# CO600 Group Project: EchoBook

# Echo Parakeet Population Tracking & Management

*The School of Computing*

*University of Kent*

*Canterbury, United Kingdom*

Members:

Afzaal Ahmad Razeem (aar23)

Elina Voitane (ev83)

Karlis Jaunslavietis (kj232)

Priyesh Patel (prp6)

Supervised By Daniel Knox (D.Knox@kent.ac.uk)

University of Kent

## Abstract

*This report details the stages behind the creation of our EchoBook System, a system designed to help researchers in Mauritius track and manage the Echo Parakeet (echo) population. The aim of this system is to reduce the time spent by researchers manually checking if a bird has visited a feeder and to provide tracking of visits 24/7. The report aims to cover the background of the area, our design decisions, the development process, our outcomes and the future work that can be done on this project, as well as reflects on the successes and failures of the project.*

## 1. Introduction

The Mauritian Wildlife Foundation (MWF) is a non-profit conservation agency operating in Mauritius with the aim of preserving the island's threatened wildlife. Among the fauna they work with is the Echo parakeet, a threatened species of parrot. One of the most important tasks in the conservation effort is monitoring and tracking the population, which allows the conservationists to observe trends and changes, and adjust their approach based on their findings.

Currently, population tracking is done manually by the use of coloured bird rings. These are observed by field staff who then enter the data into a spreadsheet. This process is tedious, error-prone and takes up human resources that could be better used elsewhere; the population of parakeets has also recovered to the point where they are starting to run out of colours for rings.

The aim of our project was to automate the monitoring process and provide a single source of truth for all bird-related data, as well as building a user-friendly graphical interface to access and modify this data in a simple and efficient manner. This would benefit the MWF by improving the efficiency of their existing processes and enable them to get the most value out of their data.

This report describes the system our team built and the processes we took to arrive at the final product, as well as reflects on the outcomes. The following sections discuss the background of the project as well as the initial planning stages, while the later sections focus on the technical aspects of the project, aiming to give the reader a thorough understanding of our project as a whole.

## 2. Background

### 2.1 The Client

Before starting our project we had to understand who we were developing the system for and what they do. In our first project meeting we spoke to one of the researchers working with the MWF, Debbie Fogell (going forward, will be referred to as the client).

From the client we learnt of the MWF's ongoing project of restoring and maintaining the echo population in Mauritius, this project is a joint effort between MWF and the Mauritius National Parks and Conservation Services, however the only side we had contact with is the former.

### 2.2 Current Infrastructure

To be able to plan the hardware setup we needed to understand the layout of the area we had to work with and the constraints we would face so we met with the client and found out that there are two main sites, Bel Ombre (Southern Site) and Camp (Northern Site).

Bel Ombre:

- Heavily shaded, with little to no power.
- Situated down a steep hill about 100m away from echo aviaries (nest sites)
- No external hoppers (feeders) at this site, feeding is done at aviaries

Camp:

- has a large clearing with echo and pigeon aviaries and hoppers clustered around
- Has a solar-powered field station

Based off this, we decided to focus on developing the system for the North site due to its' greater infrastructure.

From their website [1] we also learnt that they are currently managing around 108 aviaries between these two site with an estimated 540 birds (as of April 2011).

### 2.3 Data Gathering

In addition to knowing the infrastructure of the client's project we also needed to know what data the gather on the birds and how they get it so that we could plan where our system could fit into the process.

From our first meeting we learnt that bird visits to hoppers were recorded manually by field staff dedicating around an hour a week observing the hoppers and noting the colour combinations on the bird's rings. This leads to many birds not being seen due to a range of factors, including but not limited to: human error, bird not being hungry at that time or bird visiting a different hopper that was unobserved. If a bird is unseen for a month it is presumed dead and often this is a false negative that could have been avoided by an automated system.

In addition to visit data, there is general information on birds that is stored such as nest site, their parents, release group information and much more (which will be discussed in section 3). This data is all entered manually by researchers; automating this part was outside the scope of the project.

### 2.4 Existing Data Management

To plan how we would be storing and display information we had to understand what the client used currently, to do this we spoke once again with the client.

Through meetings with the client and subsequent emails we learnt that the main form of data management is a large, shared spreadsheet called the *studbook*. The researchers make notes in their own books and at the end of a working period they would use their notes to update the studbook.

The studbook contains all the birds that have ever been part of this conservation project and as such has become very large, with over 2000 entries. An excel sheet like this lacks any ability to track changes and to display data in a clear and meaningful manner, all of which could be done by a well designed front-end system with access to database storage.

## 3. Requirements

Based on the background knowledge collected, we could now start planning our system. The first step of this was we needed draw this knowledge together into a clear list of requirements that we could follow during development and then use to evaluate our end system.

This section lists all functional and nonfunctional requirements that were requested by the client during our initial meeting and developed at the later stage during email interactions and user testing.

### 3.1 Functional requirements

1. Allow users to log in and out of the system
2. Allow users to store all information that is currently collected about birds (Example of main spreadsheet 'Stud Book' has been sent by client for us to learn what information is currently collected about birds )
3. Allow users to view all historical data collected about birds
4. Allow users to add information into the system
5. Allow users to edit information in the system
6. Allow users to delete information from the system
7. Allow users to export reports from the system

### 3.2 Nonfunctional requirements

1. **User guide** that explains how to use and maintain the system and fix errors.
2. **User Friendly GUI**
3. **Robust and durable infrastructure** (Hopper units need to be modular and easy to take off

and put on or be able to withstand disinfectant baths)

4. **Security** - System should allow different levels of access. Field staff should be able to input data but not extract reports Project coordinators should be able to input and edit data and extract reports. Higher level staff should be able to export reports but not input or edit data.
5. **Reliabilit**y - System should be accessible 24 hours a day, 7 days a week.
6. **Serviceability** - Any future modifications or services should be easily implemented and managed.
7. **Allow access to multiple users** to use system at the same time (especially important after breeding period when all data is entered in bulk)
8. **System should store information about data modifications.** Users with highest level of access should be able to see what information has been added into system and if information is edited the system should display both modified and historical data in order to be able recover information if errors are made.
9. **Data validation** - System should only allow correct data types used in certain fields

### 3.3 Added On: Nice to Haves

1. **Extended monitoring** - use a camera mounted on the feeder to take a picture of the birds as they land
2. **Remote access** - make the system accessible remotely, without the need to use a stationary workstation

## 4. Tool Decisions

### 4.1 Hardware

The minimum set of hardware required for this project was determined to be:

- Network hardware - transmitting data from feeders to server
- RFID tags & readers - uniquely identifying each bird that visits
- Server(-s) to aggregate data, process and store data, and make it available to users
- Power solution for all the above - client's site is off-grid

The client requested a network / power solution that wouldn't involve running exposed cables on the site. Therefore, we opted to implement a wireless network, where each individual device is solar-powered.

To reduce the amount of power required (which, in turn, would reduce the size and expense of the solar panels), we only considered three low-power wireless network protocols: Bluetooth Low Energy (BLE), LoraWAN and ZigBee. After researching all three and taking into account our supervisor’s advice, we chose ZigBee due to its good range / power consumption balance and ease of setup.

Although energy coming from the sun would be enough to power the devices during the day, relying solely on that would mean that the system would be unoperational at night or during extended cloudy periods. Therefore, we also purchased solar-charged batteries that would power the devices for a sufficient amount of time (at least two days) even if direct sunlight wasn’t unavailable.

When choosing the RFID readers / tags, the most important factors were size, read distance and ability to penetrate obstacles (such as casing or dust). We were also limited to passively-powered RFID tags (eg. tags that don’t use an internal battery, but rather rely on the power they receive from the reader when they come in close contact). Read distance (determined by frequency) is inversely proportional to ability to penetrate obstacles (determined by wavelength). We chose to focus on the latter, as even RFID tags with the lowest frequencies still had a read distance sufficiently large for our needs. We therefore purchased passively-powered RFID readers and tags that operate at the 134.2 KHz frequency; these have around 2 cm of read distance and a good ability to penetrate obstacles.

At the time of handing over the project, we hadn’t yet made a decision on the server hardware. This was because there is a very wide choice of servers available, and there’s no unorthodox requirements for the server that would make one product to stand out from others. Therefore, we left this choice up to the person/-s who would be deploying the system.

### 4.2 Software

There was some debate of whether to use a relational or a NoSQL database. We decided to use a relational database because there were few use cases where we would benefit from NoSQL and so we chose to go for the more traditional option, which benefited us in terms of increased familiarity and extensive documentation / online resources. We chose MySQL out of all available relational database software options, as it was the software we were most familiar / experienced with.

Although we had initially planned to write the main EchoBook system as a desktop application in Java due to familiarity, we decided to build a web application using JavaScript (using the NodeJS runtime) instead, as JavaScript projects are generally much faster to get started, and require less planning, configuration and third party tools. Furthermore, NodeJS is better suited for web applications (which was preferable to a desktop application due to platform independence) than Java due to a wide range of beginner-friendly frameworks available that allow to bootstrap web applications in a much shorter amount of time than it would require if we wrote everything by hand.

As we had relatively little experience with JavaScript and NodeJS, we wanted to use a framework that minimizes the amount of low-level code that we have to write (such as database queries, HTTP requests etc.). After some online research and discussion with our peers, we decided to use Sails, a full-stack model-view-controller (MVC) NodeJS framework that provides extensive code generation, sensible default configuration and a good level of extensibility [2]. This would allow us to focus on the business logic, rather than spending time on configuration and figuring out technical parts, with the option to implement more advanced functionality during the later stages of development if we saw the necessity.

For reading incoming bird visit data and storing it in the database, we wanted to implement a service that's separate from the main system to reduce the level of coupling and ensure that, if the main system goes down, we still record visit data. For developing this service, we chose to use the Python 3 programming language due to the *pyserial* library recommended by our supervisor who had used it in the past and found that it simplifies serial communication, allowing us to focus on handling the incoming data, rather than than worrying about the technical aspects of receiving / reading the data.

## 5. UI Design

This section includes information about how we have developed the user interface for the system and what techniques have been used to design existing UI prototype.

### 5.1 Lo-Fi 1

At the planning stage of the project, during user requirements gathering process, the project was focused on system infrastructure. Based on the current system used by the researchers in the field, it was believed that the system should have a minimalistic user interface that allows basic functionality. Based on this assumption, the first UI Lo-Fi prototype (Appendix B) was designed to understand what the system should include and decide on the basic layout. This version of UI prototype was created with the myBalsamiq tool but was not used further to work on improvements and instead we switched to Google Slides that provided us with all necessary tools to design interactive UI prototypes with greater ease and accuracy than myBalsamiq.

### 5.2 Lo-Fi 2

The second Lo-Fi prototype (Appendix C) was focused on how to display information in the system. We had begun to work with data generators to create data tables, based on information from the Stud Book sent by client, which could then be used in the prototype. The style of the system had now been changed to a two-pane app and more advanced features such as pagination and basic filters were introduced to add more user-friendly functionality to the system. It was an improvement from the first version, however, it was established that we require better understanding of users' requirements to develop this version of UI prototype further.

### 5.3 Hi-Fi 1

During development planning stage it was decided to use the Sails web application framework which uses the Vue.js framework for front-end. As the previous versions of UI prototype were not shown to the client, it was decided to create a brand new version of UI design (Appendix D) based on the initial web app layout provided by Sails. The project's aim and software development approach shifted towards User-Centred design aiming to develop a system that is largely based on user requirements and provides a user-friendly GUI. The new system development technology provided us with a great choice of interactive features that were used in the system. Once the new layout had been designed, the priority during this UI development stage was to decide on how all important information that is collected about birds is stored and displayed in the system. This version of UI prototype was tested by the client and all suggested modifications and improvements were made according to the feedback.

### 5.4 Hi-Fi 2

Final version of the UI prototype (Appendix E) includes all required functionality and suggested improvements after second user testing. The UI has been checked for the colour contrast adherence to accessibility best practices, based on Web Content Accessibility Guideline (WCAG). While it was crucial to work closely with the client during UI design process and continuously seek feedback on each aspect of the system, during the design stage it was not always possible to get feedback from the client on time. Some decisions had to be made by our team to make sure that we deliver our project on time. However, these UI related decisions did not affect the most important functionality of the system which is ability to store and view data that is already collected by researchers.

## 6. Development

### 6.1 Coordinator Service

The coordinator service was written in Python 3, as explained in section 4.2. It reads data from a serial port, where a receiving ZigBee is connected, one byte a time, until there is enough confidence that it has enough data to reconstruct an incoming RFID data packet. Although RFID packets should always be 13 bytes long and have the same start and end bytes, the possibility of network errors mean that we can't rely on packet size and start and end bytes alone, and testing showed that different packets aren't separated by any special characters in the incoming data stream. This approach ensures that data loss is minimized and the service can recover from receiving incorrectly formed data.

The stored data is then handled asynchronously to prevent blocking of other incoming data. The service attempts to extract a valid packet from the incoming data by looking for a 13-byte combination that starts with the start byte and ends with the end byte (effectively discarding any network noise before or after the packet). If found, the packet is validated using the XOR checksum included in the packet. If no valid packet is found after iterating through all the data, it is discarded.

If a valid packet is found, the service then attempts to load the relevant RFID tag record from the database using the raw RFID data received from the tag. If a record is not found, the packet is discarded, as it is assumed to be coming from a RFID tag not belonging to our system. If a record is found, it attempts to match the tag to a bird record in the database. Regardless of the outcome of the bird-matching step, the visit is then recorded in the database (along with the time of visit and the bird ID, if found), where it is then directly accessed by the main EchoBook system and shown in the 'Live View' page.

**6.2 Datastore**

As Mentioned in section 4.2, our database was built on MySQL. After several iterations of making a database diagram, implementing it, testing it and tweaking it, we came up with the final version of the database, version 6 (Appendix A), which provided all the needed functionality put forward in the requirements gathering.

To meet these requirements, the final system had one database containing nine tables. To manage the data on the birds that researchers have to enter manually (see last paragraph of section 2.3) we had the Main Bird table that held most of the information, some aspects were separated to allow for long term historical data tracking, a previously unavailable feature). These were the nest sites (nest site itself being store in *nestsite* and information on birds living in them in *birdnest*) and the bird condition (in *birdcondition*). Having these as separate tables allows a new entry to be made whenever there is a change (i.e. when a bird moves to a new nest, or its condition is changed) and thus preserving the old one which can then be accessed through our front-end.

The remaining five tables were regarding functionality introduced by our system, the most anticipated ones being the RFIDTag and Visit tables which would track the assignment of one of the introduced RFID rings and a bird so that when the Coordinator service reads the ring the visit can be recorded for the correct bird. Next, we had the Users table which stores the login details and permissions associated with the account for the system to then use to permit / deny access to features as required.

The lock table was added to provide mutual exclusion when editing birds, nest sites and users and thereby prevent multiple researchers trying to alter the same resource and cause issues where one person's work maybe overridden.

Finally we had the changelog, this table provided the structure needed to provide on of the main things that the client’s current system lacked, tracking changes. With this table and the front-end functionality the system can now track which users make which changes down to individual fields and provided accountability and a frame of reference to restore data in the rare case where someone deliberately tampers with data, or on the more likely case where an incorrect value is entered by a less experienced user.

## 6.3 Front-End Interface

For the Front-end interface of the application, we used many different technologies and frameworks. All the pages were created in EJS (Embedded JavaScript), which is a templating language that generates HTML markup with plain JavaScript. For the styling of the pages Sails has Bootstrap 4 preinstalled, which is a popular front-end design framework. We used it for styling of all the pages and modified it where needed. We also added our own custom CSS styling to the pages to match the requirements. The Vue.js framework was used for the front-end as a whole to aid development and improve the functionality of the interface.

Throughout the development process of the front-end interface we closely followed the UI designs and requirements specified by the client. We were in frequent contact with our client to understand the users of the system and their needs and requirements from the system.

The contents of the pages are loaded dynamically. For better user experience, we made sure that each page of the website is consistent and follow the same style and structures. For example, header and footer serves the same functionality among different pages, and are kept consistent throughout the website. Additionally, we used modals instead of creating new pages for different functionalities of the website (such as registering bird, editing bird, registering user). By using modals we reduced the number of pages which makes the website easier to navigate for users.

We made the website more user friendly by not displaying all the information on one page, instead we used pagination where the user can decide how much data they want to see on a single page. In addition to that, on the Bird Table page where a broad range of data is available, we added the functionality to remove/add columns so the users can select the data they want to see.

## 7. Testing

### 7.1 What was done

During development System testing was done continuously as new components were added. We used the functional requirements and user stories as a base and came up with our own other scenarios as we walked through the system. The issue with this informal style of testing we adopted is that it ended up undocumented, if a component worked as expected it was added and if not it was reworked until it did, as such, there was no formal record of the testing performed as it was implicit.

### 7.3 What could have been done

Although we were able to get user input on the system at a few different stages and this provided us with knowledge of changes the client wanted, we never ran a full User Acceptance Test.

This was mainly down to the time constraint we had with both the completion of the system and scheduling time with the client, based of the input we received from the demonstrations at intermediate stages, this is certainly an aspect that could have benefited our system.

The other form of testing we would have liked to have utilised is unit / integration testing to test the data handling areas of the system. This was an oversight on our end as the development time exceeded expectations and we lacked the time to write these tests. We had SQL scripts to populate the database with test data and we then manually used the database software and the NodeJS debugger to test that the queries produced  the values we expected. However, automation would have helped us ensure all tests were run each time, and not just the tests relevant to a particular subsection of the system.

## 8. Deployment

As the client's site doesn't have an internet connection, any deployment would have to happen either before the server is delivered to the client, or with all dependencies (such as libraries & runtimes) bundled for offline installation.

We prepared a proof-of-concept offline deployment method, with instructions provided in the user manual, as well as *readme* files in the Deployment folder. This installs the main EchoBook system, the coordinator service (along with the necessary software dependencies) and prepares the database for first use by creating the necessary tables and inserting a single user to first access the system.

Database server installation isn't included in the deployment for the following reasons:

- It's an action unlikely to be taken more than once
- No specific MySQL version is required
- The person deploying might want more control over configuration than we would be able to provide if we automated this step

The method was tested only on 64-bit Ubuntu 18.04 as it is proof-of-concept only and will likely need some refinements before being put to practice, such as automatic recovery in case of a crash. It should, in theory, work on any Linux machine with Bash available.

## 9. Reflection

This section will look back at the requirements defined in section 3 and determine whether and how we have achieved them.

### 9.1 Functional requirements

1. Allow users to log in and out of the system - **done**
2. Allow users to store all information that is currently collected about birds - **done** (Register Bird allows the user to enter all this information)
3. Allow users to view all historical data collected about birds - **done** (Bird Table and Single Bird View lets the user view a summary of all birds, and detailed information about a single bird, respectively)
4. Allow users to add information into the system - **done** (the user can add information on birds, nest sites and rings)
5. Allow users to edit information in the system - **done** (the user can add information on birds, nest sites and rings)
6. Allow users to delete information from the system - **partially done** (the user can delete nest sites and rings that aren't assigned to birds, but not bird records)
7. Allow users to export reports from the system - **done** (the user can generate a spreadsheet report by

using the 'Export' button in Bird Table)

### 9.2 Nonfunctional requirements

1. **User guide - done**
2. **User Friendly GUI - done** (possible improvements discussed in section 11)
3. **Robust and durable infrastructure - partially done** (these factors were taking into account when planning out the infrastructure, but field tests would need to be conducted to be certain)
4. **Security - done** (access permissions are set when creating users and can be edited / revoked at any time by an administrator. These are verified in both front-end and back-end)
5. **Reliability - partially done** (there is nothing to suggest that the system wouldn't be operational 24/7, but field tests would need to be conducted to be certain. Also, to some extent, dependant on deployment discussed in section 8)
6. **Serviceability - done** (the codebase is modular, well structured and commented throughout. It follows standard Sails framework practices)
7. **Allow access to multiple users** to use system at the same time - **done** (user sessions are managed by the *express-session* middleware. Concurrent write access to data is controlled via a mutual lock that prevents multiple users from editing the same data at the same time)
8. **System should store information about data modifications - done** (can be viewed in User Activity Monitor, all create/edit/delete actions are recorded with the relevant data changes)
9. **Data validation - done** (where an input field is expecting a certain type of data, it is live validated and the user alerted and prevented from submitting if the data is invalid)

### 9.3 Added On: Nice to Haves

1. **Extended monitoring - not done** (wasn't done due to time constraints and low priority)
2. **Remote access - partially done** (as the system is a web application, it can be accessed over a wireless network; although there currently isn't one on client's site, it could be installed without much hassle and without requiring any changes to our system)

The system fully meets the majority of requirements set out in the planning stage; where a requirement was met only partially, this was largely due to oversights in planning or lack of development time).

## 10. Conclusion

In conclusion, we have designed and built a working system that tracks visits and manages data on Echo Parakeets. Its' performance has been demonstrated to the client on multiple occasions and has been approved by the client as a useful system that met the original requirements.

This project initially drew our attention as a chance to develop something that could make an actual difference, and as the project comes to an end we believe we have achieved this as our effort has provided the groundwork to help researchers spend time on more important areas and help further conservation efforts faster.

If future groups aim to design a system for a specific client, the most important advice to take away from this is the client should always be your first thought. Although it can slow down the creation process (waiting for responses and setting up meetings), without the input we received from the client we can clearly see points in our design and  development where we would have diverged from the client's vision and we would have ended with an unwanted system. However, by focusing on the client we avoided this and even spotted details we would have otherwise missed that were vital to what the system was intended for.

## 11. Future Work

There are many avenues that can be taken to further the work done here, in this section we will take a look at possible improvements that we view as the most important.

- **Support of API mode**

Currently, the coordinator service can only read data received in XBee AT mode (which was used due to its lower complexity). API mode support could be added as it allows for greater extensibility and more features, such as identifying when a network device has gone offline and seeing exactly which feeder the read came from and therefore be able to track where birds are feeding in addition to when; this wasn't required by the client but it could be a useful addition in understanding more about the echo population.

- **User Efficiency - UI Tweaks**

There were several small design tweaks we identified that could be implemented in future versions to improve user experience in a two distinct ways.

First type of improvement deals with reducing the number of clicks required to navigate to certain areas by refactoring the modals to individual controllers and then linking to them from where they are needed, some examples of this would be getting to a single bird view from the live view or assigning a ring to a bird directly from the manage ring section.

The second type deals with user control over data. In its current state the system allows some basic filtering via word-contains searches, range based date searches and exact text matches. Filters are not available on all fields however, and adding this would allow greater control to the user in seeing the data that they want, and would equal the level of filtering they had in the legacy system (which is still possible when our exporting feature is used to generate an excel sheet similar to the client's current spreadsheet).

- **Improve back-end efficiency**

Currently, the way in which data is retrieved and processed in some areas of the system is very inefficient due to certain features not being available in the 3rd Party query handler and it resulting in us having to write quick workarounds to avoid spending extensive development time on rewriting the queries to use an alternative such as SQL native queries. This causes degraded performance when viewing or filtering birds which is likely to worsen when more bird records are added.

## Acknowledgements

We would like to thank our project supervisor Daniel Knox for the advice and assistance throughout the project and thank Debbie Fogell, our liaison with the client, for answering all our questions, giving feedback on our designs throughout the project and providing us with the knowledge we needed to understand the user and to tailor the system to suit their needs.

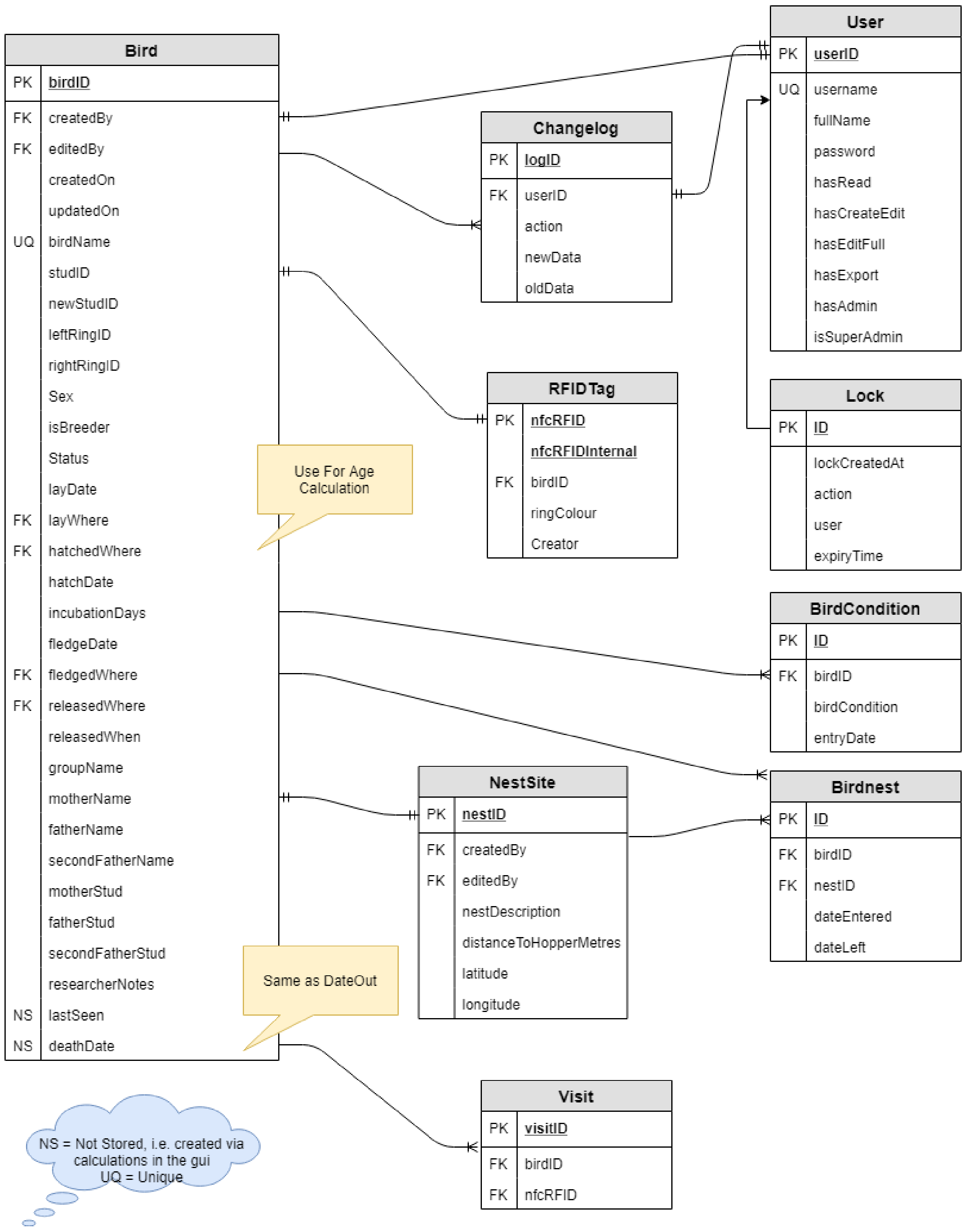
## Bibliography

[1] Mauritian-wildlife.org. (2019). *Welcome to the Mauritian Wildlife Foundation (MWF) - In The Field - Mauritius - Echo Parakeet.* [online] Available at: http://www.mauritian-wildlife.org/application/index.php?tpid=30&tcid=32 [Accessed 27 Mar. 2019].

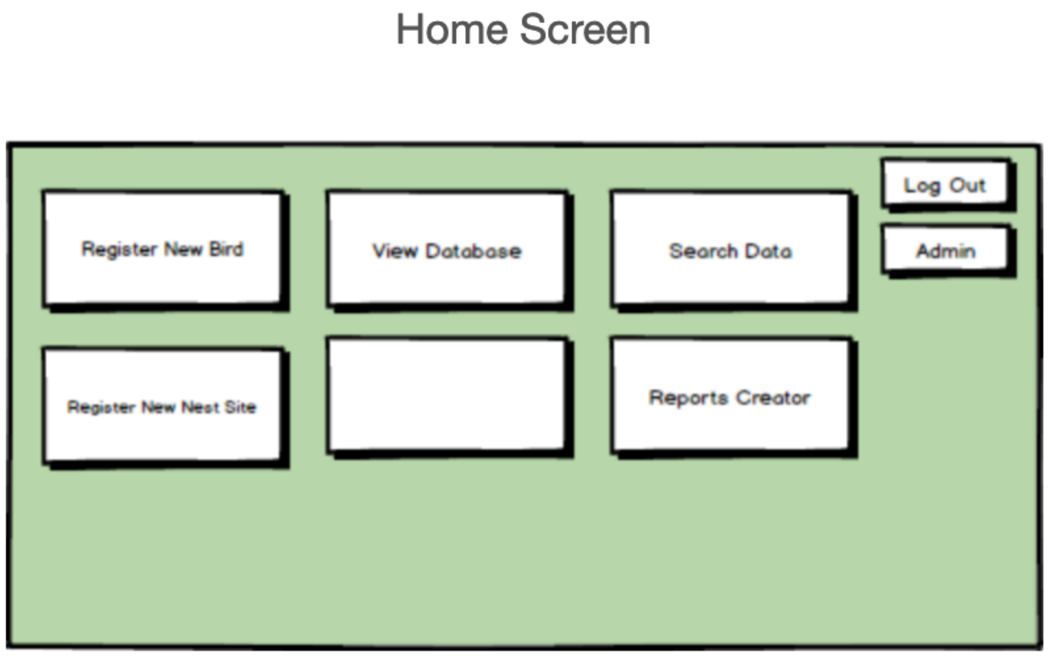
[2] Sails.js. (2019) | Realtime MVC Framework for Node.js [online] Available at: https://sailsjs.com [Accessed 27 Mar. 2019].

# Appendices

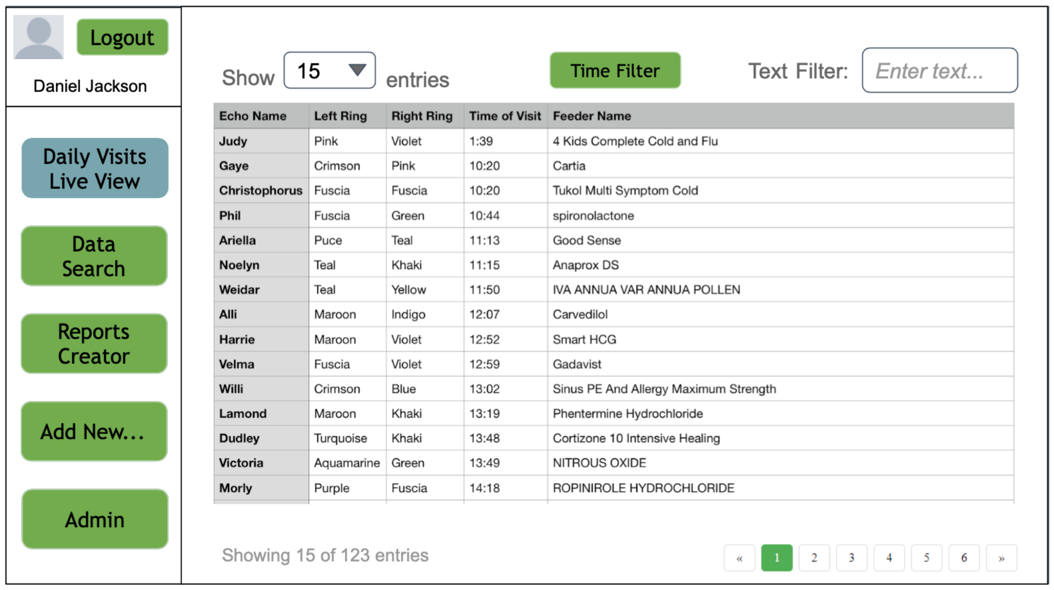
## Appendix A: Version 6 of database diagram



## Appendix B : Lo-Fi prototype Version 1



## Appendix C: Lo-Fi prototype Version 2

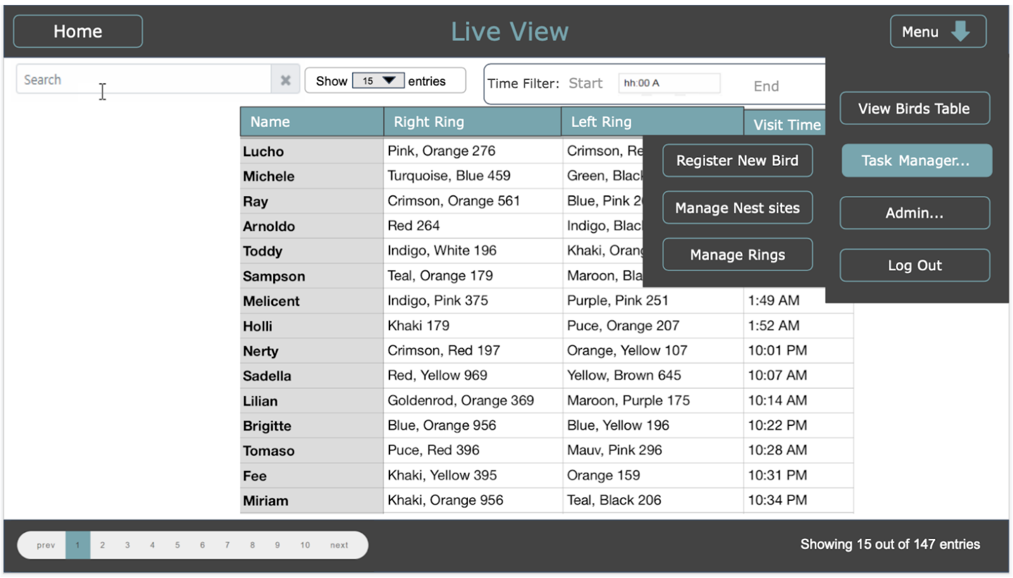


| Echo Name | Left Ring | Right Ring | Time of Visit | Feeder Name |
|---|---|---|---|---|
| Judy | Pink | Violet | 1:39 | 4 Kids Complete Cold and Flu |
| Gaye | Crimson | Pink | 10:20 | Cartia |
| Christophorus | Fuscia | Fuscia | 10:20 | Tukol Multi Symptom Cold |
| Phil | Fuscia | Green | 10:44 | spironolactone |
| Ariella | Puce | Teal | 11:13 | Good Sense |
| Noelyn | Teal | Khaki | 11:15 | Anaprox DS |
| Weidar | Teal | Yellow | 11:50 | IVA ANNUA VAR ANNUA POLLEN |
| Alli | Maroon | Indigo | 12:07 | Carvedilol |
| Harrie | Maroon | Violet | 12:52 | Smart HCG |
| Velma | Fuscia | Violet | 12:59 | Gadavist |
| Willi | Crimson | Blue | 13:02 | Sinus PE And Allergy Maximum Strength |
| Lamond | Maroon | Khaki | 13:19 | Phentermine Hydrochloride |
| Dudley | Turquoise | Khaki | 13:48 | Cortizone 10 Intensive Healing |
| Victoria | Aquamarine | Green | 13:49 | NITROUS OXIDE |
| Morly | Purple | Fuscia | 14:18 | ROPINIROLE HYDROCHLORIDE |

Showing 15 of 123 entries

« 1 2 3 4 5 6 »

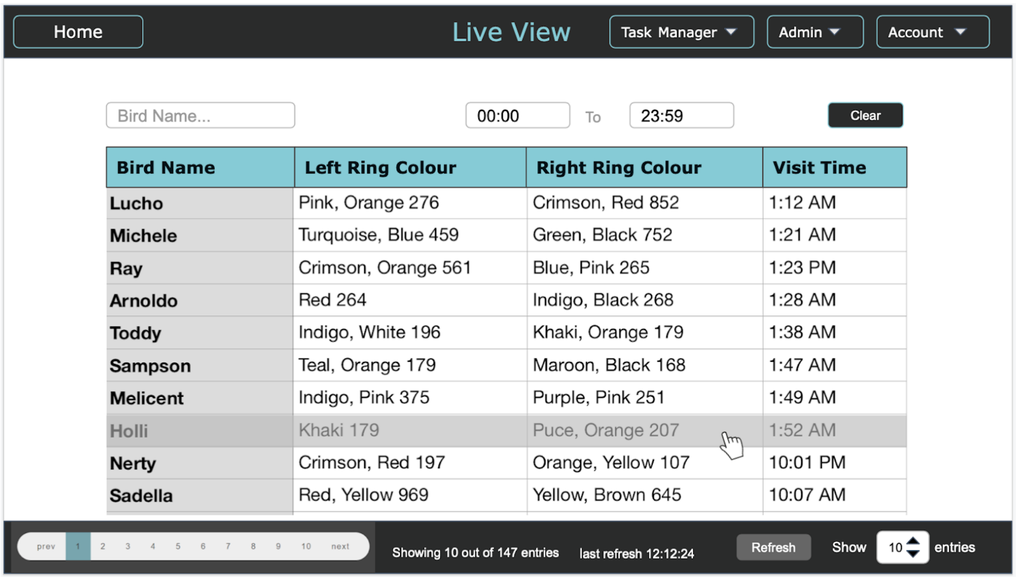
## Appendix D: Hi-Fi prototype Version 1



| Name | Right Ring | Left Ring | Visit Time |
|---|---|---|---|
| Lucho | Pink, Orange 276 | Crimson, Re | |
| Michele | Turquoise, Blue 459 | Green, Black | |
| Ray | Crimson, Orange 561 | Blue, Pink 2 | |
| Arnoldo | Red 264 | Indigo, Black | |
| Toddy | Indigo, White 196 | Khaki, Orang | |
| Sampson | Teal, Orange 179 | Maroon, Bla | |
| Melicent | Indigo, Pink 375 | Purple, Pink 251 | 1:49 AM |
| Holli | Khaki 179 | Puce, Orange 207 | 1:52 AM |
| Nerty | Crimson, Red 197 | Orange, Yellow 107 | 10:01 PM |
| Sadella | Red, Yellow 969 | Yellow, Brown 645 | 10:07 AM |
| Lilian | Goldenrod, Orange 369 | Maroon, Purple 175 | 10:14 AM |
| Brigitte | Blue, Orange 956 | Blue, Yellow 196 | 10:22 PM |
| Tomaso | Puce, Red 396 | Mauv, Pink 296 | 10:28 AM |
| Fee | Khaki, Yellow 395 | Orange 159 | 10:31 PM |
| Miriam | Khaki, Orange 956 | Teal, Black 206 | 10:34 PM |

Showing 15 out of 147 entries

## Appendix E: Hi-Fi Version2



| Bird Name | Left Ring Colour | Right Ring Colour | Visit Time |
|---|---|---|---|
| Lucho | Pink, Orange 276 | Crimson, Red 852 | 1:12 AM |
| Michele | Turquoise, Blue 459 | Green, Black 752 | 1:21 AM |
| Ray | Crimson, Orange 561 | Blue, Pink 265 | 1:23 PM |
| Arnoldo | Red 264 | Indigo, Black 268 | 1:28 AM |
| Toddy | Indigo, White 196 | Khaki, Orange 179 | 1:38 AM |
| Sampson | Teal, Orange 179 | Maroon, Black 168 | 1:47 AM |
| Melicent | Indigo, Pink 375 | Purple, Pink 251 | 1:49 AM |
| Holli | Khaki 179 | Puce, Orange 207 | 1:52 AM |
| Nerty | Crimson, Red 197 | Orange, Yellow 107 | 10:01 PM |
| Sadella | Red, Yellow 969 | Yellow, Brown 645 | 10:07 AM |

Showing 10 out of 147 entries last refresh 12:12:24