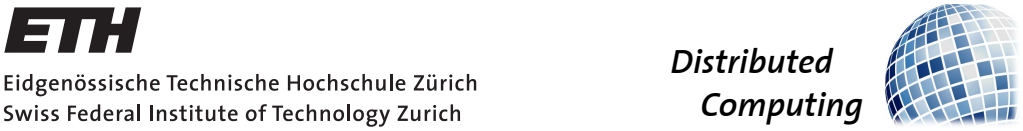


# Is It Schroedinger's Cat?

Semester Thesis

Laura Peer

lpeer@ee.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

**Supervisors:**
Jara Uitto, Barbara Keller
Prof. Dr. Roger Wattenhofer

March 27, 2014

So this is where we part, My Friend,
And you'll run on, around the bend.
Gone from sight, but not from mind,
new pleasures there you'll surely find.

I will go on; I'll find the strength,
Life measures quality, not its length.
One long embrace before you leave,
Share one last look, before I grieve.

There are others, that much is true,
But they be they, and they aren't you.
And I, fair, impartial, or so I thought,
Will remember well all you've taught.

Your place I'll hold, you will be missed,
The fur I stroked, the nose I kissed.
And as you journey to your final rest,
Take with you this...I loved you best.

-Jim Willis

# Acknowledgements

I would like to thank my supervisors Jara Uitto and Barbara Keller for their consistent and helpful support.
Additionally, I would like to thank but also mostly apologize to all the animals whose peaceful naps I've disrupted with my scanning device.
Lastly, I would like to express my thanks to Professor Roger Wattenhofer for the opportunity to work on this interesting project at the Distributed Computing Group.

# Abstract

More and more smartphones are sold and used every day. The countless sensors they carry open up new and exciting possibilities to reach and inspire users.
In this thesis we design and implement a system that enables faster and easier recovery of lost animals implanted with RFID microchips. We develop an appealing website that allows users to register themselves and their animals on our system. We develop an Android application that can read animal RFID microchips out in the field and that keeps track of the exact GPS location of the smartphone. When a lost animal is scanned, the Android application connects to our web application, which records the incident and immediately informs the owner of the lost animal by providing an exact location and the details of the finder.
The result of this thesis is a fully developed software system, which, if widely used, could prove to be more effective than current animal recovery methods. We make use of the existing and deployed hardware, the RFID microchips, without having to disseminate our own means of animal identification.

# Contents

Chapter 1

# Introduction

There are numerous animal microchip tagging and registering systems currently implemented. In 1996, Canada, Europe, Asia and Australia agree upon the two ISO standards 11784 and 11785 [13]. ISO 11784 defines the structure of the information content on the RFID microchip. ISO 11785 defines the communication protocol in place between the RFID reader and the implanted chip. Some countries, for example, Australia, Switzerland and the UK, have even made cat and dog microchip implanting a legal requirement.
Every microchip has a unique ID stored in memory, which is tied to the animal's owner in a registry database. Without the registry, ownership recovery is useless, because nobody knows which ID belongs to which animal or rather owner. Some countries or regions maintain a centralized database to ensure uniqueness of the chipped animals. Some registries are offering their services for free, some registries are maintained by the companies that produce the implantable microchips for a profit. Pet Microchip Lookup [3] offers a broader ID lookup among a selection of animal registries. It is important to note that it is up to the owner to update his entry in the registry upon an address or phone number change.
In Switzerland, a company called Animal Identity Service (ANIS) [14] provides the microchip implant hardware and offers a maintained database registry where the owner information is stored in conjunction with the unique ID of the microchip. A one-time installation fee and a yearly database cost is charged by the animal's regular veterinary office, where the chip is implanted.
In the USA, the American National Standards Institute (ANSI) [4] endorses the use of the aforementioned standards, but, according to the American Veterinary Medical Association(AVMA) [17], as of July 2013, 98% of microchipped pets in the USA are implanted with devices that don't comply with ISO standards 11784 and 11785. 24PetWatch [1], HomeAgain [12] and FriendChip [6], for example, are companies that distribute microchips which operate on 125KHz instead of the ISO standard's mandated 134.2KHz. FriendChip uses encryption while microchips produced by 24PetWatch and HomeAgain store the data in the clear. These differences between microchips make life quite difficult for animal shelters, because they need be prepared to scan for all possible chip variations in order to successfully recover the identity of an animal, which requires the purchase of

numerous expensive RFID microchip scanners. In 2004, an 8-month-old Pit Bull Terrier was euthanized in Virginia because the shelter's 125KHz scanner failed to detect the dog's ISO compatible chip [19].
Lord et al. [31] state that the return-to-owner rate for cats is 20 times higher if a microchip was implanted. Dogs are 2.5 times more likely to be recovered if they were implanted with a microchip. The difference in factor between cats and dogs probably originates from the fact that more dogs wear identifying collar tags and that there is no need for shelters to scan for a microchip if the identity of the animal is already found. Owner contacts where found of 72.7% of microchipped animals. The remaining owners had either forgotten to sign up for a database, had signed up for an unknown database or had neglected to update their current location.
These numbers make a good case for the benefits of microchipping animals, however, the current situation is not ideal. The scattered databases make animal recovery bothersome. Different microchip technologies and expensive microchip readers make it harder than necessary to identify pet animals, sometimes with tragic results.
We present a system that addresses these issues. We provide an appealing web interface where users can register themselves and their animals in our system. Our database is unified. It does not discriminate between manufacturer brand or the geographic location of the animal or owner. By designing our own RFID reader device that seamlessly integrates into the workflow of our Android application, we provide a notably cheaper and more flexible alternative that allows users to scan animals in the field instead of having to bring them to a shelter or police station. Our work utilizes existing and deployed technology, as in animal RFID implants. Their global acceptance and usage is increasing, which makes our potential user group grow. We make use of the number of smartphone sensors the users carry with them at all times. Since smartphones are potentially always connected to the Internet, we provide a push notification service that informs the owner immediately when one of his animals is found. In addition, the smartphone can calculate the exact location of the found animal. Our system provides an interactive map where the owner can see exactly where his animal was located in an appealing way. The finder of the animal has the option to stay anonymous. This may motivate users to at least communicate the location of the animal to his owners in hit-and-run scenarios.

Chapter 2 lists current related work in the field of animal microchipping. Chapter 3 outlines the methods and progress of our thesis and lists the encountered problems and design decisions taken while developing our system. Chapter 4 discusses the results, Chapter 5 states unsolved issues and describes potential improvements and Chapter 6 provides a brief summary of the thesis.

Chapter 2

# Related Work

There is an extensive number of corporations all over the world that produce and provide microchip implants for sale and maintain animal databases where their customers can choose to register their animals for a yearly fee.
24PetWatch [1], HomeAgain [12] and FriendChip [6] provide microchips that work on a frequency of 125KHz. FriendChip even uses encrypted microchips. AKC Companion Animal Recovery [2] provides microchips that work on a frequency of 128KHz. ANIS [14] and Bayer ResQ [7] comply with the ISO standard that is widely accepted and dictates the use of the frequency 134.2KHz for their chip implants. Similarly to our system, these microchip manufacturers offer information storage for animal owners, but what they lack is the quick and painless localization and notification of users. Because there are a number of manufacturers, each of which has his own owner database, it is unclear where exactly the owner record is stored. Our system offers a universal storage that does not depend on chip brand, chip frequency, or even country of origin.
A less invasive route is taken by the company Smartphone Pet ID Tag [15], which sells collar attachment tags that feature a URL and a QR code. Scanning the QR code with a smartphone using any QR reader software outputs the owner's contact information. The additional link on the tag enables users without smartphones to manually initialize the retrieval on the listed website. PetHub [18], an award-winning startup, goes one step further and adds an NFC microchip into the collar tag in addition to the written URL and the QR code. When read, a notification goes out to the pet owner and a map is sent with the current location if the owner pays 19$ per year for the premium service. Free accounts are also contacted, but they don't receive maps or immediate push notifications. Compared to our system, these systems identify the animals in a less permanent fashion. Collars can fall off or be taken off and are thus a less persistent feature when compared to using microchip implants. The advantage, however, is that the tag is clearly visible when attached and shows the name of the company, which is sure to return a record for the found animal compared to the only possibly existent, invisible microchip with unknown operating frequency and unknown manufacturer in the animal microchip implant case.
A project called rfiDOG [22] was presented at GeekCon2013. It aims to create

a credit card size device that replaces expensive RFID readers and connects to a smartphone using Bluetooth, which searches Israel's national dog database for a match[1]. Investigating further proves difficult because of the language barrier and the fact that all links on the project description page lead to a dead end. This project is quite comparable to ours, except that it does not provide means to directly reach the owner of the lost animal. Instead it puts the burden of contacting the owner to the user of the smartphone. In addition, it does not make use of the smartphone GPS sensor.
Since NFC technology does not support the low frequencies in use in animal implant microchips, a product called RFIDler [21] which was successfully funded on kickstarter.com and estimated to be completed by February 2014, could potentially enable reading animal implant microchips. The project aims to produce an embedded hardware that reads various RFID microchip protocols in a frequency range of 125-134KHz. Additionally, we find an already produced version of an embedded RFID reader chip that supports the same frequency range in an online store called Priority1Design [20]. Both these products provide the functionality of reading animal microchips at a lower price than the commercially available animal microchip RFID readers and could provide useful additions to our project.

[1] Interestingly, we stumbled upon this project after contemplating whether to call our project rfiDOG as well.

Chapter 3

# Design

This Chapter describes the process of building the system, our design decisions along the way and sheds some light on the encountered difficulties. To facilitate the recovery of lost animals, we implement a web application, described in Section 3.1, which accepts and performs tasks upon user request, for example registering an animal in a central database or notifying the system of a newly found animal. To make the system flexible we develop an Android application, described in Section 3.2, which uses the GPS sensor to calculate the location of the lost animal and uses the mobile network access of the smartphone to communicate with the web application. Section 3.3 describes the scanning device we design to work in conjunction with the Android application, which reads the implanted microchip data. Section 3.4 describes the Android push notification scheme we implement to inform the animal's owner immediately when their animal is found.

## 3.1 The Web Application

Our web application is developed using the Django Framework [16] for a number of reasons. First and foremost we have an affinity for Python, the language the Django Web Framework is written in. Python is a very readable and highly object-oriented language that works well across all platforms. Secondly, Django already comes with a number of well-documented built-in schemes. User authentication, session management, HTML template generation and object-relational mapping allow for a very quick and painless application development process.

### 3.1.1 The Database Model

Django ships with a default User model and a backend that handles user authentication and authorization. Every registered user has a unique private key or id and authenticates himself using his unique username and his password. Specific areas of the web application require certain access permissions. These restricted areas are governed by the authorization backend, which determines if

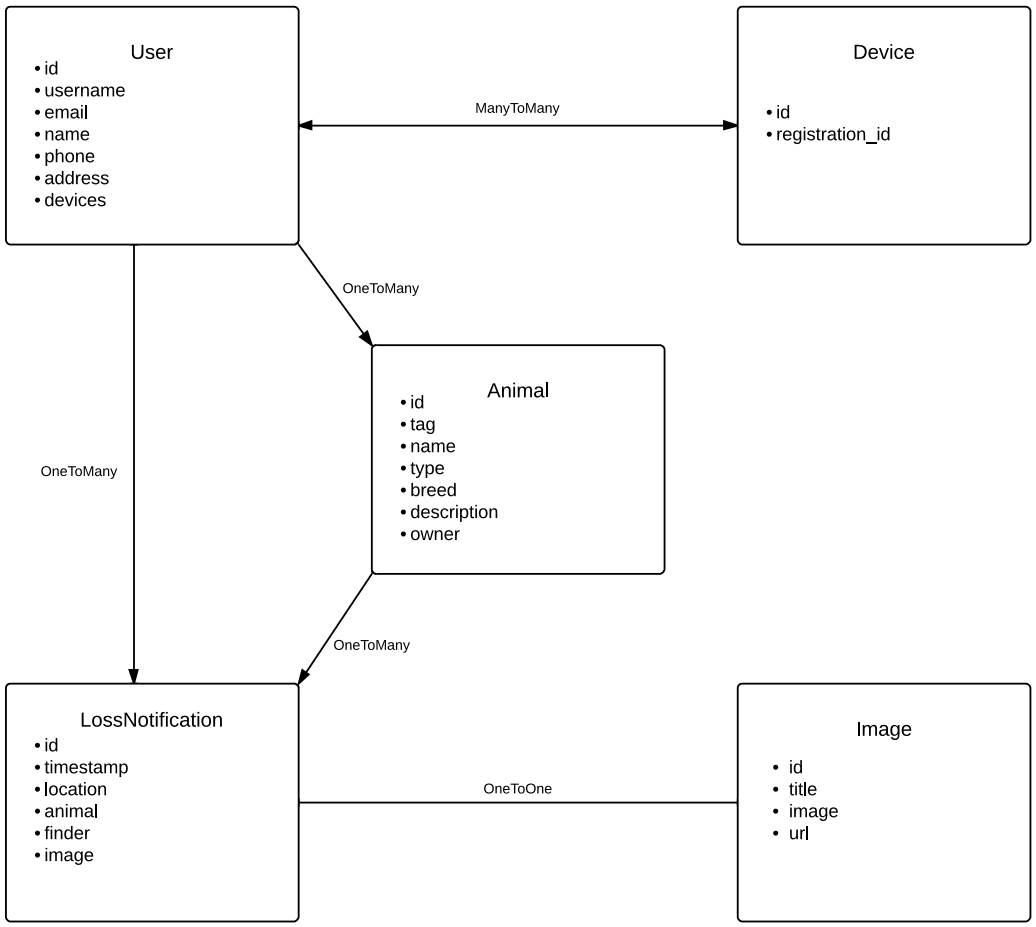


Figure 3.1: Web Application Database Model

the logged-in user is allowed access. We keep these nice mechanisms in place and add a number of fields to the User object, for example, an email address, phone number and a postal address.
In addition to the User object, we define an Animal object in the database. An Animal object stores a unique microchip tag ID, a name, some details and a link to exactly one User object, the owner. This relationship, seen in Figure 3.1, is called a OneToMany relationship. While one user is allowed to own a number of animals, one animal is only allowed to have exactly one owner in our system. Every time a user adds an animal into the system, a new Animal object is created, added to the database and linked to the User object.
Users who own Android smartphones can usually receive push notifications. We therefore add a Device object for every user who uses the Android application.

The Device object contains a unique device id and links to one or more User objects. The Device and User objects are in a ManyToMany relationship, as seen in Figure 3.1. One phone may thus be used by a number of users and one user may own a number of smartphones or tablets that are using our software. For the case when an animal is found we define a special database object called LossNotification. The LossNotification object holds an id, a timestamp and a location field. It specifically links back to exactly one Animal object and exactly one User object. The OneToMany relationships occur because the action of scanning and reporting a found animal on our system consists of exactly one finder and one lost animal. However, a user may find lost animals frequently or one animal could disappear and be found more than once causing a number of LossNotification objects to be created which all tie to the same Animal or User object. Every time a LossNotification object is generated, the web application downloads a location map from Google Static Maps API [23] and ties the Image object which contains the downloaded map to the LossNotification object in a OneToOne relationship.
If a user decides to delete his profile from our system, we implement a function which removes all the owned animals from the database. All the LossNotification objects that tie to the deleted user profile are updated to point to an anonymous user. All LossNotification objects that show where an owned and now deleted animal was found are deleted and all the user's Device objects are deleted from the system if no other user is tied to them.

### 3.1.2 The Website

We develop an appealing website to improve user experience. The website enables users to interact with the web application. Users can register, manage their account information, can add, edit or delete their animals and view any of their lost or found animals. The website structure is seen in Figure 3.2. Initially, as seen in the gray states, the user can either login, register or recover a forgotten password. After registering or recovering the password, an email is sent to the newly created or already existing user. By clicking on the link contained in the email, the user is activated in the system and is redirected to the login screen. The email activation link is set to time out after a period of time, which should protect the database from becoming cluttered with inactive profiles. Requiring users to have a valid and working email address initially increases the chances that the user is reached if one of his animals is recovered, because the system sends out an email to the owner when it receives a new animal loss notification. There are three main pages for logged in users. The animal page lists all registered animals for the logged in user account. The user page shows user details and allows the logged in user to change his information or password. Finally, the notification page lists all lost or found animals. If a user is interested to see where his most recently lost animal was found, he can click the topmost lost animal

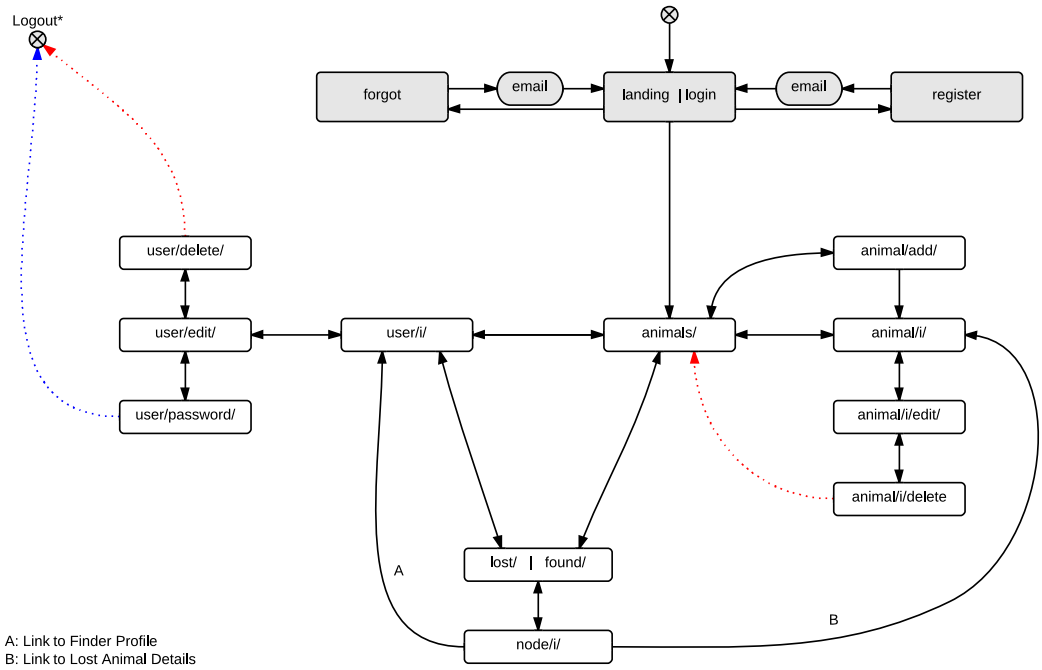


Figure 3.2: Website Structure

notice, which redirects him to the resource at *node/i/*. An example view can be seen in Figure 3.3. The page shows a map with a pin at the position where the animal was found. To generate the map we use Google Maps JavaScript API v3 [11]. If the marker is clicked, a box appears on the map next to the pin. The box shows a link to the finder profile and to the lost animal page and lists the finder's contact details, the time when it was found and a comment field. Contacting the finder to organize an exchange is very simple from this point on.

### 3.1.3 The Web Application API

The website from Section 3.1.2 returns HTML pages for every view the user wishes to access. We implement an API to allow the Android software more efficient access to the web application. Essentially, our API consists of a number of entry points located on specific uris, see Figure 3.4, where the web application either accepts or returns data in JSON format. On every request it receives, the web application determines if the user has the necessary permissions to access the data. If the user is invalid or does not have the necessary permissions, a HTTP status code 404 is returned. Every entry point allows certain HTTP methods. For example, the uri *node/lost/* can only be accessed using a GET request. When

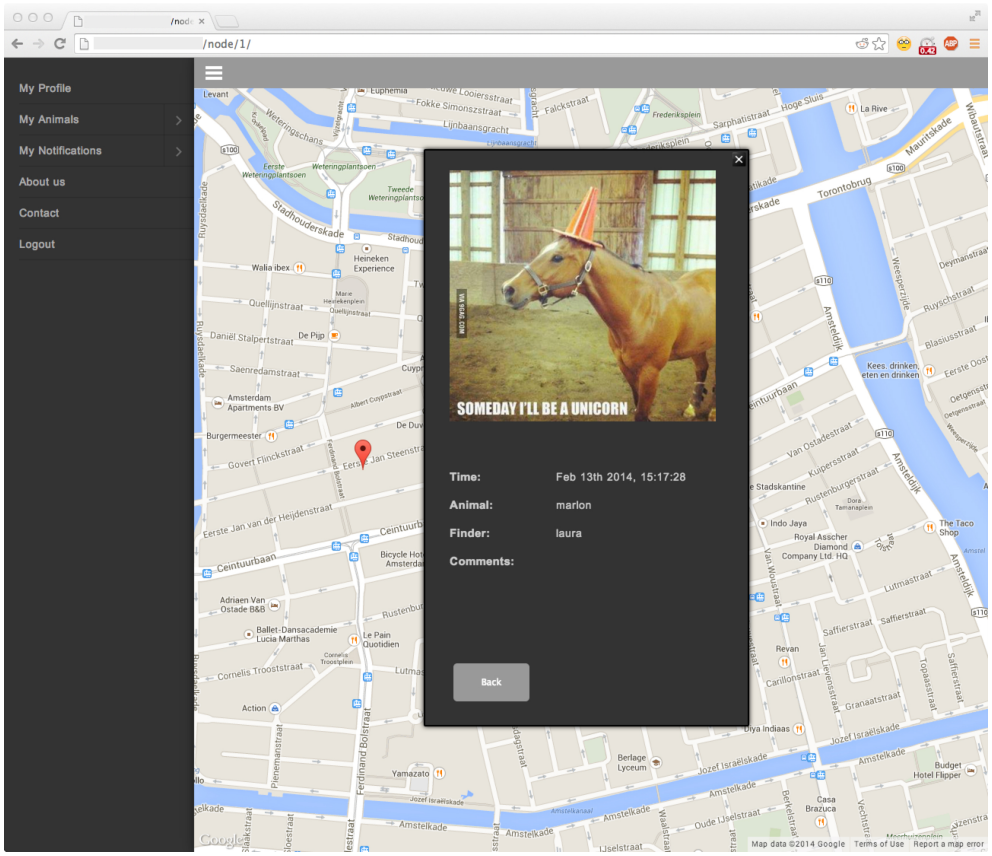


Figure 3.3: Example Lost Animal Website Page

successfully requested, it returns an encapsulated JSON object which contains all lost animal records. Any other request method for the aforementioned uri will result in a HTTP status code 400.
HTTP is stateless by nature, which means that it does not attempt to keep track of subsequent requests. This has the effect that the user has to provide some form of credentials for every request. The simplest approach would be to send username and password with every request, but that is not very secure. We thus implement token based authentication. A user who wishes to access the API first sends a POST request to uri *token/fetch/*. This request contains the username and password in the payload. If the credentials are valid the web application returns a token. This token is valid until the user logs out. The client sends the authentication token on every subsequent request to the API. We use a scheme where the token is sent using HTTP's Authorization header. The web application looks for this header on every incoming request and validates the token. Section 3.2 sees the API in action when it is passing and receiving data from the Android application.

```
url(r'^users/$', user_list ,name='api_user_list'),
# get logged in user details
url(r'^user/profile/$', logged_in_user_detail, name='api_logged_in_user_detail'),
url(r'^user/password/$', api_change_password, name='api_change_password'),
url(r'^user/evaluate/$',evaluate_login,name='evaluate_login'),
url(r'^user/logout/$',android_logout,name='android_logout'),
# just for users that are not I
url(r'^user/(?P<pk>[0-9]+)/$', user_detail, name='api_user_detail'),

url(r'^animals/(?P<pk>[0-9]+)/$',animal_list, name='api_animal_list'),
url(r'^animal/(?P<ak>[0-9]+)/$',animal_detail, name='api_animal_detail'),

url(r'^node/found',notification_list_finder,name='api_list_found_notifications'),
url(r'^node/lost',notification_list_owner,name='api_list_lost_notifications'),

url(r'^node/(?P<pk>[0-9]+)/$', notification_detail ,name='api_notification_detail'),

#url(r'^(?P<tag>[a-zA-Z0-9_]+)/$',create_notification,name="api_create_notification"),
url(r'^notify/$',create_notification,name="api_create_notification"),
url(r'^test/$', testGCM ,name='testGCM'),
url(r'^gcm/register/$', testGCMregistration ,name='registerGCM'),
```

Figure 3.4: API URIs

## 3.2 The Android Application

Figure 3.5 illustrates the possible state transitions within the Android application. The white boxes represent states within the application itself, the pink boxes represent entry points to the web application API described in Section 3.1.3.

When the user launches the application, he is presented with a SplashScreen, which redirects to the next state after a timeout. Next, the application checks its Google Cloud Messaging status. We shall refer to Section 3.4 for any detailed discussion on Google Cloud Messaging, the system we use to implement smartphone push notifications.

After the Registration activity, the Android application moves on to the Login activity. First, the application checks its local database for the existence of a authentication token. You may recall from reading Section 3.1.3 that we use a token as authentication for the API. If a token is found in the local database, the application validates the token at entry point *token/validate/*. If the validation fails, the token entry is removed from the application database and the user is shown a login screen, which can be seen in Figure 3.6. If the validation is successful or the user has entered valid credentials, the application redirects the user to the main application screen, seen in the upper right screenshot of Figure 3.6.

In the main application screen, the user is presented with a logout button on the top and a menu containing three icons on the bottom of the screen. The icon on the left leads to the scan menu view, the icon in the middle leads to the notification list or history view and the icon on the right leads to the user profile view.

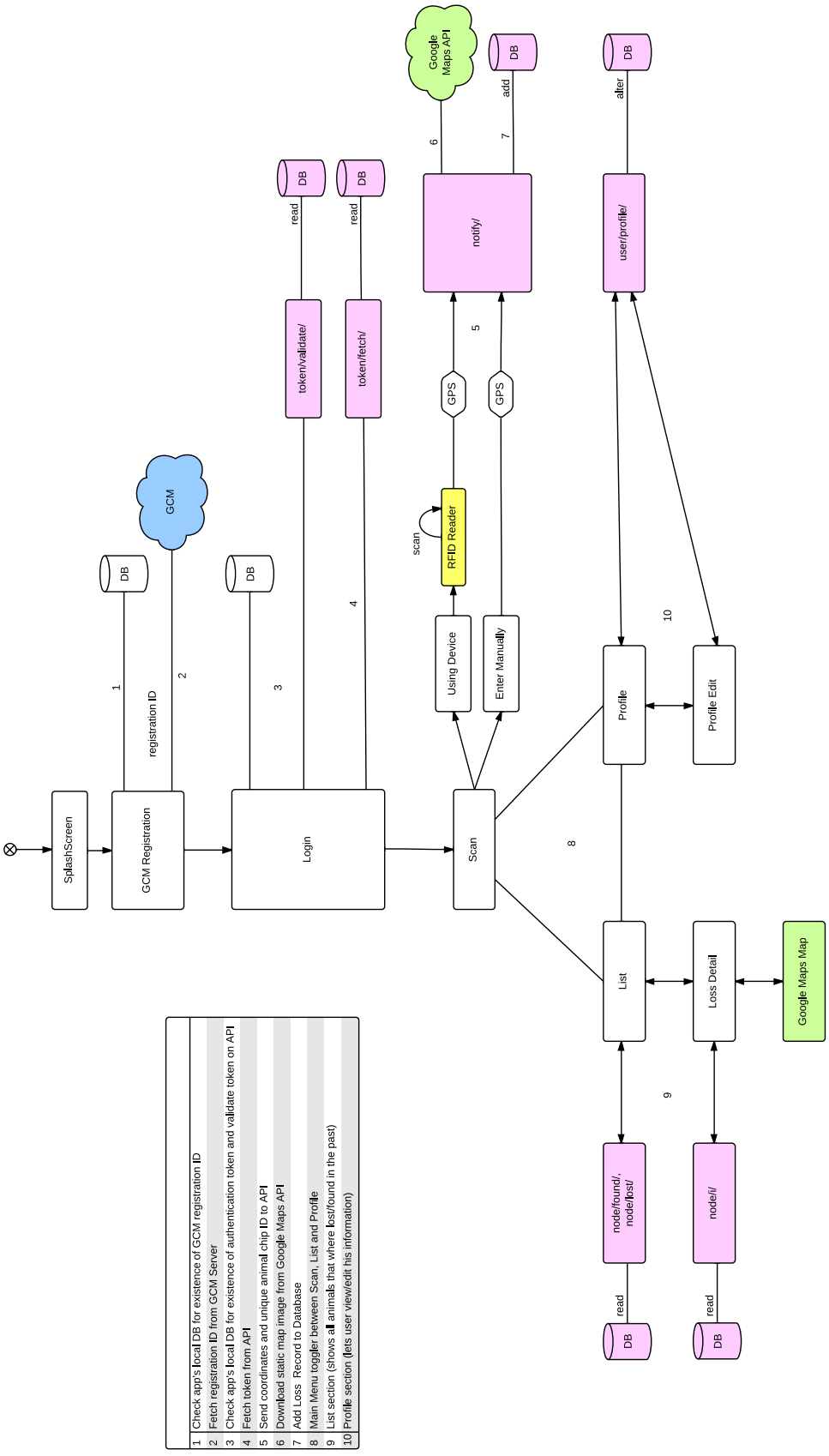


Figure 3.5: Android Application Flowchart

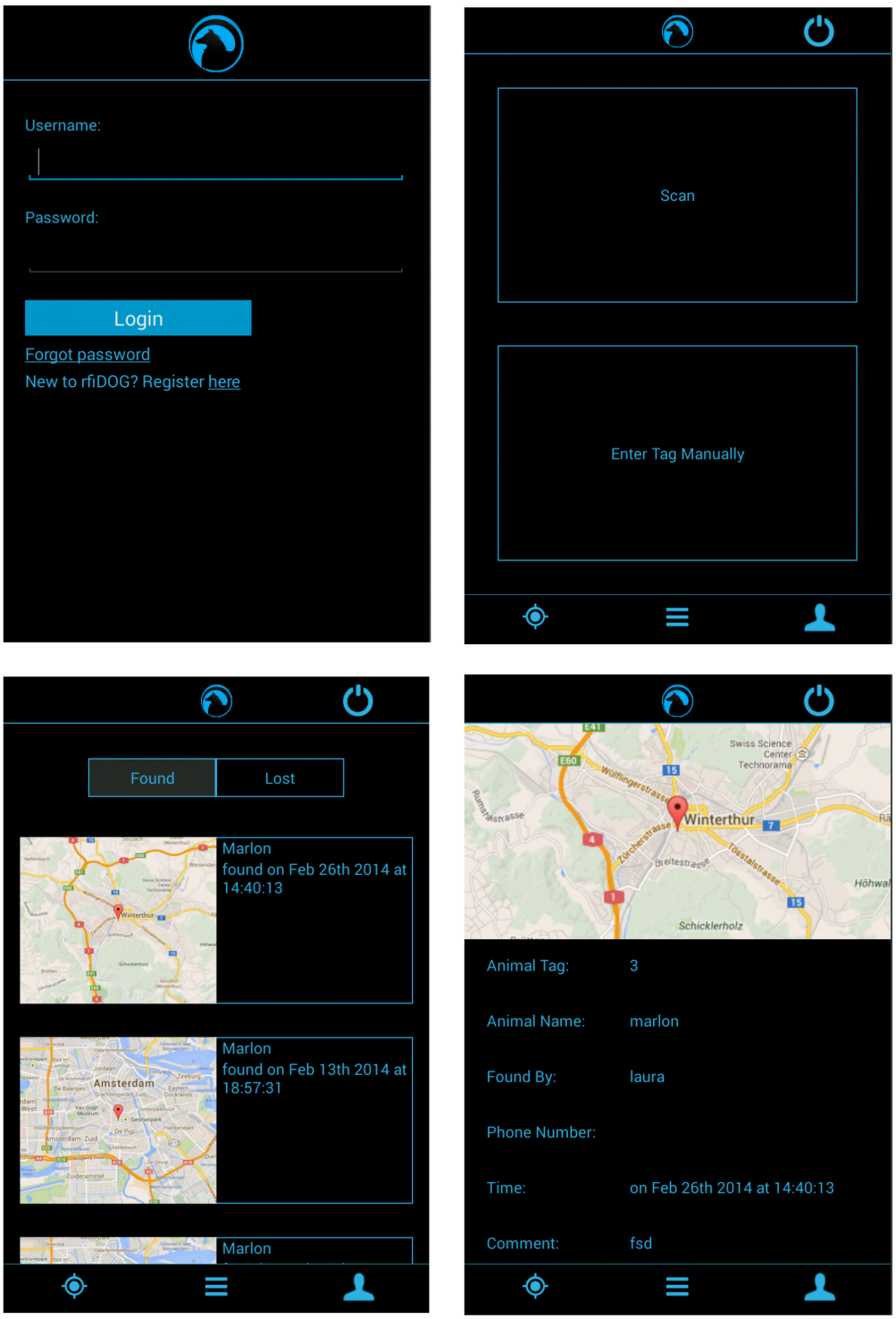


Figure 3.6: Android Application Screenshots

Tapping the logout icon makes the application remove the token from its database. In addition to deleting the token locally, the application sends a GET request to entry point *user/logout/*, which changes the token for the next login session.
The user profile view shows the logged in user's private information and provides a mechanism to alter the data or change the user password. Altering the user data is done by sending a PUT request to *user/profile/* with the changed data in the payload. It goes without saying that users may only alter their own data.
The notification view displays a scrollable list of all lost or found animals. An example list can be seen in Figure 3.6. If the user clicks on any entry of the list he is redirected to a new screen, which shows a map image, downloaded from the API, and detailed information about the lost animal and the finder. Clicking the map image on the top of the screen redirects to a map generated by Google Maps Android API v2 [10], which shows a pin at the exact location where the animal was found.
The scan menu view offers two options. The user can either scan for a chip by using the external device, which we describe in Section 3.3, or he can choose to enter a known tag id manually. Whichever of the two option he chooses at this point, the application displays a dialog box letting the user choose if he wants to post an anonymous notice and requests a short comment. After the user selects the privacy mode, the application calculates its current location using GPS and sends the manually entered or scanned tag, the current position, the comment and the privacy mode setting to the API using a POST request. There are a number of possible responses at this stage. The Android application receives the message *'tag not found'* if the web application could not find an animal tag that matches the sent one. A message stating that the user has scanned his own animal is returned if the sent tag belongs to the logged in user. The Android application receives a success message and is redirected to the new loss record, an example of which can be seen in the bottom right screenshot of Figure 3.6, if the sent tag is indeed found in the web application database and belongs to another user.

HTTP requests made to the API are blocking. They wait for a server response. We design a class called Request which implements a Java AsyncTask that fetches data from a server and takes request method, url and a payload as arguments. The nice thing about AsyncTasks is that they perform an operation in a background thread while the main thread is still running. Our application thus calls upon this class for all communication with the API and eventually gets the response from the API through an interface we define for the Request class.

## 3.3 The RFID Scanning Device

The initial thought behind this thesis is to read out animal microchip implants using Near Field Communication (NFC), a technology featured in most current smartphones. NFC is a relatively new technology whose most common current use cases are contactless payment, wireless device synchronization or the triggered setting of preprogrammed environment modes. NFC derives its core technology from RFID or more specifically from the near-field magnetic induction principle, described in Appendix A, but is additionally extended to support close-range two-way communication. NFC operates at a frequency of 13.56MHz. Unfortunately, while some RFID microchip and reader systems do work on higher frequency bands, we are not able to find a single animal microchip implant that does, which is why we set out build our own device that seamlessly integrates itself into the workflow of our Android application, as if it were part of the device itself. Refer to Appendix B for a detailed account of our hardware selection process. Suffice it to say at this point that the device we design to scan embedded animal tags consists of an Arduino [5] microcontroller connected through a serial port to a RFID reader microchip capable of reading the most commonly available unencrypted animal tag implants within the frequency range 125-134.2KHz, using near-field RFID, also called magnetic induction, as described in Appendix A. The Arduino is connecting to the Android application over a serial Bluetooth connection.
The information flow between the Android application and the RFID scanner device is shown in Figure 3.7. The Android application initializes and opens the Bluetooth serial connection to the Arduino and sends the '$' symbol. The Arduino constantly checks the serial line for input and if it detects '$' in the buffer it immediately sends a command to the RFID reader microchip over its local serial port. The command is a directive that sets the RFID reader into scan mode. If and when the RFID reader happens to pick up a tag within its scanning range, it proceeds to read out the value and sends it back to the Arduino. The Arduino, in turn, sends the received tag value on to the Android application, which subsequently closes the Bluetooth connection.
After sending the initial symbol to the Arduino, the Android application waits for a response from the Arduino. The entire Bluetooth serial communication takes place in a background task. The foreground task shows a spinning loading wheel which is stopped when a response comes back and both threads merge again. We specify a timeout to prevent the Android application from infinitely staying in listening mode. The timeout is set to 120 seconds, because it can be a bit difficult to physically locate animal microchip implants in live animals. If no tag is found in the interval, the application returns to the scan menu view seen in Figure 3.6.

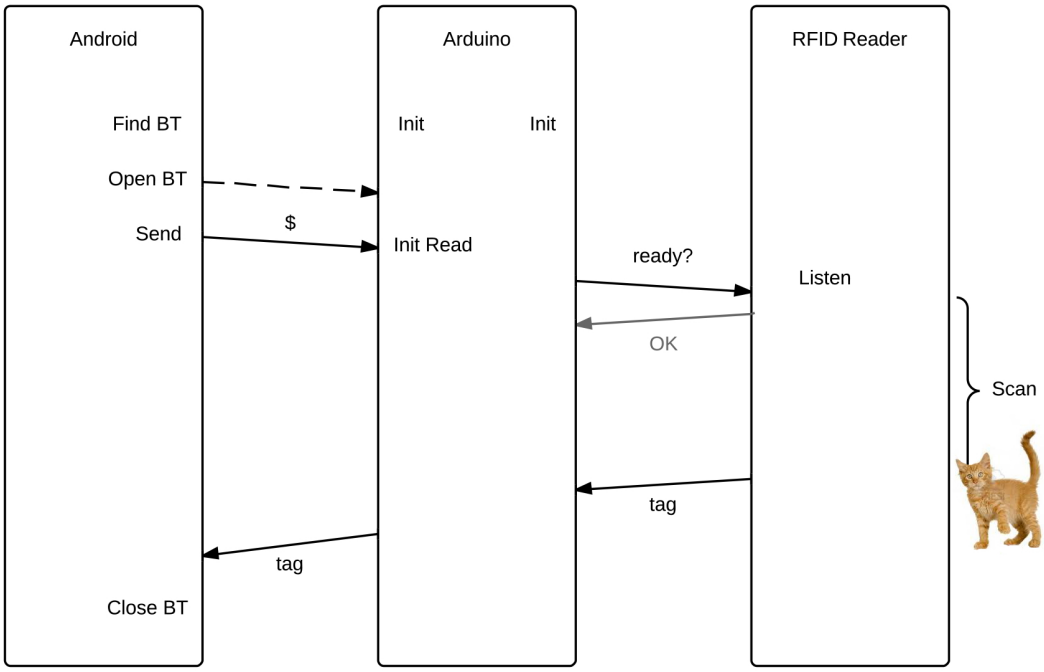


Figure 3.7: Dataflow between Android, Arduino and RFID Reader

## 3.4 Push Notification System

We want to implement a smartphone push notification system which notifies the owner instantly if one of his lost animals is found. For this purpose we make use of a free service called Google Cloud Messaging (GCM). One of the main benefits of outsourcing the push notification system functionality is that we can delegate complex tasks like message queuing, message redelivery on failure, assigning and keeping track of unique device IDs and their current IP addresses to a more robust and specialized system.
Figure 3.8 illustrates an example GCM system. Whenever a push notification needs to be sent to a user, the web application delegates the delivery of the message to the GCM system by sending a push notification request to the GCM cloud server. The request contains one or more unique IDs which point to the devices of the user we wish to notify and some data that needs to be processed on the device or devices after the push notification is delivered.
In our use case, the data consists of the username and the name of the lost animal because we want the push notification to have the message text: *"Hello 'username', 'animalname' has just been found."*. Additionally, we send a link to the detail view, which is shown when the user taps on the push notification. An

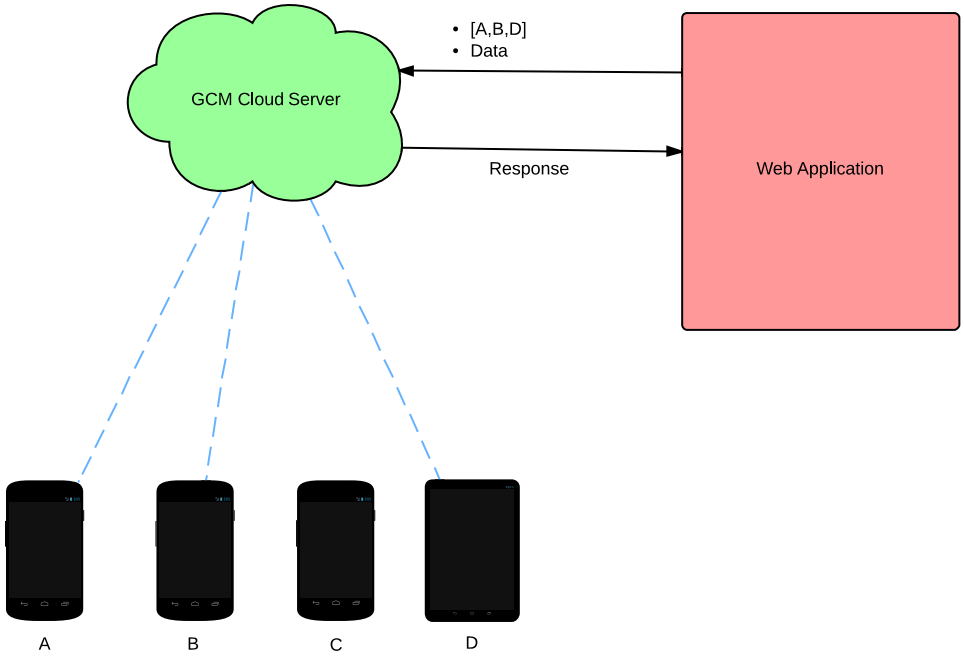


Figure 3.8: Google Cloud Messaging System

example detail view is seen on the bottom right snapshot in Figure 3.6.
The registration IDs we provide to the GCM cloud server are known for every registered user. Our web application stores these registration IDs in Device objects in the database, as discussed in Section 3.1.1. The registration ID is generated, disseminated and potentially sometimes refreshed by the GCM system for every device that uses our Android application. Recall from Figure 3.5 that the device tries to fetch a registration ID first from the local application database and then from the GCM cloud server after the application starts. Once registered, the Android application persistently keeps the registration ID in its local database. If run for the first time, the Android application registers the device with the GCM cloud server. Figure 3.9 shows this process.
In step 1 of the Figure, the device sends a registration request containing the SENDER ID to the GCM cloud server. The GCM cloud server responds in step 2 by providing a registration ID which uniquely identifies the device in the future. The GCM system provides a SENDER ID for every project it hosts. It also provides an API KEY which allows the 3rd party server, in our case our web

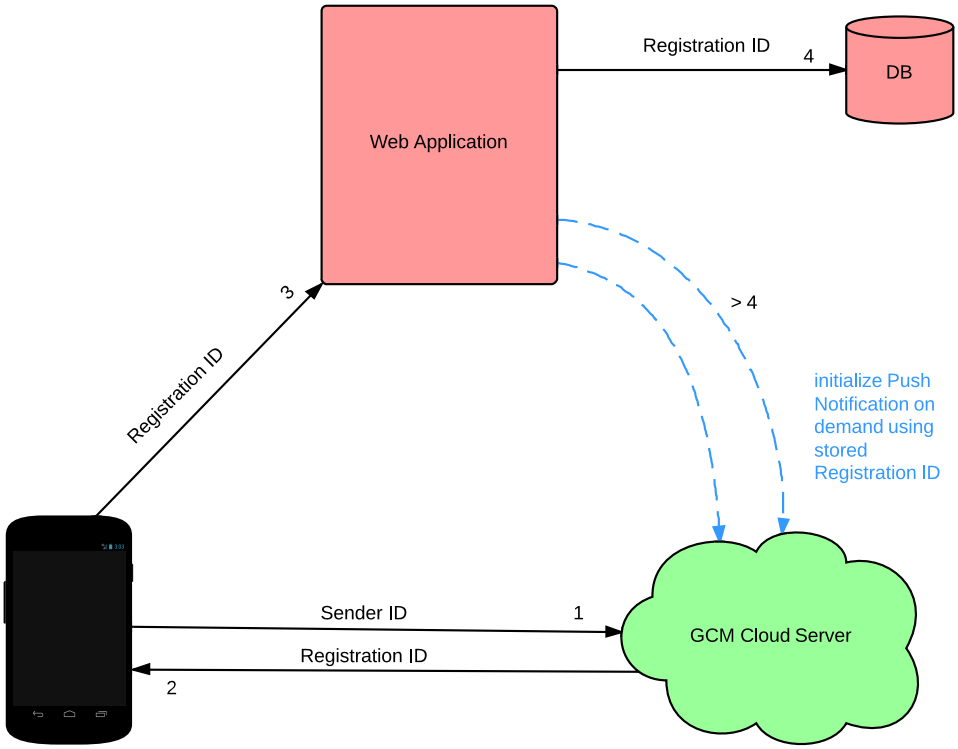


Figure 3.9: GCM Device Registration

application, to authenticate itself. Finally, after obtaining a registration ID from GCM, the android application registers itself in step 3 with the web application. The web application stores the registration ID in a Device object in step 4 and can from then on delegate a push notification to that device.

When sending a push notification request to the GCM cloud server, our web application receives a response as seen in Figure 3.8. If the notification request uses an invalid API KEY, the GCM system returns a status code of 401. If the structure of the sent request data is malformed, GCM returns a status code of 400. These two cases result from some misconfiguration within the web application and are handled by storing the failed request in the database. Further, the administrator is immediately notified by email because manual debugging is required at this point. If an error occurs within the GCM system itself, it returns a status code 5xx. It's likely that such an error does not persist within the GCM system. Our web application tries resending the request after a backoff time.

With a valid API key and well formatted data, the GCM cloud server returns status code 200. This, however, does not guarantee the proper delivery of the push notification to all devices yet. The response data contains delivery information for every provided registration ID. The web application removes registration IDs from the database if they are seen as invalid by GCM, which means that they either never came from GCM or have expired.

Chapter 4

# Results

In order to determine the quality of our system, we take a look at the web application in Section 4.1, then, we analyze the Android application in Section 4.2 and conclude our judgment by discussing the RFID Reader Device in Section 4.3 and the Push Notification System in Section 4.4. Since the result of this thesis is an implemented prototype system, we refrain from posting screenshots of our system in use in this Chapter and instead suggest the reader try it out first hand to get a visual impression.

## 4.1 The Web Application

The web application works nicely. So far we have not come across any show-stopping bugs. Using Django as the implementation framework proves to be a good choice. After a two weeks of climbing the learning curve, the database structure is defined, the API backend and frontend website are delivering the data and we feel confident that we can solve any new problems that come our way with little code and almost no code reuse. The deployment to a live server consists of 10 shell commands, uploading one folder and is completed within less than 10 minutes.
We design the web application with security in mind on every step. We make sure that our web application code is outside of the web server root to obstruct unauthorized access. Django protects its web application database against SQL injection by automatically escaping the user inputs given to the database query functions. We do not stray from the built in query functions which means we are not opening up our database to this vulnerability. Django protects against cross site request forgery by checking a token value in each POST request to the web application. This makes it impossible for an attacker to replay or outsource a POST request to our web application.
Currently, our web application is only accessible using HTTP. This has the drawback that user session information is transported in the clear between server and client. Any man in the middle is able to see authentication data, for example tokens and passwords, being passed around. This can lead to leaked credentials

and hijacked sessions, which is bad.
The database is saved in one file on disk. At this time, there is no data backup scheme in place that protects against the loss of data. If a database entry is purposefully deleted on request of a user, there is currently no recovery method in place if the user changes his mind in retrospect.

## 4.2 The Android Application

The Android Application is designed to enable four core functionalities. It is capable of locating the device using the smartphone's GPS system, it is designed to interact with our web application using GET, POST and PUT requests, it initiates a two-sided Bluetooth connection with the RFID Scanning Device and it is able to receive and display push notifications. All these functionalities have been implemented and work accordingly.
The Android application displays alert messages to inform the user of connection or web application errors or successful operations. The application main menu, which sits at the bottom of the screen, enables quick navigation between the different zones of the application and increases the application's usability.
The application does not store any content from the web application except for the authentication token for the login period, which causes it to make a request to the web application on every view that contains details about a user or animal. This results in the Android application always displaying the newest possible data, but makes the application inoperable in offline mode.
If the Android application fetches the list of lost or found animals it receives all the records in the web application database in one single request. An animal that is found a great number of times causes this list to grow and with it the requested data, which could cause memory leaks in the worst case scenario. A user can exploit this issue by somehow finding or guessing an animal tag number that belongs to another user and generating a flood of lost animal notices, thereby clogging up the owner's newsfeed and creating countless push notifications on the owner's smartphone. There is currently no safeguard in place that hinders one user from generating a flood of lost animal notices in the web application.

## 4.3 The RFID Scanner Device

The RFID Scanner Device has successfully scanned a number of purchased example tags that comply with ISO standards 11784 and 11785. It has additionally scanned six dogs in New Jersey, implanted with unencrypted microchips produced by HomeAgain [12] that operate on a frequency of 125KHz. In addition, it successfully scanned three dogs and one cat in Switzerland all chipped with ISO compliant microchips provided by ANIS [14]. The scanning range, as spec-

ified by the RFID reader manufacturer Priority1Design [20], is ∼4cm. In our tests we saw a successful read within ∼1.5cm if the chip was embedded in the animal and ∼2cm for our tester chips that we held against the RFID reader's coil antenna.
Priced at 100USD it is cheaper than all other preexisting commercially available RFID readers we have found. The fact, however, that we need a RFID Scanner Device at all to read animal chips is bad news. It would be so much more elegant and usable if every person with a NFC enabled smartphone could just walk up to a lost animal and scan it.

## 4.4 Push Notification System

We have tested our GCM implementation many times by sending example push notifications to our Android application on a Samsung Galaxy S3 GT-i9300 running Android Jelly Bean 4.1.2. We measure timespans between 3 and 7 seconds between the moment we ask the GCM system to send the push notification and the moment our smartphone buzzes with the incoming notification.
GCM has some limitations we cannot influence. The limitation that defines the maximum data size of one message does not affect our system because our push notification does not grow in size and is well within the set limit of 4KBs. The limitation that states that a maximum of 1000 devices can be notified in one message does not really impact our system either because we find it unlikely that a single user owns and uses more than a 1000 devices. That user would just be confused about which device to use and would probably not have any space left in his apartment for any pets because of all the devices that lie around. The amount of stored and undelivered push notifications is set to a maximum of 100 notifications. They are generally delivered all at once when the smartphone regains its Internet connection. If the limit of 100 notifications is reached before the smartphone connects to the GCM system, however, the notifications are deleted and replaced by a special push notification that informs the smartphone about the deletion of the messages. Currently, we have not implemented a scheme to handle this situation.

Chapter 5

# Future Work

The results in Chapter 4 state that we have adequately achieved our goal of developing a system that scans and registers users and microchipped animals and facilitates the retrieval of lost pets. However, there are some parts within our proposed design that may be further explored or improved in order to achieve even better results. The following Sections describe some ideas or guidelines for possible improvements and extensions.

## 5.1 Improvements

To enhance the web application, we suggest a number of improvements. A scheme to make regular, robust and distributed backups of the web application database should be implemented. Additionally, we would like to improve the specific data removal from the database. Currently, when a record is deleted, there is no turning back, which means that the user cannot change his mind after he deleted something, which is not nice. Also, serving the web application API and website using HTTPS reduces the risk of leaking sensitive user data, session data and authentication tokens to unauthorized third parties. Anti-flooding techniques should be implemented to detect and protect against malicious users who repeatedly request a resource or clog up the lost animal newsfeed of unsuspecting users.
The Android application should not download the entire lost animal newsfeed on every request. Instead, only the most recent entries should be shown, with the option to fetch a number of older entries. Additionally, a scheme that detects the deletion of stored and undelivered GCM push notifications needs to be implemented if a device is offline for a longer period. This could be handled by sending a push notification synchronization directive to the web application.

## 5.2 Extensions

The bond between humans and pets can be a very personal, intimate and familial one. For this reason, it is all the more important to optimize the recovery of lost pets. We propose bridge the incompatibility gap between RFID animal microchips and Android NFC by either developing a new breed of NFC compatible 13.56MHz implantable microchips, or to include the means of scanning low frequency RFID into smartphone devices. We assume that the latter solution would be simpler because smartphones are replaced and exchanged much more frequently than pets, and, unlike animals, smartphones don't feel pain when a new hardware feature is added. If one of these schemes were implemented, a simple swipe with the smartphone would suffice to read out the tag number. This solution has extreme elegance and saves the finder from making a trip to the nearest animal shelter or veterinarian's office, where they have access to expensive RFID reader devices.
The web application can additionally be extended to scrape freely available animal databases for owner information, which increases the chances of finding the owner of a lost pet who has not yet registered his animal in our web application database.

Chapter 6

# Summary

During the course of this thesis, we develop a complete system that is able to scan and register microchipped animals and notify animal owners immediately when their lost animals are found.

We implement a web application that stores information about registered users, owned animals, smartphone device IDs and animal loss records in a database. In addition, the web application nicely displays the data on a website, where users can easily register themselves and their pets, and it provides entry points where smartphones can send and receive data.

We design an Android application which accesses the web application's data and is able to notify the web application if a lost animal is scanned. The Android application is capable of calculating its current position using GPS and is able to scan an implanted animal microchip by connecting to an external Bluetooth RFID reader device. Further, every time an animal is found, the owner's Android application receives a push notification from our web application, which we implement using the Google Cloud Messaging system.

We find that our system works well, even though it's not as simple and uncomplicated as we would have initially hoped. Due to the hardware incompatibility between the smartphones and the animal RFID implants, we need to build and integrate a separate RFID reader device priced at ∼100$ to read the RFID implants.

The implanted chips vary in general when it comes to protocol and frequency, especially in countries that have not adopted the ISO standards that define the characteristics of implantable animal chips. The bond between humans and pets can be a very personal, intimate and familial one, which puts additional importance on their successful recovery. The hardware incompatibility between the smartphones and implant microchips is a thorn in the eye. It would be nice if future smartphones were able to read lower frequencies, or if the next breed of animal microchip implants were compatible with the NFC standard. Being able to elegantly swipe the smartphone across the pet without having to rely on an extra device would greatly improve the recovery process.

# Bibliography


[1] 24petwatch. http://www.24petwatch.com/. Accessed: 2014-03-03.

[2] Akc reunite. http://www.akcreunite.org/. Accessed: 2014-03-03.

[3] American national standards institute. http://www.petmicrochiplookup.org/. Accessed: 2014-03-03.

[4] American national standards institute. http://www.ansi.org. Accessed: 2014-03-03.

[5] Arduino. http://www.arduino.cc. Accessed: 2014-03-03.

[6] Avid friendchip. http://www.avidid.com/. Accessed: 2014-03-03.

[7] Bayer resq pet identification system. http://www.bayerdvm.com/show.aspx/productdetail/resq. Accessed: 2014-03-03.

[8] E-zpasses get read all over new york (not just at toll booths). http://www.forbes.com/sites/kashmirhill/2013/09/12/e-zpasses-get-read-all-over-new-york-not-just-at-toll-booths/. Accessed: 2014-03-03.

[9] Ericsson mobility report: Global smartphone subscriptions to reach 5.6 billion by 2019. http://www.ericsson.com/news/1741771. Accessed: 2014-03-03.

[10] Google maps android api v2. https://developers.google.com/maps/documentation/android/. Accessed: 2014-03-03.

[11] Google maps javascript api v3. https://developers.google.com/maps/documentation/javascript/. Accessed: 2014-03-03.

[12] Homeagain. http://public.homeagain.com/. Accessed: 2014-03-03.

[13] Iso wg3 summary on the evolution of microchip technology for companion animals. http://www.wsava.org/sites/default/files/ISO%20a%20Brief%20Historical%20Overview.pdf. Accessed: 2014-03-03.

[14] Kennzeichnung und registrierung in der schweiz. http://www.anis.ch/de/microchip/kennzeichnungregistrierung-in-der-schweiz/. Accessed: 2014-03-03.

[15] Lost'n found. http://www.smartphonepettag.com/id/found-a-pet.php. Accessed: 2014-03-03.

[16] Meet django. http://www.djangoproject.org/. Accessed: 2014-03-03.

[17] Microchipping of animals. https://www.avma.org/KB/Resources/Backgrounders/Pages/Microchipping-of-Animals-Backgrounder.aspx. Accessed: 2014-03-03.

[18] Pethub. https://www.pethub.com/. Accessed: 2014-03-03.

[19] Pet's death rekindles electronic id debate. https://www.avma.org/News/JAVMANews/Pages/040701a.aspx. Accessed: 2014-03-03.

[20] Priority1design. http://www.priority1design.com.au/. Accessed: 2014-03-03.

[21] Rfidler - a software defined rfid reader/writer/emulator. https://www.kickstarter.com/projects/1708444109/rfidler-a-software-defined-rfid-reader-writer-emul?ref=search. Accessed: 2014-03-03.

[22] rfidog. http://www.geekcon.org/geekcon-2013/rfidog/. Accessed: 2014-03-03.

[23] Static maps api v2. https://developers.google.com/maps/documentation/staticmaps/. Accessed: 2014-03-03.

[24] Worldwide smartphone population tops 1 billion in q3 2012. http://blogs.strategyanalytics.com/WDS/post/2012/10/17/Worldwide-Smartphone-Population-Tops-1-Billion-in-Q3-2012.aspx. Accessed: 2014-03-03.

[25] Aykut Atali, Hau Lee, and Özalp Özer. If the inventory manager knew: Value of visibility and rfid under imperfect inventory information. *Available at SSRN 1351606*, 2009.

[26] Vipul Chawla and Dong Sam Ha. An overview of passive rfid. *Communications Magazine, IEEE*, 45(9):11–17, 2007.

[27] David Gibson, Nicholas Roddy, Louis Schick, and Glenn Shaffer. System and method for managing a fleet of remote assets, 2002.

[28] Dara J Glasser, Kenneth W Goodman, and Norman G Einspruch. Chips, tags and scanners: Ethical challenges for radio frequency identification. *Ethics and Information Technology*, 9(2):101–109, 2007.

[29] John Halamka, Ari Juels, Adam Stubblefield, and Jonathan Westhues. The security implications of verichip cloning. *Journal of the American Medical Informatics Association*, 13(6):601–607, 2006.

[30] MultiMedia LLC. MS Windows NT kernel description, 1999.

[31] Linda K Lord, Walter Ingwersen, Janet L Gray, and David J Wintz. Characterization of animals with microchips entering animal shelters. *Journal of the American Veterinary Medical Association*, 235(2):160–167, 2009.

[32] Harald J Meyer, Nantarika Chansue, and Fabio Monticelli. Implantation of radio frequency identification device (rfid) microchip in disaster victim identification (dvi). *Forensic science international*, 157(2):168–171, 2006.

[33] Christine Perakslis and Robert Wolk. Social acceptance of rfid as a biometric security method. In *Technology and Society, 2005. Weapons and Wires: Prevention and Safety in a Time of Fear. ISTAS 2005. Proceedings. 2005 International Symposium on*, pages 79–87. IEEE, 2005.

Appendix A

# RFID Technology

Radio frequency identification, or RFID for short, is a technology developed to remotely identify or track objects, animals and in some cases even humans. It has been effectively used to improve inventory tracking in retail stores [25]. Large IT firms, data centers or banks place RFID tags into objects of interest in order to automate inventory checks of their vast array of IT assets [27]. The Northeastern United States highway toll collection system EZPass uses RFID technology to detect and register passing cars [8]. Domestic animals, wild animals and livestock alike are being implanted with RFID tags. In some use cases tagging can help analyze the spread of disease. Sometimes tagging is done to identify and distinguish between animals or even measure their health status. The use of RFID technology in humans raises a number of ethical concerns, especially when implanted subdermally. Some voiced concerns are the potential breach of privacy by unpermitted surveillance and information gathering, the possibility of malfunction, which could cause harm, or even the risk of painful and violent theft [28]. It is nonetheless occasionally implemented in health care, for example, to store patient information [29], to distinguish between disaster victims [32], or, for example, in biometric security authentication schemes [33].
RFID devices can be either passive or active. Active devices require a power source and can communicate periodically. Passive devices have no internal power source and generally consist of an antenna, a microchip and a protective casing. Unless they are destroyed or tampered with, they have an indefinite life span and are generally small, which makes them good candidates for implanting or attaching to objects. Because they lack a power source, passive RFID tags can only communicate or reply by drawing power from the signal emitted from a RFID reader device. There are two different predominant methods of transferring power from the reader to the tag: magnetic induction and electromagnetic wave capture. Figures A.1 and A.2, taken from Chawla et al. [26], portray these two schemes.
The magnetic induction scheme, seen in Figure A.1, is also referred to as near-field RFID. The reader, seen on the left, passes an alternating current through an antenna coil. This current generates an alternating magnetic field which affects the circuit of any passive RFID tag within reach. The external magnetic field

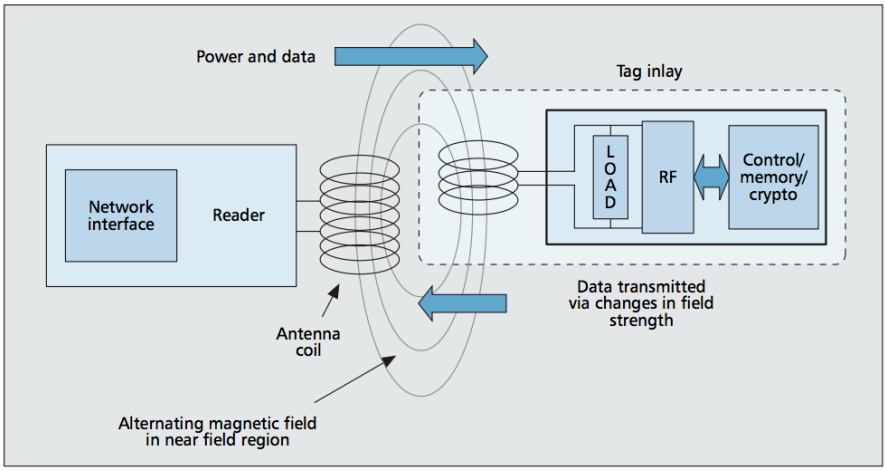


Figure A.1: Near-Field RFID

induces an alternating voltage across the tag's own coil antenna, which, in turn, powers the tag circuit and the attached microprocessor. The tag microprocessor then communicates with the RFID reader using a scheme called load modulation. In load modulation the RFID tag fluctuates the current across its own antenna coil in a controlled manner, for example by periodically rerouting current through a load, as seen on the right hand side of Figure A.1. This fluctuation in current causes fluctuations in the overall magnetic field between the two devices and is detected by the RFID reader. The RFID tag can thus fluctuate the field according to a desired encoded message and communicate with the reader.
Near-field RFID has physical limitations which dictate the maximal distance between reader and tag during successful operation. With increasing distance, the magnetic field drops at a factor of

$$\frac{1}{r^3}$$

with $r$ being the distance between the reader and the tag. If the reader and the tag move further apart, the absolute value of the magnetic field grows weaker, which causes the tag circuit to harness less energy, which in turn leads to a weaker field fluctuation while communicating, which makes it harder for the reader to get the message. Additionally, the range for which near-field RFID can be used approximates to

$$\frac{c}{2 \cdot \pi \cdot f}$$

where $f$ denotes the alternating magnetic field's frequency. As this frequency increases, the operational distance between tag and reader decreases proportionally. The frequency of operation, which dictates the maximal distance requirement, is determined by analyzing the number of bits in a message and the potential need to properly handle multiple simultaneous tag read processes.
For use cases where the maximal distance requirement does not match the target operational frequency researchers have developed the scheme we describe in the following paragraph.

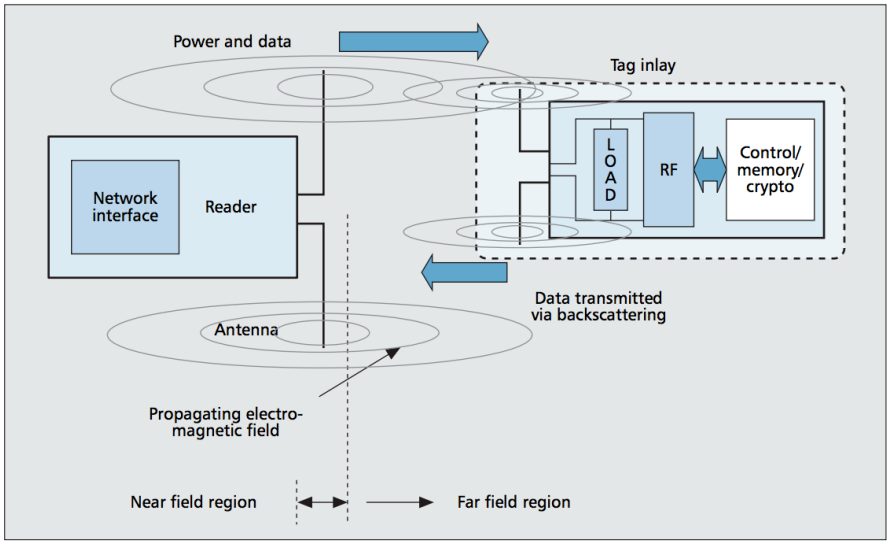


Figure A.2: Far-Field RFID

Electromagnetic wave capture, seen in Figure A.2, is also referred to as far-field RFID. This scheme sees the RFID reader and tag devices equipped with dipole antennas. It is designed to work for greater reader to tag distances and can thus not rely on load modulation, because the reader would no longer detect the fluctuations in magnetic field. Instead, it uses a technique called back scattering. Both antennas are tuned to a precise operating frequency. The reader's antenna is able to absorb most of the energy of the received signal at that frequency. Using a fraction of the incoming energy, the tag circuit can dynamically fluctuate its impedance, which causes signal reflections at differing energy levels. The RFID reader is equipped with a sensitive radio receiver that can detect the incoming signal pattern.
Far-field RFID is limited by the emitted signal energy level at the reader and the sensitivity of the receiver's radio when picking up the reflected signal from the tag's antenna. The returned signal at the reader is a lot weaker than the

emitted one, because the signal has to travel to the tag and then back to the reader, which causes it to be attenuated twice. Modern radio receivers, however, can be sensitive enough to still pick up a reflected signal from a distance of 5 to 20 meters, as stated in Chawla et al. [26].

Appendix B

# RFID Scanner Hardware Selection

We decide to opt for a wireless connection between smartphone and scanner because not all smartphones carry the same plugs and because we want to maximize compatibility of our device. We decide to use a serial Bluetooth connection between the Android application and the scanner device, because most common smartphones carry Bluetooth adapters, because Bluetooth chips are readily available and because they don't cost too much.
We find a number of costly commercial RFID readers, priced around $400 without any Bluetooth capabilities. Using these would require the user to scan and then enter the tag manually, which breaks the application flow. Trying to find a product that is more extensible proves to be a bit difficult. We find a number of products that fulfill our needs except for the fact that they are sold out. On alibaba.com, an online electronics portal, we find a selection of questionable RFID devices priced at around 200$. We say questionable because the descriptions and specifications are not clear about which frequencies the devices are capable of communicating on. Chatting, or rather exchanging mixed messages, with various operators does not quell our doubts about the quality and legitimacy of the products.
Short of designing a device from scratch, which would surpass our current experience level and thus interfere with the thesis timeline in an unforeseen and adverse way, we finally decide to use an Arduino [5] microchip as the backbone of our scanning device. Arduino is a good choice because it's readily available, it's not too costly, has a big community of users and is very well documented. Arduino microcontrollers can easily be equipped with a Bluetooth shield. Now the only thing left to decide is which RFID reader chip to use.
Recall from Chapter 1 that there are a number of operating frequencies currently in use for animal microchips. It would be preferential to get a reader that covers the entire used frequency band which can read out all the possible different tags with the exception of encrypted tags. We get lucky and find a small web store in Australia called Priority1Design [20], which specializes in the design of animal microchip RFID readers. They state on their website that their devices can

read frequencies between 125 and 134.2KHz. The hardware costs of the entire developed system amount to a total of 100$.