Remote Dog Chip Reader
Spring 2021

Students: Mahmoud Sheikh Khalil
Jamal Tannous

Supervisor: Boaz Mizrahi

Project ID: 5821

The Andrew and Erna Viterbi Faculty of
ELECTRICAL & COMPUTER ENGINEERING

TECHNION
Israel Institute of Technology

Table of Contents:

Table of Figures:

1. Abstract:

Lots of pets today have microchips inserted in their body. Each one of those microchips has an id which consists of 16 bytes.

In order to scan the pet's microchip, the user needs an RFID chip scanner.

Most interfaces that deal with pet chip readers are hard to operate, you need to place the pet in a certain position with the help of extra personnel in order to manage to scan the microchip placed inside the pet.
The hardships do not end here, because after scanning the microchip, the doctor must insert the chip's number manually on a separate computer in order to access the relevant data.

It is also noticeable that most of these pet scanners are relatively very expensive.

In this project we have managed to design a compact, battery powered pet chip scanner.
The scanner transmits the serial number to a connected phone, using what is called a real time database.
Afterwards the user can access the pet's data using an application which reads from a cloud storage dedicated to store information about the pets that are/were treated in the doctor's clinic.
In case the pet is being treated for the first time, then adding the information about the pet is very easy and intuitive; the user should fill out the fields using the same application.

2. Motivation:

As mentioned above, most pet chip scanners are normally hard to operate, they are also generally expensive and need somewhat complex steps in order to get sufficient data regarding the pet.
Most of these pet scanners don’t do anything except show the scanned RFID chip serial number, the user should do all of the other steps manually.
Those chip scanners are normally expensive and can not be afforded by every vet.

Our goals in this project are to design a chip scanner which is compact and battery powered.
We want to make the experience of using the scanner as intuitive and automated as possible, this means we want t to enable the user can extract and update the data with minimal effort.
In order to accomplish this we need to design a dedicated application, which connects to databases and displays the desired data.

3. Specifications/Requirements:

- The RFID chip reader should be compact and battery powered.

- After scanning the chip’s serial number, the data transmission should occur in an automated way using a reliable data transfer protocol.

- The data should be stored on cloud storage that is easily accessible by the user.

- Adding and updating data should be intuitive and user friendly.

4. Block Diagram:

This is a simplified diagram and only captures some general ideas and steps, further information will be given in the following pages and diagrams.

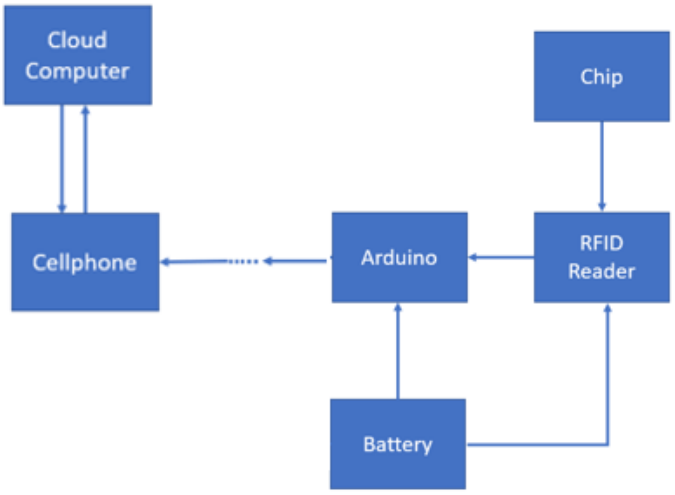


Figure 1 - Block Diagram

In this diagram we show the procedure all the way from scanning the chip to getting the data from the cloud storage system.

The microchip is scanned by the chip reader and then the data passes through Wi-Fi (Passing through some dedicated cloud storage servers, which will be explained furthermore in the following pages), until it reaches the cellphone of the user, which uses an application in order to view/add or modify the pet's data and take a picture of it.

5. Hardware Components:

1) RDM6300 (EM4100 RFID Reader):

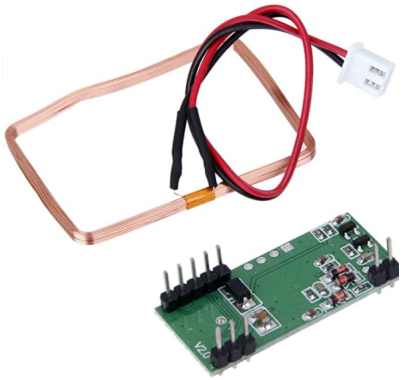


Figure 2 - RDM6300 (EM4100 RFID Reader)

- **Connects to ESP 32**
- **Low Cost.**
- **Does not consume a lot of power.**
- **Power Supply: External 5V**
- **Less than 100ms decoding time.**

2) Arduino ESP 32:

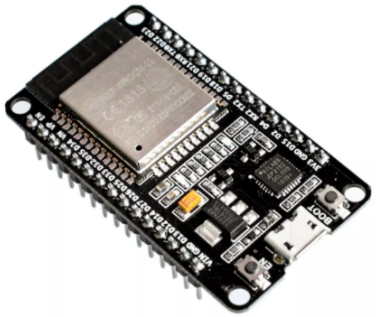
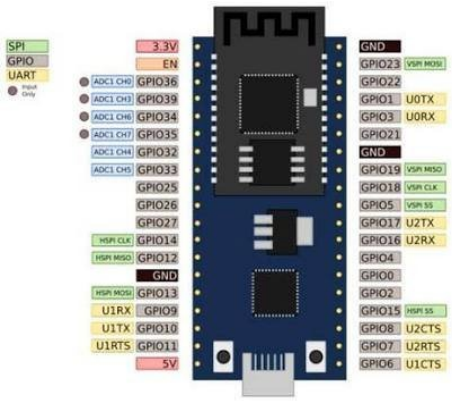


Figure 3 - Arduino ESP 32

- **Reads the scanned data from the RFID reader.**
- **Low Cost.**
- **Does not consume a lot of power.**
- **Power Supply: External 5V**
- **Transmits the scanned data using Wi-Fi.**

6. Some Schematics:



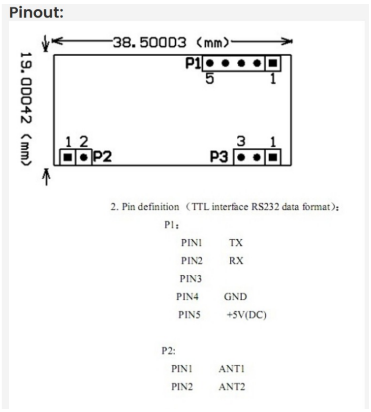


Figure 4 - Arduino ESP 32 Schematic *Figure 5 - EM4100 RFID Reader Schematic*

Connecting the hardware components:

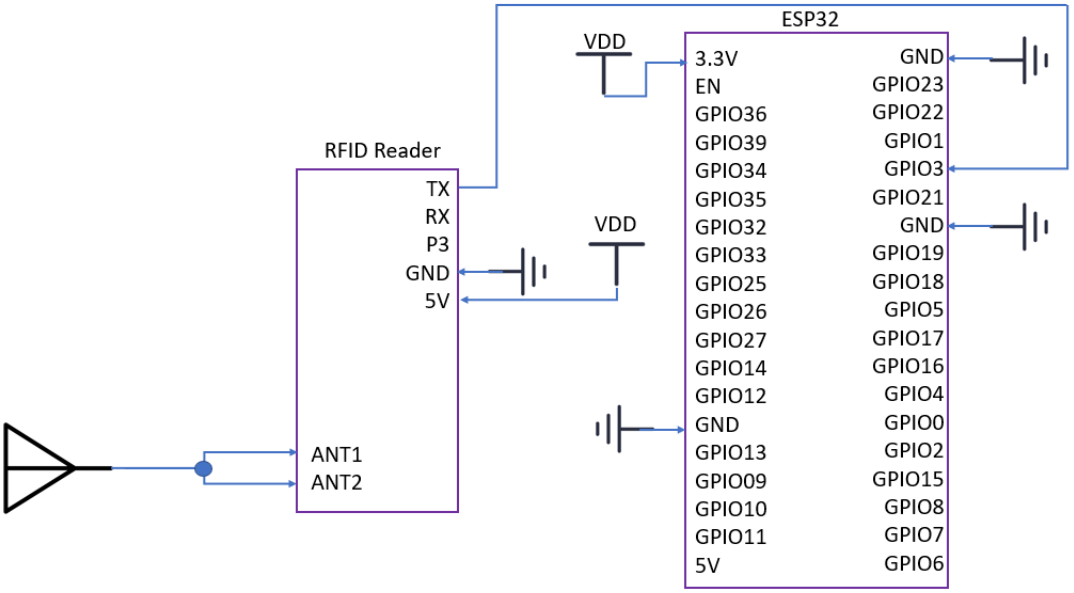


Figure 6 - Microchip Scanner (Full Hardware Schematic)

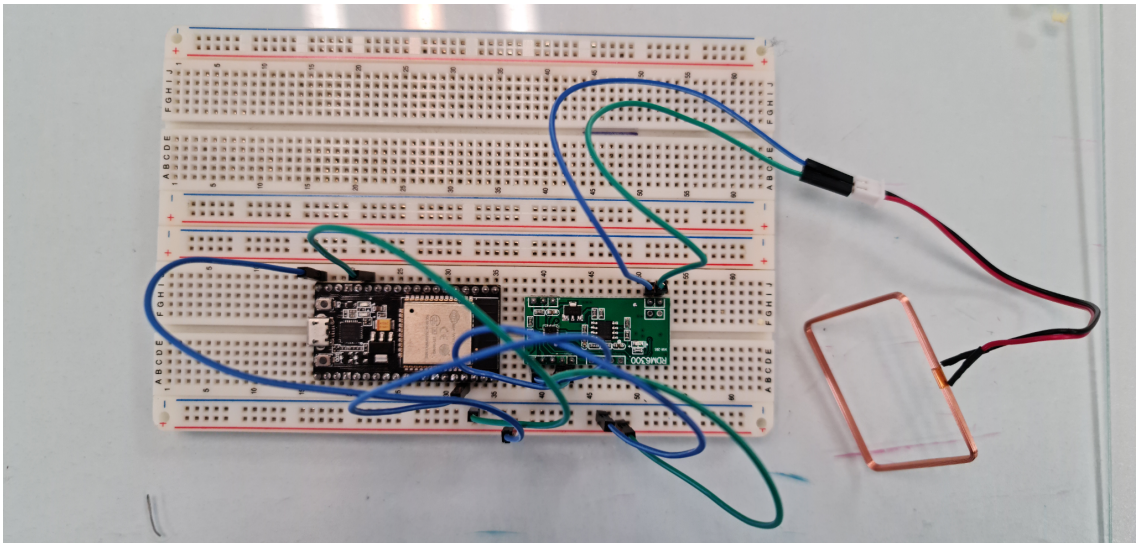


Figure 7 - Real Life Microchip Scanner (Full Hardware Breadboard Layout)

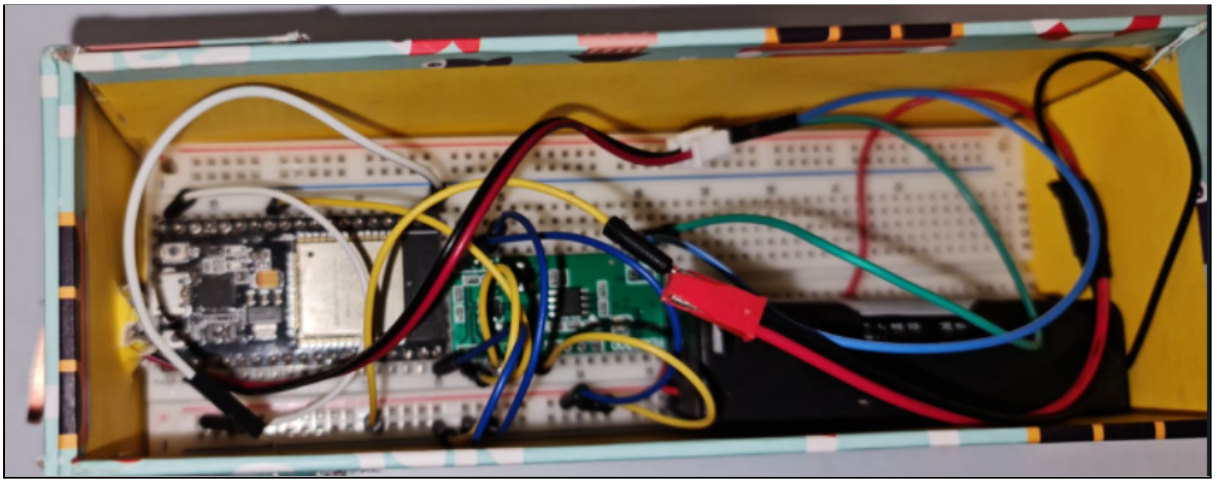


Figure 8 - Real Life Microchip Scanner (Full Hardware Breadboard Layout inside the Box)

7. Project's Software Side:

1) **Arduino IDE:** The code describing the hardware side of the project was all written in Arduino IDE, through this code we used some libraries in order to scan the data from RDM6300 (EM4100 RFID Reader), the scanned data is then transmitted through the WI-FI module which is a part of our ESP 32.
   We used libraries to enable the use of Wi-Fi in order to send the scanned data from the ESP32 to our databases. Each chip reader has a unique Id, which the user must type when signing up for the first time in order to use the application.
   Whenever the reader scans the chip, it sends its serial number using Wi-Fi to a special database called RealTime Database (which will be explained later).

   From the RDM6300 library, the class Rdm6300 includes the code, fields and methods that govern the behavior of the chip scanner module; the class contains a method called "update()" which is very important to our implementation of the scanner.

   The update() method in steps:
   - Checks whether a data stream is available.
   - Checks if the data format is correct.
   - Checks using a checksum if the scanned data has errors.
   - Checks if the available data is new.
   - Returns the newly scanned data.

After this brief explanation about the update() method, we can explain a bit about how we used it in our implementation.

Here is an attached part of the Arduino space code, displaying the main loop which waits for scanned data from the chip reader and sends the scanned data to the Realtime Database:

```
void loop() {
  if (Firebase.ready() && rdm6300.update()){
      currentId = rdm6300.get_tag_id();
      Serial.println(rdm6300.get_tag_id(), HEX);
      dtostrf(currentId,6,0,txString);
      Serial.printf("Set string... %s\n", Firebase.setString(fbdo, F("/scanner1"),txString ) ? "ok" : fbdo.errorReason().c_str());

    }

  delay(10);

}
```

Figure 9 - Main Loop in Arduino Space

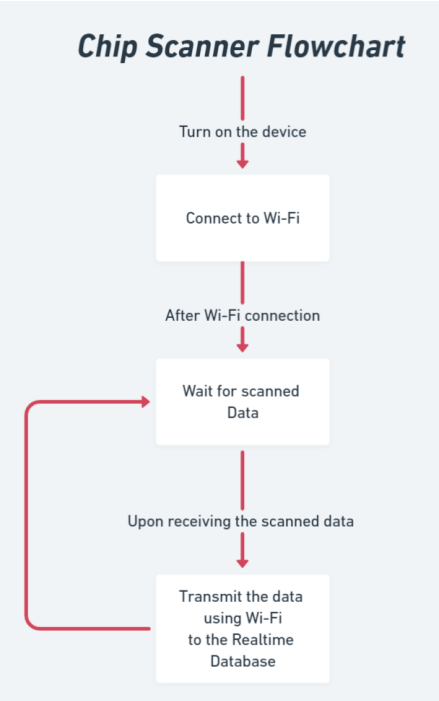


Figure 10 - Chip Scanner Flowchart

2) **Android Studio:** We've used Android Studio to program and design our application which connects to the chip scanner and accesses the data corresponding to the scanned chip.
Using Java we accessed the data that is located on the cloud storage.
We also programmed the application to be able to update and add data on the fly.
We also added some security features; the user has to be logged in using a username and a password in order to view the scanned data.
Through the application, the user can connect to the chip scanner and receive the scanned data.
After receiving the data the user can access the pet's information that is located on the cloud storage and update it.
Adding a new pet's information is also very easy; the user should only fill out the empty fields.

How the application works:
First we login/signup, afterwards we wait for the scanned data.
Once the data is ready in the Realtime Database, the application requests the relevant pet's data from Cloud Firestore and Cloud Storage.
If the requested data is not available then the application displays empty fields, which enables the user to fill in the missing information.

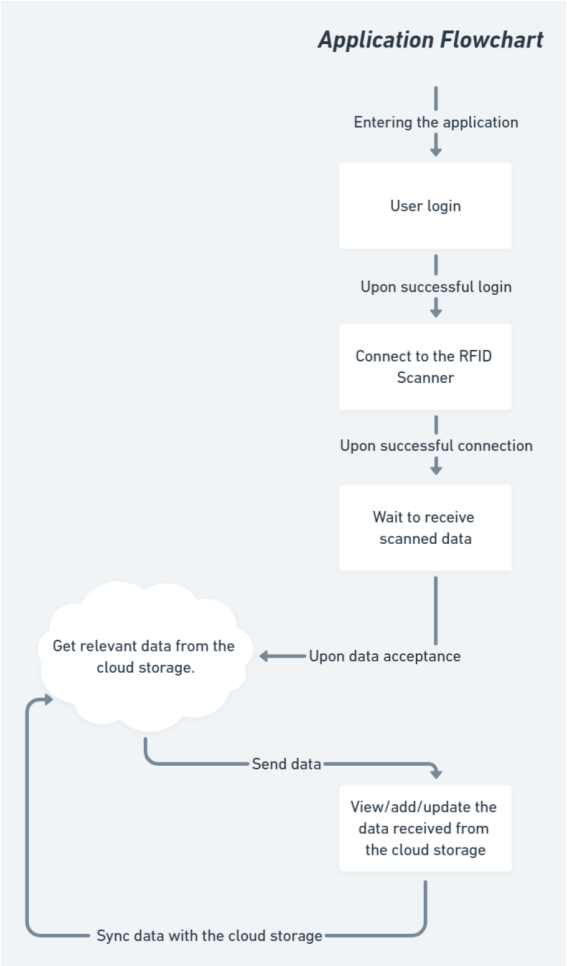


Figure 11 - Application Flowchart

Through Android Studio we managed to program several activities.
The switching between activities during the real time usage of the application contributes to a more seamless and clear user experience.
The abundance of activities also simplified the debugging stage, by making it easier to pinpoint the error's source.
The application switches between the activities through what is called an Intent, which is in short a rule notifying when to switch.

The application also has some security features, and it is built in a way in which no user can access the data of another, through these features we enable multiple users to use the same chip scanner in a way which avoids conflicts in their data, in other words, each user can only view the data which he scanned/added.
When users want to sign up, they must enter a username and a password, along with the unique id of the chip scanner that they want to use.
This data gets stored in a special database called Cloud Firestore, which saves for each user: its username,password and the chip scanner id that he is using.

The application also connects to another Cloud Firestore database which stores the information about the scanned pet. For each pet stored in this database there is a unique identifier corresponding to its scanned microchip's serial number. Whenever we scan a microchip, whose serial number does not already exist in the database, then the database will ask the application to add new data.
Finally, we want to mention that the application allows the user to take a picture of the scanned pet and save it in our cloud storage.
In order to save pictures in the cloud, we created another cloud storage system dedicated for pictures of pets, the cloud storage type is called Storage Database.
The application sends the data to the Cloud Firestore by using a map data structure unique for Firestore.
The images sent to the Cloud Storage are sent as a byte array, and the Cloud Storage converts them to pictures.

8. Explanation about the used databases

In this project we used several cloud storage servers and types, all offered by the services of Google Firebase.

What is Google Firebase: Firebase is a toolset to help us develop and improve the app, it gives us tools that cover a large portion of the services that developers would normally have to build themselves, but don't really want to build, because they'd rather be focusing on the app experience itself. This includes things like analytics, authentication, databases, configuration, file storage, push messaging…
The services are hosted in the cloud, and scale with little to no effort on the part of the developer.
That's why we've chosen to work with the Firebase toolset, this would enable us to focus on developing an app which relies on stable and scalable databases that contribute to seamless user experience.
From the toolset called Firebase we used three types of cloud storage systems: Realtime Database, Cloud Firestore, Cloud Storage.

Realtime Database: The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in real time.
Data is stored as JSON and synchronized in realtime to every connected client.
Instead of typical HTTP requests, the Firebase Realtime Database uses data synchronization, and so, every time data changes, any connected device receives that update within milliseconds.

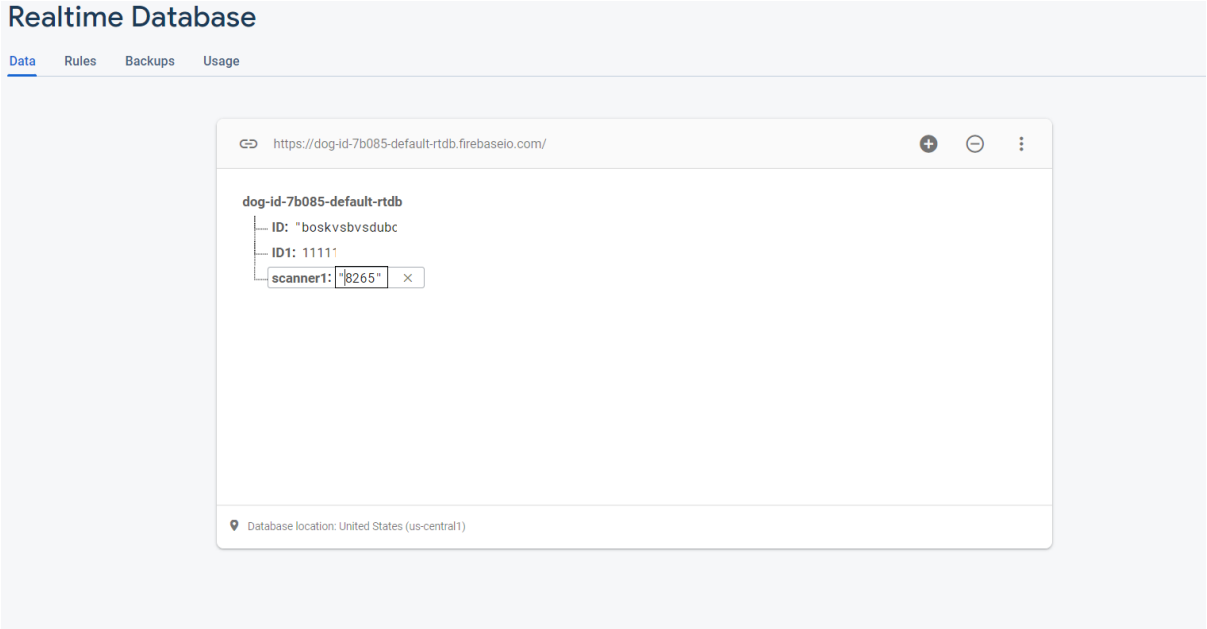


Figure 12 - Realtime Database

This database gets updated within milliseconds after scanning the pet's chip.

Cloud Firestore: Cloud Firestore is a NoSQL document database that lets you easily store, sync, and query data for your mobile and web apps at global scale.
In this system we store data in documents that contain fields mapping to values. These documents are stored in collections, which are containers for the stored documents. We can use those containers to organize the data and build queries. Documents support many different data types, from simple strings and numbers, to complex, nested objects.
In our application we used Cloud Firestore to store information about both the users and the pets that are in the system.

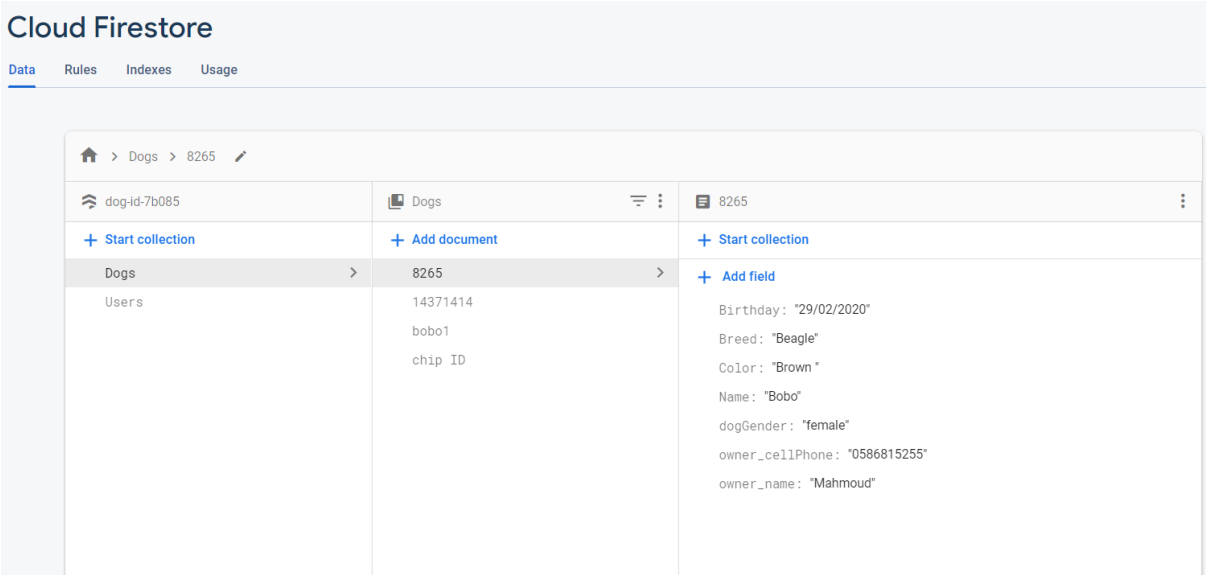


Figure 13 - Cloud Firestore for Dogs

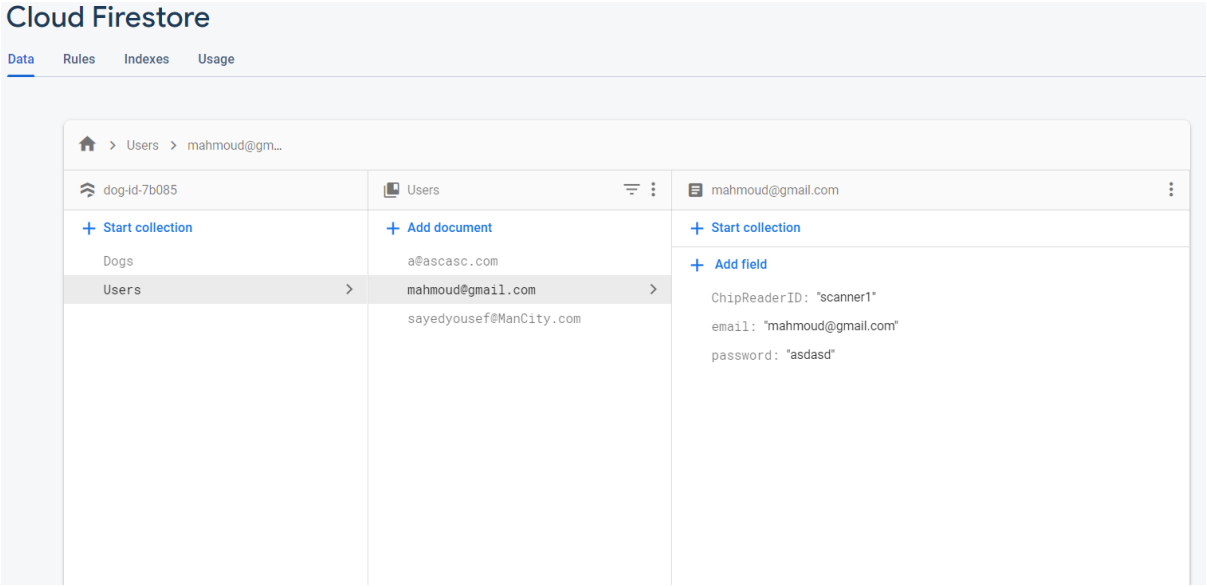


Figure 14 - Cloud Firestore for Users

Cloud Storage: Cloud Storage is designed to help you quickly and easily store and serve user-generated content, such as photos and videos.
Cloud Storage helps the client structure and build hierarchies to store data.
The client can also easily retrieve that stored data using queries.
Cloud Storage is built in such a way that it is scalable, so the developer should not worry about scalability issues.

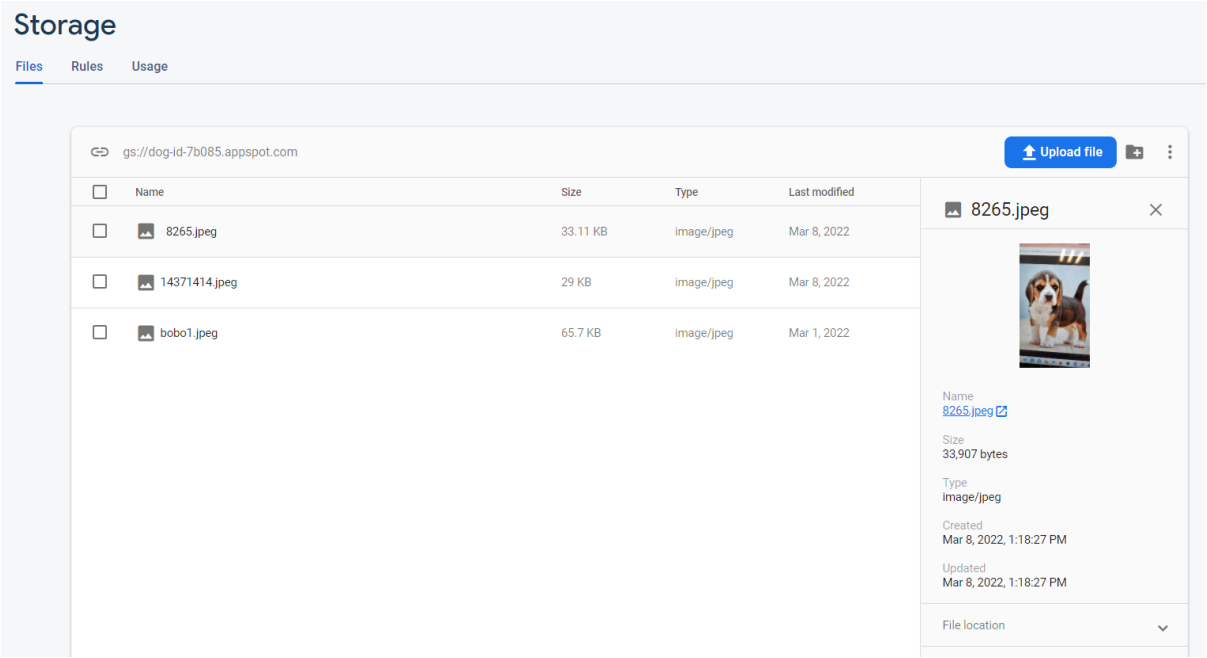


Figure 15 - Cloud Storage

That's why we decided to use Firebase as a toolset to develop our app and ease the way of accessing the data that is stored in the cloud.
Security was also taken into account, that is why we decided to use Firebase, for it is known for being very secure.

In total:

We used one Realtime Database which gets updated whenever a new chip gets scanned.

We used two Cloud Firestore databases, one which is dedicated for users, for each entry we save the username, the password and the chip scanner id that this user is currently using.

The other cloud storage we used is the one dedicated to store information about the pets in the system, the id for each pet in this database is its chip's serial number.

The last database that we stored of a Cloud Storage type, this database is dedicated to store images of the scanned pets.

By using seperate databases, where everyone serves a certain purpose, we ensure that there is a better encapsulation of the data and a better tolerance for scalability.

9. An all encompassing flowchart

After all of the above given explanations, we can now draw an encompassing flowchart of the entire procedure of using the application, i.e. the chart includes some backend steps.

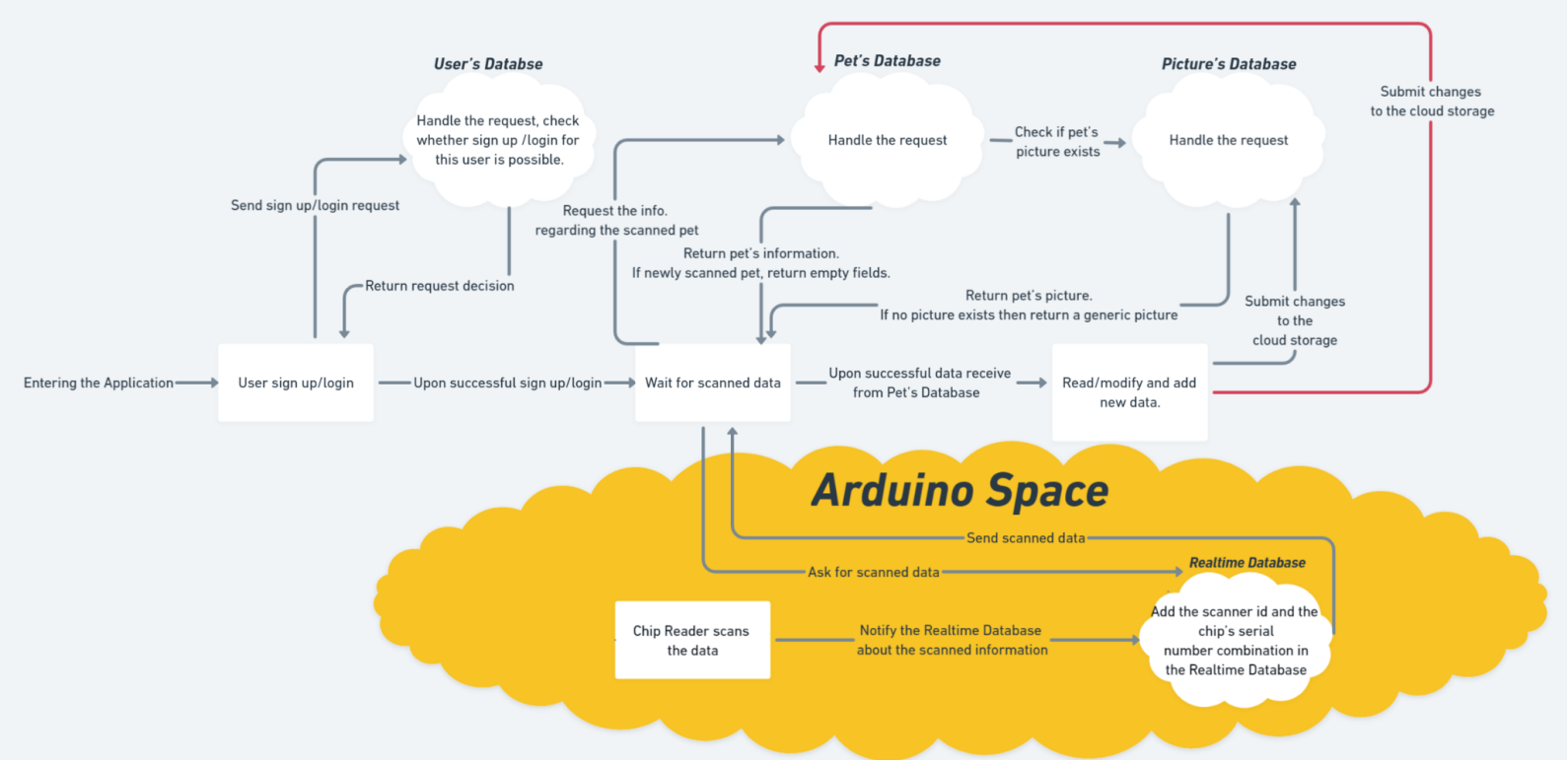


Figure 16 - An all Encompassing Flowchart

In the given flowchart, the queries to the cloud servers are all done in the background, which gives the user a seamless experience.
The “Arduino Space” describes the procedure of the actual scanning of the microchip by the chip reader and sending the scanned data to the Realtime Database.

10. Screenshots from the application

Figure 17 - Application's Login Activity

This activity is the one that appears on the screen once the user enters the application.
The user can login using the saved Email and password that are saved in our databases.
If we're talking about a new user, then the user should press on the signup button.

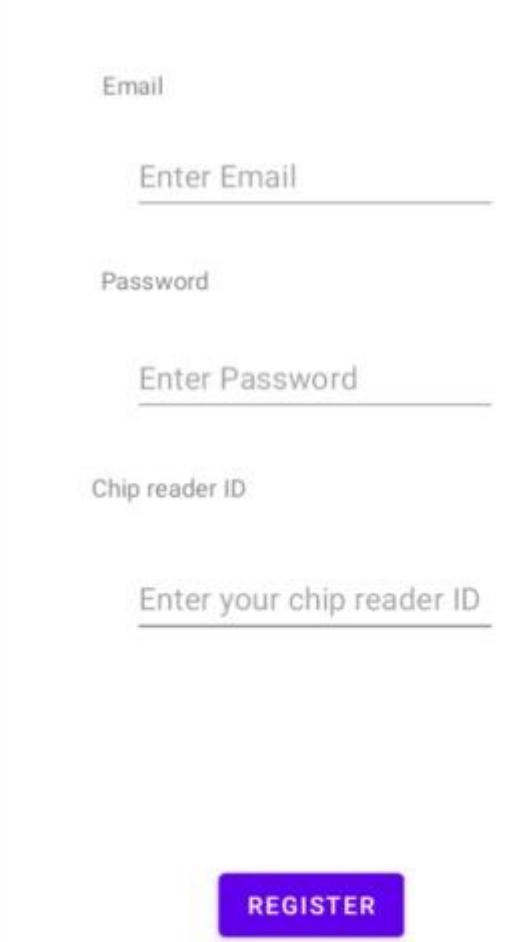


Figure 18 - Application's Signup Activity

In order to sign up, the user should enter his Email and password, which in turn will be saved in our databases for future use.

Figure 19 - Reading the scanned chip's number

In this activity, the user should press the "Read Scanned" button in order to receive the data scanned by the chip scanner. Once the application receives the data that was transmitted by the scanner through the Realtime Database, it would trigger the communication of the upcoming activity.
In the backend there will be a check whether the scanned data is new or not, and according to the output of this query, the application will pass to a corresponding activity.

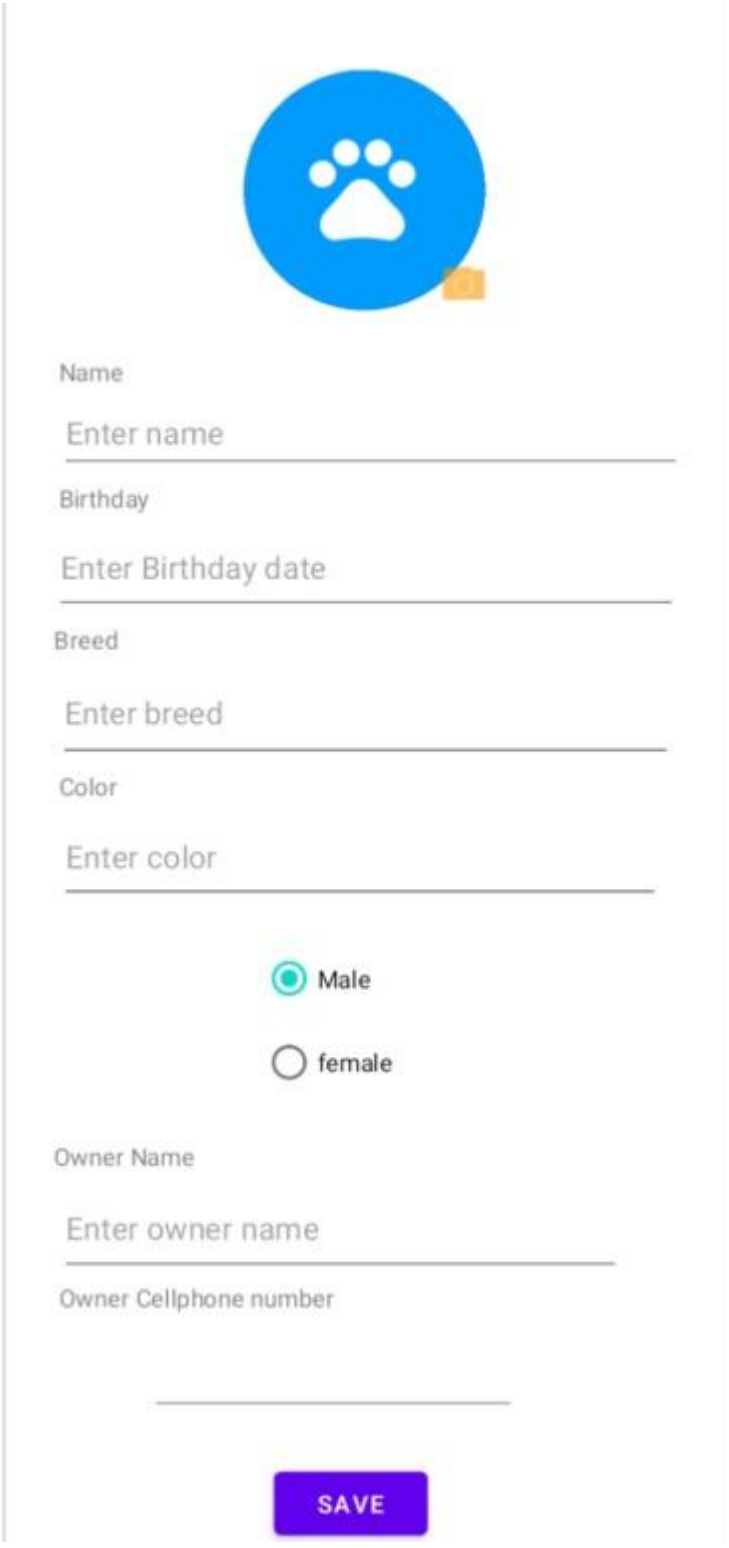


Figure 20 - Registering a new pet

In this activity the user can register the scanned pet’s information.

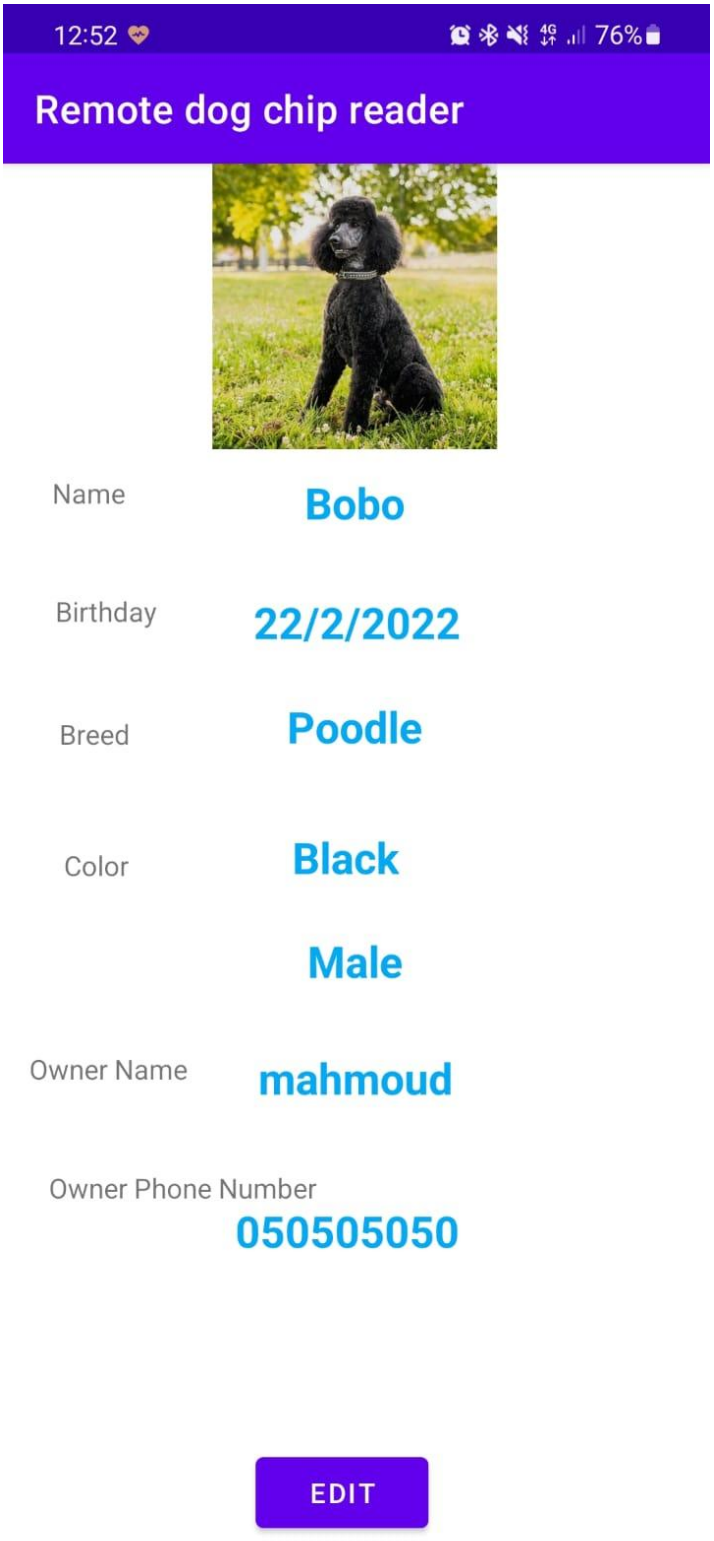


Figure 21 - Displaying the data of a scanned pet

The user can also edit the pet's information very easily, and it will be updated in the databases.

11. Bill of Material

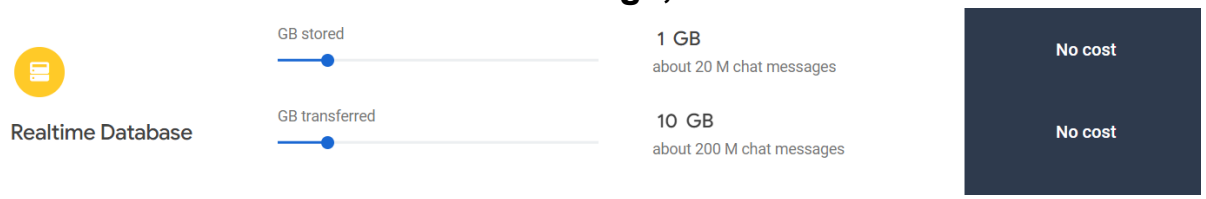
Hardware Parts:

Quantity	Device	Description	Price
1	RDM6300 (EM4100 RFID Reader for ESP32)	Scan the RFID Microchip	7.3 ₪
1	ESP32 Arduino Card	Receive the scanned data and send it to the Realtime Database	12.75 ₪

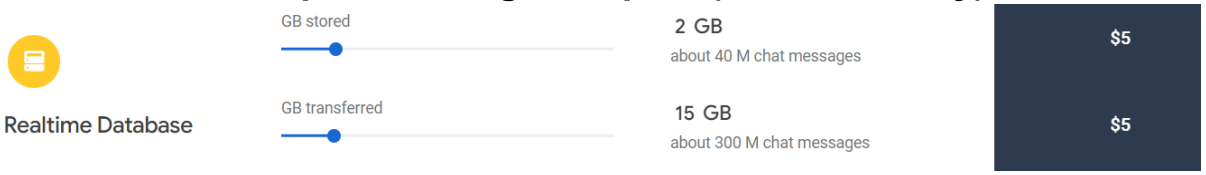
Bills for maintaining the databases:

Realtime Database:

Realtime Database is free of charge, but with certain limitations

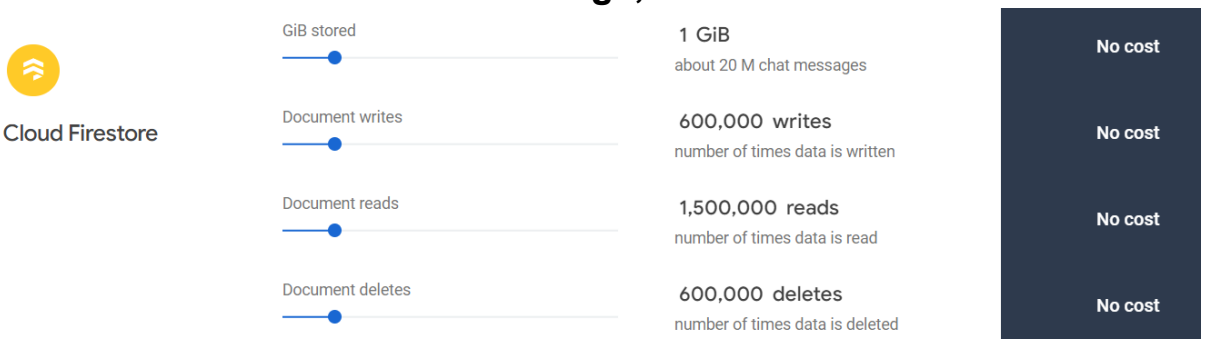


Example of a long term plan (billed monthly)

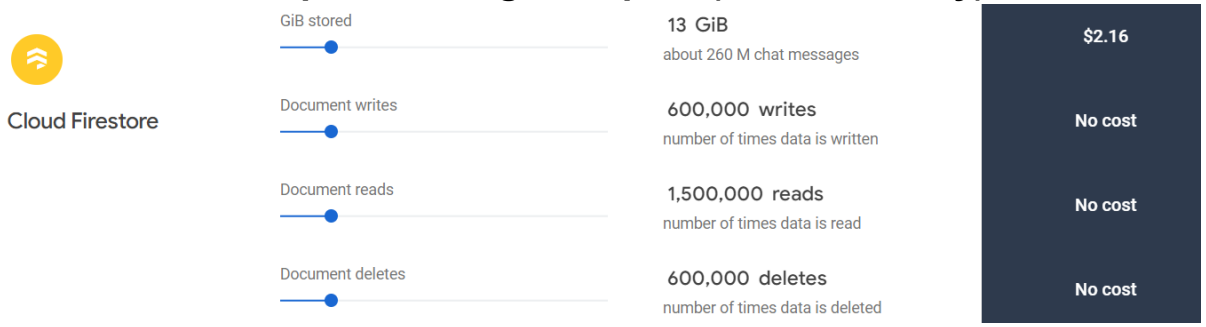


Cloud Firestore:

Cloud Firestore is free of charge, but with certain limitations

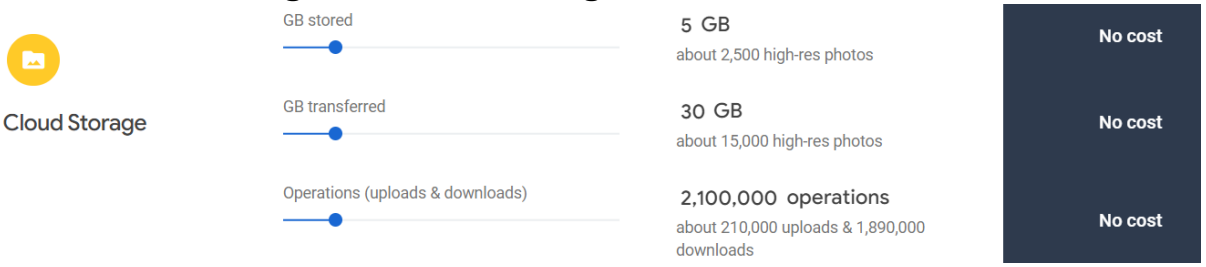


Example of a long term plan (billed monthly)

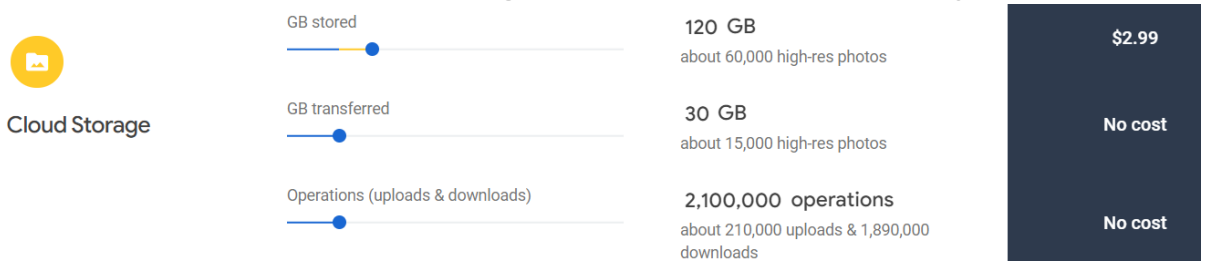


Cloud Storage:

Cloud Storage is free of charge, but with certain limitations



Example of a long term plan (billed monthly)



12. Conclusion

All in all, we managed to design an easy to use pet chip scanner.
We managed to design a device that grants a seamless and intuitive user experience
The size of the chip scanner is relatively small, so it's easy to scan the pet using it without the need of adjusting the pet’s position in order to scan its chip.
When talking about the software side of the project, which gained the main share of the time dedicated for it, we notice that we managed to create an easy to use application, which is easily maintained, updated and enhanced through the use of multiple activities with defined goals.
In this project we also managed to use some popular and secure databases, depending on the toolset called Firebase. Through Firebase and the allocation of special types of databases (Realtime Database, Cloud Firestore and Cloud Storage), we ensure a secure, easy to maintain and scalable cloud storage system that will contribute to a fast and easy reading and modification to the data stored on it.

13.Suggestions for Future Projects

When talking about suggestions for future enhancements and projects, lots of ideas popup.

Here are some ideas regarding future projects and enhancements:

1) Adding additional fields in the pet's database, for example: vaccination dates.
   One could program the application to alert the user about the need of vaccinating the pet, once the old vaccination is no longer valid.
   The alert should be invoked without scanning the pet, which means that the application should be given some permissions to run in the background.
   This enhancement might be implemented as follows: Every certain amount of a fixed time, the application will wake up in the background and go through the entire database, where the pets' informations are saved, it will scan the vaccination dates and compare it to the current date, if a certain vaccination date implies that the pet's vaccination is no longer valid, then it will be added to the "alert list" that will be delivered to the user upon the end of scanning the database.

2) Adding support for GPS trackers.
   We can modify the databases and the application to enable us to view the location of each pet in our database at every given time.
   This can be done through adding additional activities that enable the user to search for the pet using the name, breed and more.
   The availability of this feature will ease the search for lost pets, since the user can be notified about their position using the application.

3) Adding capabilities that enables the pet's owner to automatically pay the vet using the pet's RFID chip.

   It is possible to modify the application and our databases to become capable of storing payment methods and linking them to the pet's microchip.

   This obviously requires further security measurements in the application and especially the databases.