.</td>
</tr>
</tbody>
</table>
could be placed over the peg to keep the door unlocked without constantly drawing electrical power.

<table>
<thead>
<tr>
<th>2</th>
<th>A power delivery system is required on the door to meet all its requirements. This system should be around 5VDC to provide enough power for the solenoid locking module and the processor as well. Additionally, a boost converter is required to step up the main voltage to the appropriate level. The system should not rely upon a connection to a power outlet for regular operation.</th>
<th>This power delivery system needs to be efficient and powerful enough for the electromechanical and processing needs of this application. Therefore, a boost converter with streamlined efficiency is imperative.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>The door will wait for 3 seconds before opening</td>
<td>The door should not open automatically when the tag comes in range in case the dog does not want to go out.</td>
</tr>
<tr>
<td>2</td>
<td>The dog door will relock within 30 seconds if the pet does not go through the door.</td>
<td>30 seconds will provide ample time for the dog to go through the door while minimizing the power drawn from the locking mechanism.</td>
</tr>
<tr>
<td>2</td>
<td>The system will take less than 30W of power for an hour of use</td>
<td>Since the system runs off 5V, this would mean the max current would be 6A. The main 2 units drawing power from the power supply are the processor and the solenoid, both of which should take less than or equal to 3A. Any more than this would make it difficult to power the system for a long time.</td>
</tr>
</tbody>
</table>
4 Engineering Standards Specification

4.1 Safety (WB & BC)

When selecting the type of battery to be used, there were two options that stood out from the rest and for vastly varied reasons: lithium-ion and lead acid batteries. Lead acid batteries were economical with less than desirable power storage and long-term health. Lithium-ion batteries were the exact opposite in that they were pricey in comparison with longer operating lifespans and denser energy storage. Both would fare well enough, but for safety reasons we decided to work with low voltage arrays of lithium-ion batteries as lead acid batteries require ventilation to prevent harm to users or designers and pose other complications, safety and otherwise.

As it stands, the reed switch and magnet combination should, in theory, prevent any sorts of physical injuries (pinches, pokes, etc.) from the solenoid locking system. Since the door should only be locked or unlocked when in the correct position as determined by the reed switch, the area should be clear for safe operation.

4.2 Communication (JS)

To communicate with the RFID module, the ESP32-CAM will use the UART protocol. Similarly, a master/slave configuration will be used between two ESP32-CAMs to allow for a singular microprocessor to communicate with the database.

To communicate with the database, the ESP32-CAM will use Wi-Fi for everything. The images taken from the camera and the data from the RFID module will be sent to the database over Wi-Fi.
4.3 Data Formats (JS & AS)

The data format of the images in the server’s blob storage will be stored as JPEG images. The data in the database will either be stored as strings or primitive data types, such as integers.

The microprocessor will be sending out the photo as a JPEG that will then be stored in blob storage.

4.4 Design Methods (AS & JS)

The application will be designed using Visual Studio and an Android emulator. The targeted operating system for the application is Android. The backend of the application be designed in Visual Studio and will be hosted using Azure App services. The database will be designed using SQL Server Management Studio. The Database will be an Azure SQL database. Images will be in the Server will be stored in Azure Blob Storage.

The protocol used for sending the data from the microprocessor and the database is MQTT. MQTT is an Internet of Things protocol that sends/publishes messages to a server. In this instance, the messages sent are packets of data. There are also ways to retransmit data and confirm that transmission sent is successful. Another viable way of accomplishing this is using web socketing. Web socketing allows for a two-way communication between the RFID reader and the database on the Azure server by using TCP.

4.5 Programming Languages (JS & AS)

For the mobile application client side, the programming languages being used are C# and XAML using the .NET MAUI framework. For the application backend an ASP.NET
core framework will be used with the Entity Framework for accessing the database. Both these frameworks also use C#. The database will be an SQL database, so it will use the SQL language.

The ESP32-CAM will be using the Platform.IO IDE. Platform.IO allows the implementation of the C and Arduino languages, and I have found it works very well with any ESP32 development board. C will also be used to get the data from the RFID module to be able to be sent into the database.

4.6 Connector Standards (WB & BC)

To reduce the chance of short circuiting and to be as touch safe as possible, Wago style wire splice connectors shall be used for proper termination. Additionally, proper soldering of any pins shall be imperative to keep the system operating safely. Some modules for battery charging may also use USB-C for sourcing.
5 Accepted Technical Design

5.1 Hardware Design

![Level 1 Block Diagram](image)

*Figure 5.1.1 Holistic System View*

The holistic system block diagram shows how the systems connect to each other. The battery charger has power coming from an outlet to charge batteries. These batteries are then used to as the input to the power supply, which outputs 3.7V to a DC-DC converter. The DC-DC converter outputs 5V, which is used to power the locking module and the microprocessor. The microprocessor sends a signal to the locking module to tell the locking module when to unlock the door, which is based on the reading from the reed switch. The microprocessor also powers and sends signals to the cameras and RFID...
antennas. The RFID antennas then interact with the RFID tags, and Wi-Fi is used to connect to the database, which then in turn connects with the application.

**Figure 5.1.2 Level N Locking Module Decomposition**

<table>
<thead>
<tr>
<th>Module</th>
<th>Locking Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>William Boissoneault</td>
</tr>
<tr>
<td>Inputs</td>
<td>Microprocessor Actuation/Handshake, 5VDC sourcing</td>
</tr>
</tbody>
</table>
Outputs

| Magnetic actuation from permanent magnet to reed switch |

Description

A pull solenoid will keep the door locked unless the microprocessor actuates a MOSFET switch to unlock. A permanent magnet and reed switch combo are used to let the microprocessor know when to lock the module once again.

Level 2- DC to DC Converter

![Diagram of DC to DC Converter]

*Figure 5.1.3 Power Module Boost Converter*

<table>
<thead>
<tr>
<th>Designer</th>
<th>Benjamin Charlson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>3.7 V from power supply</td>
</tr>
<tr>
<td>Outputs</td>
<td>5 Volts to microprocessor and locking module</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>Uses a boost converter to efficiently boost the voltage from 3.7 V to 5 V</td>
</tr>
</tbody>
</table>

The full schematic for the power supply is shown above. The individual sections of the schematic are shown in the following pages. The power supply is comprised of three main components: the 555-timer, the gate driver, and the boost converter. The input voltage is 3.7V, which is supplied by lithium-ion batteries connected in parallel. The 555-timer generates a pulse width modulation (pwm) signal, which is sent to the gate driver. The gate driver helps stabilize the voltage and current from the 555-timer and sends this stabilized output to the gate of the MOSFET in the boost converter. The boost converter then amplifies the 3.7V input to 5V.
The first part of the schematic contains the 555 timer. The 555 timer outputs a pwm signal with a frequency based on the capacitance of C1, and a duty cycle based on the ratio of R2/R1. In the physical implementation of the circuit, the 10 kΩ potentiometer should be tuned such that the output voltage is exactly 5V, as physical components have a tolerance in their measured values. The OUT pin of the 555 timer then connects to the input pin of the gate driver in the next section of the schematic.
The AC output voltage of the 555 timer is shown in Figures 5.1.6 and 5.1.7. In Figure 5.1.6, multiple pulses are shown, and each pulse has a spike in voltage at its start. The oscilloscope is zoomed in in Figure 5.1.7 to show the spike more clearly in voltage. This spike is caused by an overshoot in the switching of the signal from low to high.
The next part of the power schematic involves the gate driver. The gate driver takes the output of the 555-timer as an input to pin IN1 and smooths out the voltage signal. The output of the gate driver is GT1, which connects to the gate of the MOSFET, which is part of the boost converter (in the next part of the schematic). Also, a low pass filter at the gate driver's output is used to stabilize the voltage, comprising R3 and C4 in Figure 5.1.8.
Figures 5.1.9 and 5.1.10 show the AC output of the gate driver. The voltage output does not have an overshoot voltage when switching from low to high as the 555-timer output does. The oscilloscope is zoomed in for Figure 5.1.10, showing that the voltage smoothly transitions from low to high. Despite the use of the gate driver, there is still a ringing in the voltage between the pulses, but this signal is still an improvement from the 555-timer output signal.
The final part of the boost converter design is the boost converter. The output of the gate driver is sent to the gate of the MOSFET in the boost converter. The frequency at which the MOSFET switches and the duty ratio used to switch the MOSFET help determine the resulting output voltage. The boost converter steps up the input voltage by storing energy in an inductor, which is then transferred to the capacitor at the output. The diode D2 will only turn on when the MOSFET is off, and when the MOSFET is on, current will not flow through the diode. In the schematic, the output is a 100Ω resistor, since the boost converter needs a load, but once the subsystems combine, the boost converter will output to the microprocessor and the solenoid.
The AC output of the boost converter is shown in Figure 5.1.12. The ringing in the output signal is due to the ringing in the gate driver output. However, the variation in the output voltage is still small enough for the solenoid and the microprocessor to handle.
The BQ24166RGET lithium-ion battery charging IC will be used to recharge the lithium-ion batteries. This part of the design is still in progress, but the typical application circuit for this IC to recharge a single lithium-ion battery is shown below. The datasheet where this circuit can be found is in the Appendix, and the circuit is also shown in Figure 5.1.13.
5.2 Software Design

![RFID Interactions Flowchart](image)

**Figure 5.2.1 RFID Interactions Flowchart**

**Level 1 Block Diagram**

<table>
<thead>
<tr>
<th>Designer</th>
<th>Jacob Stump</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>RFID Tag Signal</td>
</tr>
<tr>
<td>Outputs</td>
<td>Data sent to database over Wi-Fi using MQTT</td>
</tr>
<tr>
<td>Description</td>
<td>Embedded System Hardware Block Diagram. This system demonstrates the brain of the hardware side of the project.</td>
</tr>
</tbody>
</table>
Server Block Diagrams:

The server acts as an intermediary to store data between the application and the dog door. Both the server and the application put data into the server and read data out of the server. For the application to send data to the dog door, the application would put data into the server and the dog door would read that data. The same theory of operation works for sending data from the dog door to the application as well. The interaction between the dog door and the application can be seen below in Figure 5.2.2.

![Figure 5.2.2 Server Level 0 Block Diagram](image)

**Level 0**

<table>
<thead>
<tr>
<th>Module</th>
<th>Sever Level 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
<tr>
<td>Inputs</td>
<td>Data from Dog Door</td>
</tr>
<tr>
<td></td>
<td>Data from Application</td>
</tr>
<tr>
<td>Outputs</td>
<td>Data to Dog Door</td>
</tr>
<tr>
<td></td>
<td>Data to Application</td>
</tr>
<tr>
<td>Description</td>
<td>The server acts as an intermediary between the application and the dog door. Data from the door gets stored in the database to be accessed by the application and vice versa.</td>
</tr>
</tbody>
</table>
The server is made up of two main components, the database and Blob storage. The database holds data in three main categories. The categories are Pet Information, Pet Activity Information, and Locking Restriction Information. Each one of these categories has data associated with it and their own table inside the database. Blob storage is used to hold images to be accessed by the system. There are two image categories, which are Pet Profile Images and Pet Activity Images. The details of the server and its components can be seen in Figure 5.2.3.

**Figure 5.2.3 Server Level 1 Block Diagram**

<table>
<thead>
<tr>
<th>Module</th>
<th>Server Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
</tbody>
</table>
| Inputs | - Application User set locking restriction data to Database: Times of day door is inaccessible to pets, and master lock/unlock data.  
- Dog Door Pet Activity Data to Database: Pet ID, In/Out, Image, Timestamp of pet that accessed the door. |
| Application Pet Details to Database: Pet Name, and Profile Image of Pet |
| Application Pet Profile Images to Blob Storage |
| Dog Door Activity Images to Blob Storage |

**Outputs**

- Database to Dog Door User set locking restriction data: Times of day door is inaccessible to pets, and master lock/unlock data to dog door to decide if it can unlock
- Database to Application Pet Activity Data: Pet ID, In/Out, Image, Timestamp of pet that accessed the door to display in application.
- Database to Application Pet Details: Pet Inside or Outside.
- Database to Dog Door: Pet IDs in system and whether pets are inside or outside.
- Blob Storage to Application: Pet Profile and Pet Activity Images

Application Block Diagrams:

The application is used for a pet owner to view the pet activity, control the dog door, and control the saved pets. The application will receive information from the database and display it to the user. The application will send data to the database to be saved, such as pet personal information and locking restriction information. The input and outputs to the application can be seen in Figure 5.2.4.

![Figure 5.2.4 Application Level 0 Block Diagram](image-url)
The application is broken up into two main components, which are the client side and the back end. The backend handles communication between the app and the server, and the client side handles all interactions between the user and the application. Figure 5.2.5 shows the roles and interactions between the client side and the backend of the application.
**User Input:** Touch Input, Pet image input

**Outputs**
- Data to Database: Data to Pet Information table, Pet Activity Table, Locking Restriction Table
- Notification to Pet Owner on Pet Activity

**Description**
The Application is comprised of two different portions the client side and backend. The backend handles communication between the app and the server, and the client side handles all interactions between the user and the application. The application also lets the pet owner set some different settings on the pet door, such as locking restrictions.

The client side is broken up into 4 pages. The pages allow for the user to interact with the application. The pages are the locking restriction page, the pet activity page, the pet information page, and the individual pet page. The locking restriction page allows the user to view the current locking restrictions in place and add, delete, or edit the locking restrictions. The pet activity page allows the user to view all the pet activity from the dog door. The pet information page allows the user to view all saved pets in the system and add additional pets. The individual pet page allows the user to edit the selected pet’s name and image, delete the pet, and view the selected pet’s activity. Figure 5.2.6 details all of the interaction between each of the pages and the database tables.
<table>
<thead>
<tr>
<th>Module</th>
<th>Application Level 2: Client Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
</tbody>
</table>
| Inputs          | -Previously Saved Locking Restriction Data: Data on set locking restriction information from locking restriction database table via application backend  
|                 | -Pet Activity Data: Data from Pet Activity Database table via application backend  
|                 | -Pet Information Data: Data from Pet Information Database table via Application Backend  
|                 | -User Touch Input  
|                 | -Pet Profile Image input |
| Outputs         | -Locking Restriction Data: Data for Locking Restriction database table sent to application back end  
|                 | -Pet Personal Data: Pet name and pet image data for pet information database table sent to application back end.  
|                 | -Request to Add/Delete Pet: For Deletion, request to application backend to delete all information from database involving pet to be deleted. For Addition, request to application backend to add a new row to Pet Information database table |
| Description     | The client side of the application handles all user input into the application. The client side also presents to the user all data on the pet activity through the dog door, such as images and times. The client side allows the user to input locking restrictions for the dog door as well as edit personal information for each pet, such as pet name and a profile image. |

The back end handles all interaction between the database and the user input from the client side of the application. Figure 5.2.7 details the exchange of information between the client side of the application, and the database.
<table>
<thead>
<tr>
<th>Module</th>
<th>Application Level 2: Back end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
</tbody>
</table>
| Inputs       | - Data from Pet Database: Data from Pet Information table, Pet Activity Table, Locking Restriction Table  
               - Data from Client Side: Data from Pet Information page, and Pet Activity Page |
| Outputs      | - Data to Client Side from Database: Data from Pet Information table, Pet Activity Table, Locking Restriction Table to client side.  
               - Data to Database from Client Side: Data from Pet Information Page to database, with new pet name, pet image link to pet information database table, and request to add/delete pet with addition add new row to pet information table with deletion delete entry from Pet Information table and all entries for pet in Pet Activity table, data from Locking Restriction Page to Locking Restriction database table.  
               - Notification to Pet Owner on Pet Activity |
The backend of the application handles all communication between the client side of the application and the server that stores information between the dog door and the client side. The backend also checks for changes in the Pet Activity database table and if so, the application sends a notification to the pet owner.

Flow Charts:

The flow chart for the normal process of a pet gaining access and going through the dog door is detailed in Figure 5.2.8. The flow chart describes the process from the detection of the pet’s RFID tag to a notification being sent to the pet owner.
Figure 5.2.8 Door Activity Flowchart

<table>
<thead>
<tr>
<th>Module</th>
<th>Flowchart: Normal Dog Door Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Before the pet can be used with the dog door system, it must be registered. The flow chart for adding and registering a new pet to the dog door system is detailed in Figure 5.2.9.
Figure 5.2.9 Adding Pet Behavior Flowchart
<table>
<thead>
<tr>
<th>Module</th>
<th>Flowchart: Adding a New Pet to the System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
<tr>
<td>Inputs</td>
<td>-Request to Add Pet in App</td>
</tr>
<tr>
<td>Outputs</td>
<td>-Pet ID Added in Database</td>
</tr>
<tr>
<td>Description</td>
<td>The flow chart shows the process of adding a new pet into the system. First the request to add a new pet is done in the application. The application will add a new empty row to the pet information database table. If a new tag is detected within a minute of the prompt of the user to hold the new tag to the door, then the pet ID in the new row will be populated, and the user will be notified of the successful addition of the new pet. Otherwise, the newly added row will be deleted from the pet database table.</td>
</tr>
</tbody>
</table>
Removing a Pet from System

**Figure 5.2.10 Pet Deletion Behavior Flowchart**

<table>
<thead>
<tr>
<th>Module</th>
<th>Flowchart: Deleting a Pet in the System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
<tr>
<td>Inputs</td>
<td>-Request to Delete Pet in App</td>
</tr>
<tr>
<td>Outputs</td>
<td>-All Pet Information on Selected Pet Deleted from Database</td>
</tr>
<tr>
<td>Description</td>
<td>The flow chart shows the process of deleting a pet from the system. First the request to delete a pet is done in the application. The application then deletes the pet’s information from both the pet</td>
</tr>
</tbody>
</table>
The application can change pet personal information and locking restrictions. The flow chart for editing the pet personal information and locking restrictions in the dog door system is detailed in Figure 5.2.11.

![Flowchart: Updates in Application of Pet Personal Information or Locking Restrictions](image)
The flow chart shows the process updating pet personal information or locking restrictions in the database. When the application user updates either pet personal information or locking restrictions the updated data is sent to the database, and the database is updated.

Data Flow Diagrams:

In the dog door system, data flows from the application to the dog door and from the microprocessor of the dog door to the other components of the dog door. The details on how data flows throughout the dog door system can be seen in Figure 5.2.12.
The dog door system has many different features that control the data flow through different parts of the system. The details on how data flows throughout the dog door system in terms of feature functionality are detailed in Figure 5.2.13. The data flow in the Figure 5.2.13 starts with either the RFID tag input or the user input in the application.

**Figure 5.2.13 Level 1 Data Flowchart**
the door and the database is updated with the dog door access information. The system flow diagram also shows how the major features of the application interact with the database and the dog door. Each of these distinctive features, besides app update, is initiated by the user input.

In the application, data flows throughout the different pages of the application and to the server where all data the application uses is stored. Figure 5.2.14 shows how navigation is performed throughout the application as well as how data flows to and from the server from the application.

*Figure 5.2.14 Application Data Flow Diagram*
<table>
<thead>
<tr>
<th>Module</th>
<th>Application Data Flow Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
<tr>
<td>Description</td>
<td>The application data flow diagram shows each of the pages of the application. Upon the application’s start, the user is taken to the Pet Information Page. From there the user can go to the pet activity page, locking restriction page, individual pet page. The pet information page then leads to pages on each of the individual pets in the system. The pet information page, pet activity page locking restriction page display information related to their associated database table with the same name. The individual pet page displays pet information and activity information on the selected pet. The data flow diagram shows how each page can change the database. The data flow diagram also shows the updating of each page upon detection of changes to the pet database, specifically from the dog door.</td>
</tr>
</tbody>
</table>

State Diagrams:

Whether the dog door is locked or unlocked relies on a variety of different conditions. The details on how to dog door goes from the locked to the unlocked state and the unlocked to locked state is detailed in Figure 5.2.15.
Figure 5.2.15 Locking Module State Diagram

<table>
<thead>
<tr>
<th>Module</th>
<th>Locking Module State Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Andrew Shetler</td>
</tr>
<tr>
<td>Description</td>
<td>The locking module state diagram shows the condition for moving between the two states of the locking module, which are locked and unlocked.</td>
</tr>
</tbody>
</table>

**Subsystem Application/Server:**

The Azure server was created. The server's name is pet-server.database.windows.net. The Azure SQL database for the system was also created along with each of the required database
tables. The SQL database is named Pet-Database. Each or the created database tables along with their columns and their columns data types can be seen below.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>varchar(50)</td>
<td></td>
</tr>
<tr>
<td>Image</td>
<td>varchar(MAX)</td>
<td></td>
</tr>
<tr>
<td>InOut</td>
<td>bit</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 5.2.16 Pet Information Database Table*

In the Pet Information Database table, the Id column, of data type bigint, is the unique id of the RFID tag associated with the pet. The name column, of data type varchar(50), is the pet’s name inputted in the application by the user. The image column, of data type varchar(MAX), is a link to the profile image of the pet that is stored in Azure Blob Storage. The InOut column, of data type bit, is a flag for whether the last known location of the pet was inside or outside.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>InOut</td>
<td>bit</td>
<td></td>
</tr>
<tr>
<td>TimeStamp</td>
<td>varchar(MAX)</td>
<td></td>
</tr>
<tr>
<td>Image</td>
<td>varchar(MAX)</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 5.2.17 Pet Activity Database Table*

In this table, the Id column, of data type bigint, is the unique id of the RFID tag associated with the pet. The InOut column, of data type bit, is a flag for whether the pet went inside or outside through the dog door. The TimeStamp column, of data type varchar(MAX) is the time the pet activity occurred. The image column, of data type varchar(MAX) is a link to the image of the pet activity that is stored in Azure Blob Storage.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeStart</td>
<td>varchar(50)</td>
<td></td>
</tr>
<tr>
<td>TimeStop</td>
<td>varchar(50)</td>
<td></td>
</tr>
<tr>
<td>LockUnlock</td>
<td>bit</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 5.2.18 Locking Restriction Database Table*
In this table, the TimeStart column, of data type varchar(50), is the starting time pet activity is restricted through the dog door. The TimeStart column, of data type varchar(50), is the ending time pet activity is restricted through the dog door. The LockUnlock column, of data type bit, is a flag for whether the dog door is always locked or unlocked during the restricted time. The TimeStart and TimeStop column entries will follow the naming scheme <Day>_<Hour>_<Minute>. These three database tables are the implementation of the database tables described in the Server Level 1 Block Diagram in Figure 5.2.3.

The Blob storage for storing images was also created. The Blob storage will store images from the dog door of the pet activity and from the application of pet profile images. The Azure storage account is named petimagestorage. Inside the storage account is a created container named pet-images that will hold all the images in the project. An image of the stored images in the container can be seen below.

![Azure Blob Storage](image)

Figure 5.2.19 Azure Blob Storage

When an image is added in Blob storage a URL to the image is created. This URL is then what is stored in the SQL database to be easily accessed by the application. An example of the image with URL can be seen below.
The theory of operation for adding a pet profile image from the application into Blob storage is for the user to select an image for a selected pet on their application. Once the image is selected, it will then be uploaded into the Blob storage with the edited image name of Profile_<PetId>.jpg. The link created will then be stored in the pet’s associated image entry in the pet information database table. Similarly, images of pet activity from the dog door will be added to the Blob storage. The image will follow the naming scheme Activity_<TimeStamp>.jpg. The image link will then be stored in the image column of the newly created row in the Pet Activity Database Table for the new activity.
For the working subsystem, entries to the Pet Information Database Table and the Blob storage were added manually. The images added for the subsystem demonstration can be seen in Figure 5.2.18. The entries added to the Pet Information Table for the demonstration can be below.

Note the Id column entries are set to placeholder values since no RFID tags ids were known.

The application functionality was developed to take the information from the above database table and display it inside the application. The code repository for the up-to-date application can be found on GitHub with the URL https://github.com/aas183/SmartDogDoor. The main functionality of connecting to the database and displaying the data inside the app is done in four files. The approach for displaying data follows the MVVM (Model-View-ViewModel) software architecture (Stonis 2022). In the Model folder, Pet.cs holds the Pet class. The Pet class is a class that holds all information for the Pet Information Database Table. The code for the Pet.cs file can be seen below.
namespace SmartDogDoor.Model;

//class for pet information database table information
public class Pet
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Image { get; set; }
    public string InOut { get; set; }
    public Color InOutColor { get; set; }
}

In the View folder, the MainPage.xaml file is used to display data to the user. The role of the view is to “define the structure, layout, and appearance of what the user sees on screen” (Stonis 2022). The MainPage.xaml file calls the function GetPetsAsync in the PetViewModel.cs file which returns a list of pets from the Pet Information Table that is displayed by the page using dynamic data binding. The code for the MainPage.xaml file can be seen below.
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:model="clr-namespace:SmartDogDoor.Model"
xmlns:viewModel="clr-namespace:SmartDogDoor.ViewModel"
x:DataType="viewModel:PetViewModel"
Title="{(Binding Title)}"
Shell.BackgroundColor="{(StaticResource Primary)}"
x:Class="SmartDogDoor.View.MainPage">

<ContentPage.Resources />

<ContentPage.Behaviors>
<toolkit:StatusBarBehavior StatusBarColor="{(StaticResource Accent)}"/>
</ContentPage.Behaviors>

<Grid ColumnDefinitions="*,*"
ColumnSpacing="5"
RowDefinitions="*, Auto"
RowSpacing="0">
<CollectionView BackgroundColor="Transparent"
Grid.ColumnSpan="2"
ItemsSource="{(Binding Pets)}"
SelectionMode="None">
<DataTemplate x:DataType="model:Pet">
<Grid Padding="10">
<Frame HeightRequest="125" Padding="0" Style="{StaticResource CardView}"
Grid.Padding="0" ColumnDefinitions="125,*">
<Image Aspect="AspectFill" Source="{(Binding Image)}"
WidthRequest="125" HeightRequest="125" />
<VerticalStackLayout Grid.Column="1" Padding="10" VerticalOptions="Center">
<Label Text="{(Binding Name)}" Style="{StaticResource LargeLabel}" />
<HorizontalStackLayout>
<Frame Margin="1"
BorderColor="{(Binding InOutColor)}"
BackgroundColor="{(Binding InOutColor)}"
CornerRadius="50"
WidthRequest="4"
IsClippedToBounds="True"
HorizontalOptions="Start"
VerticalOptions="Start">
<Label Text="{(Binding InOut)}" Style="{StaticResource MediumLabel}" />
</Frame>
</HorizontalStackLayout>
</VerticalStackLayout>
</Grid>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>

<Button Text="Get Pets"
Style="{StaticResource ButtonOutline}"
Command="{(Binding GetPetsCommand)}"
IsEnabled="{(Binding IsNotBusy)}"
Grid.Row="1"
Margin="8"/>

<ActivityIndicator IsVisible="{(Binding IsBusy)}"
IsRunning="{(Binding IsBusy)}"
Color="{(StaticResource Primary)}"
HorizontalOptions="FillAndExpand"
VerticalOptions="CenterAndExpand"
Grid.Row="2"
Grid.ColumnSpan="2"/>
</Grid>
</ContentPage>

Figure 5.2.23 Pet Information Page View Code
In the ViewModel folder, two different files are used BaseViewModel.cs and PetViewModel.cs. The role of the ViewModel files is to “implement properties and commands to which the view can data bind to and notifies the view of any state changes through change notification events” (Stonis 2022). The BaseViewModel class is the parent class of PetViewModel. The BaseViewModel has two different variables, IsBusy and Title. IsBusy keeps track of whether the app is busy, such as getting information from the database, and notifies the View when it is busy. The Title is for setting the title of each page. The PetViewModel is the ViewModel for MainPage.xaml. The PetViewModel contains the function, GetPetsAsync, that gets the data from the Pet Information Database Table and stores it inside an observable collection. The GetPetsAsync function uses the PetService object to call out to the Azure SQL database to get the data. The GetPetsAsync function then returns the observable collection of pets the calling view, MainPage.xaml. The Code for the BaseViewModel.cs and PetViewModel.cs files can be seen below.

```csharp
namespace SmartDogDoor.ViewModel;

public partial class BaseViewModel : ObservableObject
{
    public BaseViewModel()
    {
    
    [ObservableProperty]
    [NotifyPropertyChangedFor(nameof(IsNotBusy))]
    bool isBusy; //For keeping track if app is busy

    [ObservableProperty]
    string title; //for title of each page

    public bool IsNotBusy => !IsBusy;
}
```

Figure 5.2.24 Base ViewModel Code
using SmartDogDoor.Services;
namespace SmartDogDoor.ViewModel;

public partial class PetViewModel : BaseViewModel
{
    PetService petService; //Object of PetService for getting info from database
    public ObservableCollection<Pet> Pets { get; } = new(); //Data Collection of data from Database

    //Constructor of ViewModel
    public PetViewModel(PetService petService)
    {
        Title = "Pets";
        this.petService = petService;
        //GetPetsAsync();
    }

    //Not Fully Implemented but function from going to pet info page to individual pet pages
    [RelayCommand]
    async Task GoToDetailsAsync(Pet pet)
    {
        if (pet is null)
            return;

        await Shell.Current.GoToAsync($"{nameof(DetailsPage)}", true,
            new Dictionary<string, object>
            {
                {"Pet", pet }
            });
    }

    //Get Details from pet Information Page
    [RelayCommand]
    async Task GetPetsAsync()
    {
        //If data pull is already occurring quit
        if (IsBusy) return;

        try
        {
            IsBusy = true;
            var pets = await petService.GetPets(); //Get pets

            if (Pets.Count != 0) //clear pet list if full
                Pets.Clear();

            foreach (var pet in pets) //update pet list from service call
                Pets.Add(pet);
        }
        catch (Exception ex) //if error
        {
            Debug.WriteLine(ex);
            await Shell.Current.DisplayAlert("Error!",
                "$Unable to get pets: {ex.Message}" , "OK");
        }
        finally
        {
            IsBusy = false; //set busy back to false
        }
    }
}

Figure 5.2.25 PetViewModel code
Lastly, in the Service folder, PetService.cs holds a class PetServices that has functions that get data from the server. For the demonstration, the NuGet Package MicroSoft.Data.SqlClient was used to talk to the database. The package uses the database’s user id and password to make a connection. In the final project an web based REST API will be developed that can be called from the .NET MAUI application to communicate with the database as it provides a more secure connection. The code for the PetService.cs file can be seen below.
namespace SmartDogDoor.Services;
using System;
using Microsoft.Data.SqlClient;
using System.Text;

//Class for accessing outside data from pet server
public class PetService
{
    public PetService()
    {
    }

    //Lists for entries from database tables
    List<Pet> petList = new();
    List<PetActivity> petActivityList = new();
    List<Lock> lockList = new();

    //Function to get data from Pet Information Database Table
    public async Task<List<Pet>> GetPets()
    {
        //Note: Need to make more async
        try
        {
            petList.Clear(); //clear current pet list
            //Create connection string to database
            SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
            builder.DataSource = "pet-server.database.windows.net";
            builder.UserID = "";
            builder.Password = "";
            builder.InitialCatalog = "Pet-Database";

            using (SqlConnection connection = new SqlConnection(builder.ConnectionString))
            {
                Console.WriteLine("\nQuery data:");
                Console.WriteLine("========================================\n");

                String sql = "SELECT Id, Name, Image, InOut FROM Pet_Information_Table"; //selection from database
                using (SqlCommand command = new SqlCommand(sql, connection))
                {
                    await connection.OpenAsync(); //connect to database
                    using (SqlDataReader reader = command.ExecuteReader())
                    {
                        while (reader.Read()) //read until NULL entry in database
                        {
                            //Create new Pet Object
                            Pet pet = new Pet();
                            Console.WriteLine(reader.GetString(0));
                            //Store variables from database in pet object members
                            pet.Id = Convert.ToString(reader.GetInt64(0));
                            pet.Name = reader.GetString(1);
                            pet.Image = reader.GetString(2);
                            bool inOut = reader.GetBoolean(3);
                            //set inOut member variables
                            if (inOut)
                            {
                                pet.InOut = "In";
                                pet.InOutColor = Color.FromRgba("#008450");
                            }
                            else
                            {
                                pet.InOut = "Out";
                                pet.InOutColor = Color.FromRgba("#B81D13");
                            }
                            petList.Add(pet); //add pet to list of pets
                            Console.WriteLine("{0} {1} {2} {3}", pet.Id, pet.Name, pet.Image, pet.InOut);
                        }
                    }
                }
            }
        }
        catch (SqlException e)
        {
            Console.WriteLine(e.ToString());
        }
        //Console.ReadLine();
        return petList;
    }
}

Figure 5.2.26 Pet Service Code
The application testing was run on an Android emulator. The application's main page, the pet information page, is automatically opened when the application is started. The pets in the pet information database table are automatically displayed when the application opens. Each pet is displayed in a card view with the pet’s image, name, and in/out status. The display in the application can be seen below.

![Figure 5.2.27 Pet Information Page](image)
In the final application the “Get Pets” Button will not be present. Also, a button will be displayed after the last pet card that when pressed will start the process of adding a new pet. Navigation to the individual pet page will be done by tapping on a pet's card. The individual pet page will display information on the pet from the pet information table and its activity from the pet information database table.

The main application pages will be navigated by the tab bar at the bottom of the application seen in the above figure. Clicking on the Activity Tab will bring the user to the Pet Activity Page as seen below.

![Figure 5.2.28 Pet Activity Page Navigation](image-url)
The Activity Page will display all the information from the Pet Activity Database Table with the ability to filter the activity by the activity time, or pet name. The information will be shown in a card view like the Pet Information Page.

Clicking on the Access tab will bring the user to the Locking Restrictions Page as seen below.

![Figure 5.2.29 Locking Restriction Page Navigation](image)

The Locking Restriction Page will show all the configured locking restrictions as well as allow the user to add or delete locking restrictions.
In the application GitHub repository, many of the different files contain pseudocode for the functionality of the application. The pseudocode for the addition of pets on the application repository. The addition of pets is done from the Pet Information Page (MainPage.xaml). From the page the addPetAsync function from the Pet ViewModel is called when the corresponding button is pressed. The pseudocode code for the Pet View Model Function that details the addition of a new pet can be seen below.

```csharp
async Task addPetsAsync()
{
    // Display a alert box saying do you want to add pet? Options are Yes or no

    // If no
    // quit

    // If yes
    // call pet service, addPet(), to add row to database that with generic pet name ie. Pet_3
    // and no image and no id and InOut of in

    // After prompt user to hold new RFID tag to door

    // Wait for 1 minute and doing so continously check database to see if new row has id populated from dog door

    // If it is populated
    // tell the user the pet was successfully added to system update view

    // If not
    // tell user pet addition was unsuccessful then call pet service, deletePet(), to delete pet entry
}
```

*Figure 5.2.30 Pet ViewModel add Pets Function Psuedocode*

The addPetAsync function calls the addPets function in the pet service class which connects to the database and adds the pet. The function’s pseudocode can be seen below.

```csharp
public async void addPet()
{
    // connect to database

    // read number of entries in pet information database table

    // add a new row to database table and make the pet name Pet<numofPets+1> and make InOut True
}
```

*Figure 5.2.31 Pet Service add Pet Pseudocode*
The process of deleting the pet in the application is also shown in the application repository. The deletion is done from the Individual Pet Page (PetDetailsPage.xaml) when the delete pet button is pressed. The page then calls the deletePetAsync function in the Pet Details View Model. The pseudocode for the function can be seen below.

```csharp
async Task deletePetsAsync(string Id)
{
    //Prompt user "Are you sure if you want to delete pet from system?" (Yes/No)
    //If no
    //quit
    //If yes
    //call pet service, deletePetImages(), to delete all images associated Id
    //all pet services, deletePet(), to delete all entries for selected pet's id in pet information and pet activity table
}
```

*Figure 5.2.32 Pet Details ViewModel delete Pet Pseudocode*

The deletePetAsync function calls two distinct functions in the pet service class. First deletePetAsync calls the deletePetImages function, which deletes all images associated with the pet in Blob storage. The pseudocode for the function is below.

```csharp
public async void deletePetImages(string ID)
{
    //connect to database and Blob storage
    //search pet information database table and pet activity database table for all entries for passed ID and get all entries image URLs
    //Using URL for images go into blob storage and delete all images with the received URLS
}
```

*Figure 5.2.33 Pet Service Delete Pet Images Pseudocode*

The function deletePets in the pet service class is then called to delete all the entries in the database tables associated with the pet. The pseudocode for both functions can be seen below.

```csharp
public async void deletePet(string id)
{
    //connect to database
    //read number of entries in pet information database table
    //find row with passed id in pet information database table and delete pet
    //find entries in pet activity database table with passed id and delete them
}
```
The pseudocode for updating a pet’s information is complete. Updating a pet’s information is done from the individual Pet Pages (PetDetailsPage.xaml). There are multiple functions in the Pet Details View Model for handling whether the image or name of the pet is selected to be updated from the page. The first function, changePetNameAsync, updates the name of the pet in database. The pseudocode for the function is below.

```csharp
async Task changePetNameAsync(string name, string petID)
{
    //call pet service function changePetName() to change name of pet in pet information database table entry with passed petID
}
```

The second function, changePetImageAsync(), changes the pet profile image for the selected pet. The pseudocode for the function can be seen below.

```csharp
public async void changePetName(string id, string name)
{
    //connect to database
    //find entry in pet information database table with passed id
    //change found entry with passed id’s name to passed name
}
```
async Task changePetImageAsync(image, string petID) 
{
    //User will pick an image using XML file picker and pass it to this function 
    //change image name to Profile_<petID>.<extension>
    //call pet services, addPetImage() to upload image to Azure Blob Storage and return image URL 
    //call pet services, addPetImageDatabase(), with image URL and petID to put image URL in pet information database table entry with passed petID
    //returns URL
    //call pet services, deletePetImage() with returned URL from addPetImage() call, to delete old pet profile image if it exists

Figure 5.2.37 Pet Details ViewModel Change Pet Image Pseudocode

The changePetImageAsync function calls three distinct functions in the pet service class to facilitate the change of the pet profile image. The first function, addPetImage, adds the new image to the Blob storage and returns the uploaded images URL. The second function, addPetImageDatabase, adds the URL of the newly updated image, puts it in the Pet Information Database Table entry with the selected pet, and returns the URL that was previously in the database for the pet. The third function, deletePetImage, deletes the previous pet profile image from Blob storage using the URL returned from addPetImageDatabase. The pseudocode for these functions can be seen below.

Error! Not a valid embedded object.

Figure 5.2.38 Pet Service Change Pet Image Pseudocode

The pseudocode for the process of filtering pet activities information is also in the application. The Pet Activity Page calls the filterActivities function in the Pet Activities ViewModel. The function takes the list of activities and sorts the activities entries based on the passed parameters. The pseudocode for the function can be seen below.
async Task filterActivity(bool time_pet, string petID) 
{ 
//call pet service, getPetActivities(), to update Activities observable collection

//check time_pet bool
//If 1 time filter activated
//sort entries in observable collection by most recent time first
//If 0 pet name selected.
//look through observable collection and delete entries that do not have the passed petID
//sort entries by most recent activity first
}
public async void addLock(string timeStart, string timeStop, bool lockUnlock)
{
    //connect to database
    //add entry to locking restriction database table with passed parameters
}

public async void deleteLock(Lock lock)
{
    //connect to database
    //delete entry of locking restriction database table with has entires same as the members of passed Lock object
}

Figure 5.2.41 Pet Service Add and Delete Locking Restriction Pseudocode

Subsystem RFID:

The Pseudocode for the RFID reader to microprocessor using UART is as follows:

// Define communication interfaces
Interface SerialPort(0); // logs
Interface SerialPort2(2); // C1 connection
Interface C1_Interface(SerialPort2);

// Function to check and handle status
function check_and_handle_status(status):
    switch (status):
        case 0: // Success
            break;
        case 0xff: // Unresponsive
            print("No response from Pepper C1. Check connections.");
            break;
        default:
            print("Error received: 0x%04X\n", status);
            break;

// System setup
function system_setup():
    // Initialize serial communication
    SerialPort.begin(115200, SERIAL_8N1, 3, 1);
    SerialPort2.begin(115200, SERIAL_8N1, 16, 17);
// Main program loop
function main_loop():
    // Check if Pepper C1 is responsive
    if (C1_Interface.SendDummyCommand() == 0xff):
        print("Pepper C1 unresponsive. Check connections.");
        return;
    print("Communication Established");
    // Get firmware version
    C1_Interface.GetVersion(firmware_version);
    print("Firmware Version: %s\n", firmware_version);
    // Check status and get tag count check_and_handle_status(C1_Interface.GetTagCount(&count));
    print("Current Tag Count: %d\n", count);
    // Loop through tags if count is greater than 0
    if (count > 0):
        for (uint8_t idx = 0; idx < count; idx++):
            C1_Interface.GetUid(idx, &type, &param, uid, &uid_len);
            print("UID %d:");
            for (uint8_t k= 0; k < uid_len; k++):
                print(" %02X", uid[k]);
                print("\n");
            // Enable internal polling for RFID UIDs
            C1_Interface.SetPolling(1);
    print("Internal polling enabled. Next test in 5 seconds...");
    sleep(5);
}

Figure 5.2.42 Pseudocode for the RFID Reader to Microprocessor using UART

6 Mechanical Sketch

The dog door housing should encapsulate all electrical hardware while allowing for a medium-sized dog to pass through when unlocked. The top housing will have the main processing
unit, cameras, antennae, and power subsystem. From further down the door, the solenoid arm and reed switch will be internally mounted in the dog door frame.

7 Team Information

Engineering Documents Manager:

Boissoneault, William – EE
Hardware Manager:

Charlson, Benjamin – EE

Design Team Lead:

Stump, Jacob – CpE

Software Manager:

Shetler, Andrew – CpE
8 Parts Lists

<table>
<thead>
<tr>
<th>Part No.</th>
<th>Qt</th>
<th>Description</th>
<th>Unit Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ai-Thinker ESP32-CAM</td>
<td>2</td>
<td>Microprocessor with camera functionality</td>
<td>$8.92</td>
</tr>
<tr>
<td>3152-000394-ND</td>
<td>2</td>
<td>RFID reader module for ESP-32 microprocessor</td>
<td>$46.66</td>
</tr>
<tr>
<td>RFID tags</td>
<td>3</td>
<td>Tags to be detected by Pepper C1 module</td>
<td>ECE Dept</td>
</tr>
<tr>
<td>FLEX-ANT1356-40x40-XXX v1</td>
<td>2</td>
<td>RFID antenna to extend range of Pepper C1 module</td>
<td>$13.02</td>
</tr>
<tr>
<td>3544-USE-18650-2600PCB-ND</td>
<td>10</td>
<td>3.7V, 2.6Ah rechargeable lithium-ion batteries</td>
<td>$6.19</td>
</tr>
<tr>
<td>ESP32 OV2640 Module TTL Downloader</td>
<td>2</td>
<td>Programmer for the microprocessor</td>
<td>$15.99</td>
</tr>
<tr>
<td>DSOL-1351-05C</td>
<td>1</td>
<td>5V, 100% Duty cycle, 11W @ 2.2A pull type solenoid</td>
<td>$22.89</td>
</tr>
<tr>
<td>BH-18650-W-ND</td>
<td>10</td>
<td>BATT HOLDER 18650 1 CELL 6&quot; LEAD</td>
<td>$4.54</td>
</tr>
<tr>
<td>MK06-6-A</td>
<td>2</td>
<td>Molded reed switch, 200V @ 0.5A capable</td>
<td>$7.93</td>
</tr>
<tr>
<td>469-9146-ND</td>
<td>2</td>
<td>MAGNET 0.250&quot;DX0.063&quot;H CYL 1=1PC</td>
<td>$0.43</td>
</tr>
<tr>
<td>YB-117-1</td>
<td>4</td>
<td>MAGNET 0.591&quot;LX0.591&quot;WX0.315&quot;H</td>
<td>$5.00</td>
</tr>
<tr>
<td>IRLZ24NPBF-ND</td>
<td>2</td>
<td>MOSFET N-CH 55V 18A TO220AB</td>
<td>$0.86</td>
</tr>
<tr>
<td>LTC1157CN8#PBF-ND</td>
<td>1</td>
<td>IC GATE DRV HI-SIDE/LO-SIDE 8DIP</td>
<td>$7.26</td>
</tr>
<tr>
<td>1N5822-TPMSCT-ND</td>
<td>2</td>
<td>DIODE SCHOTTKY 40V 3A DO201AD</td>
<td>$0.39</td>
</tr>
<tr>
<td>595-1735-ND</td>
<td>2</td>
<td>FIXED IND 390UH 5A 72 MOHM TH</td>
<td>$3.50</td>
</tr>
<tr>
<td>1189-1056-ND</td>
<td>1</td>
<td>CAP ALUM 3300UF 20% 10V RADIAL</td>
<td>$0.68</td>
</tr>
<tr>
<td>1988-1022-ND</td>
<td>2</td>
<td>SCHMARTBOARDDEZ .5MM PITCH, 12 AN</td>
<td>$6.00</td>
</tr>
<tr>
<td>296-30321-1-ND</td>
<td>2</td>
<td>IC BATT CHG LI-ION 1CELL 24VQFN</td>
<td>$10.92</td>
</tr>
<tr>
<td>296-1858-5-ND</td>
<td>1</td>
<td>IC OSC SINGLE TIMER 2.1MHZ 8-DIP</td>
<td>$0.99</td>
</tr>
<tr>
<td>1 kΩ resistor</td>
<td>1</td>
<td>1 kΩ resistor</td>
<td>ECE Dept</td>
</tr>
<tr>
<td>100 kΩ resistor</td>
<td>1</td>
<td>100 kΩ resistor</td>
<td>ECE Dept</td>
</tr>
<tr>
<td>0.1 uF capacitor</td>
<td>2</td>
<td>0.1 uF capacitor</td>
<td>ECE Dept</td>
</tr>
<tr>
<td>0.1 nF capacitor</td>
<td>1</td>
<td>0.1 nF capacitor</td>
<td>ECE Dept</td>
</tr>
<tr>
<td>3296Y-103LF-ND</td>
<td>1</td>
<td>TRIMMER 10K OHM 0.5W PC PIN TOP</td>
<td>$2.43</td>
</tr>
<tr>
<td>13-MFR100FTF52-100RCT-ND</td>
<td>1</td>
<td>RES 100 OHM 1% 1W AXIAL</td>
<td>$0.34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Azure Service</th>
<th>Description</th>
<th>Monthly Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Database</td>
<td>Azure SQL Database Basic B Tier</td>
<td>$4.90</td>
</tr>
<tr>
<td>Blob Storage</td>
<td>Azure Blob Hot Storage</td>
<td>$0.021 per GB</td>
</tr>
<tr>
<td>App Services</td>
<td>Azure App Services Shared Environment for dev/test</td>
<td>$9.49</td>
</tr>
</tbody>
</table>
## Project Schedules

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDP I 2023</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Project Design</strong></td>
<td><strong>46.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td><strong>Midterm Report</strong></td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td><strong>Problem Statement</strong></td>
<td><strong>46 days</strong></td>
<td><strong>Wed 8/24/22</strong></td>
<td>Sun 10/9/22</td>
</tr>
<tr>
<td>Need</td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Objective</td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Background</td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Marketing Requirements</td>
<td><strong>46.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Engineering Requirements Specification</td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td><strong>Engineering Analysis</strong></td>
<td><strong>46.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Locking System (Will)</td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Power</td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Solenoid</td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Digital Controls</td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Detection of Door Open</td>
<td>1 day?</td>
<td>Wed 8/30/23</td>
<td>Thu 8/31/23</td>
</tr>
<tr>
<td><strong>Door Power System (Ben)</strong></td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Voltage Requirements</td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Power Method (i.e., Battery, Outlet, etc.)</td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td><strong>Embedded System (Jacob)</strong></td>
<td><strong>33.38 days?</strong></td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Task</td>
<td>Time</td>
<td>Start Date</td>
<td>End Date</td>
</tr>
<tr>
<td>---------------------------------------------------------------------</td>
<td>--------</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Communication between Database and microprocessor</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Communication between Transmitter of Receiver of Pet Identification</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Communication Between Door Open Detection System and Microprocessor</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Microprocessor Requirements</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Camera Requirements</td>
<td>1 day?</td>
<td>Wed 8/30/23</td>
<td>Thu 8/31/23</td>
</tr>
<tr>
<td><strong>Application (Andrew)</strong></td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Communication Between Database and Application</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Application Requirements</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td><strong>Server (Andrew)</strong></td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Communication between server and microprocessor.</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Communication Between Database and Application</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Database Information Requirements</td>
<td>33.38 days?</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td><strong>Accepted Technical Design</strong></td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td><strong>Hardware Design: Phase 1</strong></td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td><strong>Hardware Block Diagrams Levels 0 thru N (w/ FR tables) (Will, Ben)</strong></td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Locking System Block Diagram Level 0 thru N</td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Power System Block Diagrams Level 0 thru N</td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td><strong>Software Design: Phase 1</strong></td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Flow Diagram</td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td><strong>Software Behavior Models Levels 0 thru N (w/FR tables) (Jacob, Drew)</strong></td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Embedded System Block Diagram Level 0 thru N</td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Task</td>
<td>Duration</td>
<td>Start Date</td>
<td>End Date</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Application Block Diagram Level 0 thru N</td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Database Block Diagram Level 0 thru N</td>
<td>46.38 days?</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Mechanical Sketch</td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Team information</td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Project Schedules</td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Midterm Design Gantt Chart</td>
<td>28.38 days</td>
<td>Wed 8/30/23</td>
<td>Wed 9/27/23</td>
</tr>
<tr>
<td>References</td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
</tr>
<tr>
<td>Midterm Parts Request Form</td>
<td>50 days</td>
<td>Wed 8/30/23</td>
<td>Thu 10/19/23</td>
</tr>
<tr>
<td>Midterm presentation file submission</td>
<td>33 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
</tr>
<tr>
<td>Midterm Design Presentations Day 1</td>
<td>0 days</td>
<td>Wed 10/4/23</td>
<td>Wed 10/4/23</td>
</tr>
<tr>
<td>Midterm Design Presentations Day 2</td>
<td>0 days</td>
<td>Wed 10/11/23</td>
<td>Wed 10/11/23</td>
</tr>
<tr>
<td>Project Poster</td>
<td>14 days</td>
<td>Tue 10/17/23</td>
<td>Tue 10/31/23</td>
</tr>
<tr>
<td>Final Design Report</td>
<td>47 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Abstract</td>
<td>47 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Hardware Design: Phase 2</td>
<td>47 days?</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Modules 1…n</td>
<td>47 days?</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Simulations</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Buck Simulation</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Schematics</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Buck Converter</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Solenoid Circuit</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
</tr>
<tr>
<td>Task Description</td>
<td>Start Date</td>
<td>End Date</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>Subsystem Demo (Ben)- Power Subsystem Demo</td>
<td>29.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Subsystem Demo (Will)- Locking Subsystem Demo</td>
<td>29.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Software Design: Phase 2</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Modules 1...n</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Code (working subsystems)</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>System integration Behavior Models</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Application Behavior</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Database Structure, and Information Layout</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Embedded System Behavior</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Subsystem Demo (Jacob)- Communication between microprocessor and server</td>
<td>29.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Subsystem Demo (Drew)- Communication between application and server</td>
<td>29.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Parts Lists</td>
<td>47.38 days?</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Parts list(s) for Schematics</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Materials Budget list</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Proposed Implementation Gantt Chart</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Conclusions and Recommendations</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td></td>
</tr>
<tr>
<td>Parts Request Form for Subsystems</td>
<td>32 days</td>
<td>Wed 9/27/23</td>
<td></td>
</tr>
<tr>
<td>Subsystems Demonstrations Day 1</td>
<td>0 days</td>
<td>Wed 11/15/23</td>
<td></td>
</tr>
<tr>
<td>Subsystems Demonstrations Day 2</td>
<td>0 days</td>
<td>Wed 11/22/23</td>
<td></td>
</tr>
<tr>
<td>Parts Request Form for Spring Semester</td>
<td>0 days</td>
<td>Fri 12/8/23</td>
<td></td>
</tr>
</tbody>
</table>
10 Conclusions and Recommendations

To summarize, the objective of this project is to create a smart dog door system to easily allow only pets registered pets in and out of the house while monitoring the pet’s activity remotely. A mobile application will be used to manage and monitor pets and restrictions on pet’s activity. The mobile application and dog door will interact and exchange information through a cloud-based database, so the mobile application and dog door can exchange information through internet no matter the distance apart. The dog door will check if a pet is registered by comparing a pet’s RFID tag id to all the saved pet ids in the database. If the pet is registered and there is no user defined locking restrictions in place for the current time in the database, the door will automatically unlock when a pet approaches the door. When the pet is detected going through the dog door, an image and timestamp of the activity will be stored in the database to be displayed to the pet owner in the application. The lock will use a mechanically control whether the dog door is locked or unlocked. The dog door system will save pet owners time and peace of mind by allowing their pets access to the outside and by allowing them to monitor their pet’s activity remotely. The design implementation and analysis in this document details how the design will achieve the marketing and engineering requirements set forth to achieve a successful achievement of the dog door system.

11 References


https://github.com/aas183/SmartDogDoor


https://doi-org.ezproxy.uakron.edu:2443/10.12928/telkomnika.v8i3.631


### 12 Appendices

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Datasheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microprocessor</td>
<td><a href="https://www.universalsolder.ca/downloads/ESP32-CAM.pdf">https://www.universalsolder.ca/downloads/ESP32-CAM.pdf</a></td>
</tr>
<tr>
<td>RFID Reader</td>
<td><a href="https://eccel.co.uk/wp-content/downloads/Pepper_C1/C1_User_manual.pdf">https://eccel.co.uk/wp-content/downloads/Pepper_C1/C1_User_manual.pdf</a></td>
</tr>
<tr>
<td>Component</td>
<td>Link</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>RFID Antenna</td>
<td><a href="https://eccel.co.uk/wp-content/downloads/Pepper_C1/ECCEL_Flexible_antennas_13_56MHz.pdf">https://eccel.co.uk/wp-content/downloads/Pepper_C1/ECCEL_Flexible_antennas_13_56MHz.pdf</a></td>
</tr>
<tr>
<td>Solenoid</td>
<td><a href="https://filecenter.deltaww.com/Products/download/04/0409/DSOL-1351.pdf">https://filecenter.deltaww.com/Products/download/04/0409/DSOL-1351.pdf</a></td>
</tr>
<tr>
<td>Battery Holder</td>
<td><a href="https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/338/114090053_Web.pdf">https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/338/114090053_Web.pdf</a></td>
</tr>
<tr>
<td>Reed Switch</td>
<td><a href="https://4a30d8fd18dae1bf393df49f4cedb726ad03ad145d2e3d346bd.ssl.cf5.rackcdn.com/datasheets/92/9240100021e.pdf">https://4a30d8fd18dae1bf393df49f4cedb726ad03ad145d2e3d346bd.ssl.cf5.rackcdn.com/datasheets/92/9240100021e.pdf</a></td>
</tr>
<tr>
<td>Permanent Magnet- Square</td>
<td><a href="https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/489/YB117-1.pdf">https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/489/YB117-1.pdf</a></td>
</tr>
</tbody>
</table>