Empowering Citizen Science: A Generic Data Collection Framework

Sebastián Duque Jaramillo

Dissertation 2015

Erasmus Mundus MSc in Dependable Software Systems

Department of Computer Science

Maynooth University, Maynooth
Co. Kildare Ireland

A dissertation submitted in partial fulfilment
of the requirements for the

Erasmus Mundus MSc Dependable Software Systems

Head of Department : Dr Adam Winstanley
Supervisor : Dr Peter Mooney

June 2015

Declaration

I hereby certify that this dissertation, which I now submit for assessment of the program of study leading to the award of Master of Science in Dependable Software Systems, is entirely my own work and has not been taken from the work of other save and to the extent that such work has been cited and acknowledged within the text of my work. This dissertation contains fewer than 22,000 words.

Sebastián Duque Jaramillo
June 2015

Acknowledgements

I would like to thank my supervisor Dr. Peter Mooney for his guidance, patience, feedback and support during the project. His friendliness and interest in the field and the project were always a source of extra motivation for me. I would also like to thank Dr. Rosemary Monahan for her diligence and dedication to the Erasmus Mundus MSc in Dependable Software Systems programme. Lastly, a sincere thank you to those who volunteered in our case study and gave their valuable feedback, namely and in no particular order: Suhyun, Phattara, Lu, Mário, José, Maikel and Ester.

Abstract

Citizen Science (CS) is collaboration between scientists and citizens to expand opportunities for scientific data collection and problem solving. Recent advancements such as the Internet, social networks and smart devices have created a technological platform for CS to engage more citizens to work on a wide range of scientific problems.

Due to technical, financial and management resource constraints many organisations struggle to develop effective tools to collect scientific data in CS projects. A robust web and mobile interface for scientific data collection will ensure collection of higher quality scientific data. While web and mobile applications have been developed for some CS projects many CS projects are hindered by the complexity and intrinsic costs of implementing these applications.

This thesis describes a web-based model for CS data collection suitable for both small CS communities and larger scientific organisations. Offering features commonly used in CS projects, this model reduces costs associated with software implementation and management in CS. A CS campaign is undertaken as a case study that validates our model in a real world scenario. Overall the generic data collection framework presented will empower communities and organisations to engage and use CS in more ways and on large scales.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Silvertown (2009) defined a Citizen Scientist as a volunteer who collects and/or processes data as part of a scientific inquiry. He also highlights how the origins of Citizen Science (CS) go way back in time with some birdwatching communities like the National Audubon Society (1905) in the US and the British Trust for Ornithology (1932) in the UK running these types of projects more than 100 years ago. Over the last few years, advances in technology such as widespread availability of Internet connections, smart devices (e.g. smart phones and tablets), wireless sensor networks and social networks have created a platform for Citizen Science (CS) projects to grow. Newman et al. (2012) advise new CS projects to consider inter-operable, customizable, open-source solutions where possible, encouraging the use of open-data standards and open-source software.

In this thesis we outline our contribution to the CS community by researching, designing and developing an open-source software tool which makes data collection for CS projects easier and more accessible. Our tool has modifiability as one of its main characteristics and it follows a 'simple is best' user interface approach. We believe that this tool will be of great interest to CS projects which do not have access to significant IT and financial resources to develop their own data collection tools and applications. Because it is not tied to a specific data model or schema it offers the potential to empower both large and small cs communities and organisations to collect valuable CS information.

1.1 Motivation

Citizen Science is defined by OpenScientist (2011) as "The systematic collection and analysis of data; development of technology; testing of natural phenomena; and the dissemination of these activities by researchers on a primarily vocational basis". This definition is comprehensive on the activities citizens may be engaged in and also highlights how, in most projects, they do so based on their own motivation – it is not their job. We will use this definition as the basis for CS in this project and thesis. The interest of citizens, experts and non-experts alike, in participating on CS projects has been increasing across the globe (ARC Centre of Excellence for Environmental Decisions (CEED), 2014; Mooney and Corcoran, 2014), while the strongest motives behind volunteering on a campaign have been found to be the

ability to **contribute** to *conservation* or a *scientific study* and previous **interest** in *nature* or the *campaign's field* itself (e.g. birds or insects) (Evans et al., 2005; Sickler et al., 2014).

There are different categories of CS projects in which citizens can get involved in based on their cognitive interests, time available, curiosity or other factors. They are: crowd funding, distributed computing, observation/classification, gamification and data gathering. We expand further on each of these and present examples in Chapter 2. We limit our scope on this project to the data gathering category as it is the category where functional and interface requirements can be abstracted from the specific needs of each project. There are some known concerns regarding the quality of data gathered on projects of this nature:

- Lack of training in research and monitoring protocols might lead to citizens providing poor quality or completely incorrect data.
- Citizens may not be able to grasp the specific problem and collect incorrect or unusable data.
- Groups of people with specific alternative interests might try to pollute the data to generate specific results – with data bias.

However, some argue that, when properly managed, the cost-effectiveness of CS data can outweigh data quality issues (Gardiner et al., 2012) and that citizens' rising ability to collect scientific data coupled with increased interest and participation offers very significant potential for governments to consider using information derived from CS or in a crowdsourced manner (Haklay et al., 2014; Lauriault and Mooney, 2014). Usually this risk is mitigated (or suppressed) with proper volunteer training, a well defined process for data collection, a clean and non-confusing user interface and data quality verification activities performed on the collected data by domain experts. A survey of 340 CS projects by Kim et al. (2013) found that only 39 (11%) of them provided a mobile alternative to facilitate data collection, while 180 projects (53%) used websites as a primary point of data submission. This is a major obstacle in CS at the moment. We believe the main causes for the low adoption of these technologies are the costs and technical requirements associated with providing them. The work outlined in this thesis contributes to removing these obstacles.

1.2 Overall project objective

In this thesis we shall provide a solution suitable for both small communities and larger organisations wanting to conduct a CS campaign. The former would benefit of having a readily available open source, easy to adapt solution while the latter can benefit by saving on the initial IT overhead and may require their staff to only perform the initial deployment and configuring, eliminating the need and cost of significant development activities. Additionally, our solution will enable citizen scientists to contribute from different user devices (e.g. computer, smart phone, tablet), allowing campaigns to benefit from the advantages they pose and lowering the barrier derived from the technical requirements that come with them.

Serrano Sanz et al. (2014) established an important goal to develop and share new tools for different models of participation in CS. Lauriault and Mooney (2014) identified the need

of unambiguous and robust experimental designs for CS and crowdsourcing of data and information collection that can be used by both communities and governmental organisations. Our project directly addresses this software requirement from the CS community. We see our solution as being a *bridge* as no existing software currently gives ownership to the organisation while providing a solution where server and client data models are generated from a generic definition which adapts to the needs of the CS project in hand.

1.3 Research question

The high level research question of our project is **RQ1**, *How can novel technologies empower the CS community?* The scope of this question, and our research, is narrowed and further subdivided into separate research objectives, **RQ1.1** and **RQ1.2**

- **RQ1**: How can novel technologies empower the CS community?
  - **RQ1.1**: How can multiple organisations use the same data collection model as a base for different projects/campaigns?
  - **RQ1.2**: Can a data collection model be used and maintained across platforms and user devices?

These questions are the main drivers of our project. Understanding how the CS community has adopted to use new technologies to achieve bigger goals and scale up the size of the projects is the first step in answering them. With a firm understanding of the state of the art in the use of technology in CS projects, we set as our research objective to provide a generic interface for diverse CS projects, that can be configured to accommodate a variety of data model/user interface widget combinations, based on the needs of each project. The resulting developed framework will provide a significant contribution to the CS community by bridging the gap with an open source implementation that can be adopted, widely used, improved and extended by the community members themselves.

1.4 Solution

To directly address the research questions above, we have designed and developed an open source data collection framework which simplifies the process of implementing a cross-platform interface for CS projects. Further chapters in this thesis detail our framework but a brief introduction to it follows. From a definition of the desired data-model that will be used by the project a mobile-first web application supports the data-collection stage with input validation, added metadata to the contributions and access to the GPS and camera sensors. This application is intended for the use of citizen scientists. The team in charge of the CS project can count on a *Form Builder* application which is provided to facilitate the definition of the data-collection form. They can also easily monitor and extract the data to gauge the status of the campaign or process the data in their own software.

A case study validated the suitability of our framework. In this case study we simulated a CS campaign which adapted our framework to easily create with ease the data collection application used by the citizen scientists. The overall feedback from the volunteers was very positive and lessons learned were converted into recommendations for future CS campaigns.

1.5 Structure of this thesis

The remainder of this thesis is structured in five chapters as follows: In Chapter 2 we present the findings of our literature review in the CS domain and highlight the most relevant contributions in the field up to date. We name and briefly detail the two platforms that are most closely related to our project, to which we compare in a following chapter. In Chapter 3 we begin by defining the key requirements that drive our contribution and describe the design and implementation phases, as well as the reasoning behind important design decisions and the dependability attributes that were considered. After presenting our solution we move onto Chapter 4, where we assess the value of our work in two main areas: How key requirements are met and a comparison with existing platforms, as highlighted in Chapter 2. Following, in Chapter 5 we present a case study, in which we took on a CS campaign to collect the "Signs of Spring" that volunteers could find on town. Chapter 6 closes the thesis and outlines our conclusions, where we review the value of our work and discuss opportunities that we have identified for future work in both the short and long terms.

Chapter 2

Background and Related Work

There are many similar, yet slightly different, definitions of CS in the literature. For some, it is the formalised public participation in scientific inquiry and research, usually through gathering of information (Bonney, 1996; Bonney et al., 2009). For Hecht and Rice (2014), CS goes beyond asking citizens to collect and analyze data and into knowledge-acquisition tasks and understanding of scientific objectives and processes. Tweddle et al. (2012) define CS "as volunteer collection of biodiversity and environmental information which contributes to expanding our knowledge of the natural environment, including biological monitoring and the collection or interpretation of environmental observations". What they all have in common is the inclusion of a group of volunteers to scale and achieve set scientific goals. The ways in which volunteers can contribute to expand our knowledge can be very different resulting in a categorization of models for CS projects which we expand below.

2.1 Passive involvement

Crowd funding is a practice in which an individual proposes a project or idea and then asks a large number of people to back it up with (small) monetary contributions, usually in exchange for a reward in proportion to the amount contributed. This practice is not exclusive to CS, as many other fields (such as art, fashion, games, music and technology) have used it to collect the funds required to make their projects viable. Some of the most widely used platforms for general crowd funding are Kickstarter[1], Indiegogo[2] and GoFundMe[3]. These platforms usually charge a fee of five to ten percent when the idea gets funded. Citizens can back science projects under this model showing their support with money instead of time. They can browse through the projects available for funding, filtering by category, and choose

[1] https://www.kickstarter.com
[2] https://www.indiegogo.com
[3] https://www.gofundme.com

the one they are interested in backing. There are also other crowd funding platforms that specialize in scientific research projects, such as experiment[4], consano[5] and petridish[6].

Another option for passive involvement in CS can be found on Distributed Computing. **Distributed Computing** makes use of network-connected computers to solve (usually large) computational problems by assigning smaller subproblems to each individual member of the network, thus leveraging on their cumulative computing power. These problems can be of any nature or discipline such as scientific, problems in which we focus. Citizens interested in getting involved under this model are usually required to download a piece of software that handles the shared use of their hardware resources whenever they are available. This may aid projects in reducing the costs associated with computing, such as eliminating the need of booking timeframes in a supercomputer. Werthimer et al. (2001) were pioneers of this type of projects, they use volunteer's computers to seek for signals of extraterrestrial intelligence by processing radio signals recorded from space. Abbott et al. (2009) have been searching for signals of spinning neutron stars and their partial results consist of more than 36 new such stars discovered. Others seek to find a cure to well known diseases such as Malaria (Chubb et al., 2010) or Alzheimer's, Diabetes, Parkinson's and others (Larson et al., 2002). Lastly, Stainforth et al. (2002) use distributed computing to create climate models with predictions of temperature, rainfall and the probability of extreme weather events to better understand and answer questions about world climate change.

2.2 Active involvement

Observation/Classification projects are distinguished by the type of activities that citizens are requested to perform, which are usually classifying data samples according to a given criteria. The main requirement behind this type of projects is that algorithms to identify a pattern or characteristic in a given image or sound are still not as effective as desired. Lintott et al. (2011) pioneered this field with Galaxy Zoo[7] where they had a dataset with a million images of galaxies; volunteers were given sample images and asked to classify the shape of the galaxy they were seeing. Over 50 million classifications were received by the project during its first year, which encouraged some of the team behind the project to run a more complex campaign that ended up receiving more than 60 million classifications in 14 months (Willett et al., 2013).

Zooniverse[8] is a platform that sprung from Galaxy Zoo and now groups CS projects related to Space, Climate, Humanities, Nature, Biology and Physics. Shamir et al. (2014) asked volunteers to listen to sounds recorded from killer whales and categorize them according to their sounds. They found that the results obtained by the volunteers was less accurate than the output of a computer analysis. Hennon et al. (2014) prompted citizen scientists to answer 'simple questions' that would classify tropical cyclone data, their findings suggest that, under

[4]https://experiment.com
[5]https://www.consano.org/home/index
[6]http://www.petridish.org
[7]http://www.galaxyzoo.org
[8]https://www.zooniverse.org

certain conditions and given the same data, the results obtained from the volunteers are better that those of an automated technique. Both projects are part of Zooniverse, and yet they observed different results in regards to the effectiveness volunteer involvement.

Gamification is defined by Werbach and Hunter (2012) as "the use of game elements and game design techniques in non-game contexts". In the context of CS, this usually results in user engaging experiences where contributions are being made while having fun on a game or being rewarded in a points or badges system.

Foldit[9] (Cooper et al., 2010) is one of the most cited projects in CS that makes use of gamification. It presents users with a game where the underlying goal is to better understand protein structure folding and design. Fraxinus[10] is introduced in MacLean (2013). It is a game built on top of the Facebook social network that seeks to understand a disease produced by a fungus that has killed millions of ash trees across Europe. The author mentions how the Facebook platform can be used to encourage competition between users, which may lead to an increase in the quantity and quality of the interactions with the game. EteRNA[11] is a project run by scientists and researchers from Stanford and the Carnegie Mellon universities. It presents players with puzzles on RNA sequence folding and aims to reveal new principles that may expand the possibilities and uses of RNA for potentially controlling living cells and viruses.

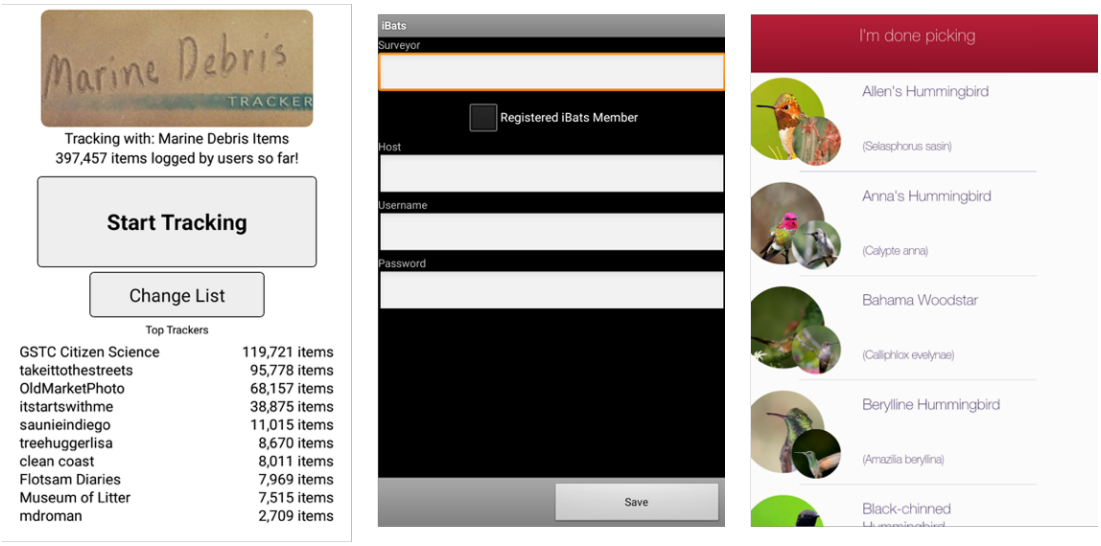
Data gathering projects are characterised by the activities performed by the citizen scientists, whom are requested to collect raw data which will later be processed and analysed, usually by qualified researchers that are also taking part on the campaign. The interface between the citizen and the scientific organisation tends to vary on a per project basis and it becomes one of the most important aspects of the project, as its success is both a key engagement factor and a driver of higher quality data.

The widespread use of smart phones and the possibility to run mobile applications on them have increased the viability of data gathering projects, as citizens can, in theory, register their contributions from anywhere and in addition provide unconventional data such as pictures taken with the camera on the device or an accurate location derived from the GPS sensor.

Some projects have developed their own data collection applications and offer them to volunteers as their main tool. The Marine Debris Tracker[12] (Figure 2.1a) project offers mobile applications for Android and iOS with which citizen scientists are expected to log sightings of trash on the coastlines or water by selecting the type of debris from a preset list and obtaining the GPS location from the sensor on the device. What's invasive![13] also has Android and iOS applications and it expects volunteers to contribute observations of invasive plants or animals that may threaten native species in participating sites. The Indicator Bats Program[14] (Figure 2.1b) monitors biodiversity of bat species in Europe and seeks to enable monitoring of change around the globe; participants used to require a sound recording device,

[9] http://fold.it/portal
[10] https://apps.facebook.com/fraxinusgame
[11] http://eternagame.org/web
[12] http://www.marinedebris.engr.uga.edu
[13] http://www.whatsinvasive.org
[14] http://www.ibats.org.uk

GPS and recording form but these were superseded by mobile applications for Android and iOS. The Hummingbirds@Home[15] (Figure 2.1c) project allows contributors to record data about hummingbirds, sources of nectar and feeding events, whether it is in an area they regularly survey or by single sightings which may happen anytime and anywhere. The project offers different alternatives for data collection such as a web interface and mobile applications for Android and iOS and is also open about the data it has collected, offering a data explorer web interface.



(a) Marine Debris Tracker (b) Indicator Bats Program (c) Hummingbirds @ Home

Fig. 2.1 Screenshots of well known CS mobile applications

There are also generic platforms that seek to support organisations interested in running a campaign by offering the possibility of hosting it for them (Newman et al., 2011) or facilitating the creation of a mobile data collection interface (Kim et al., 2013). They require users to register their campaign and set up the data gathering forms on the platform, which also ends up having an aggregating role, where previous or current campaigns can be found. They are presented and examined with more detail in Sections 2.3.1 and 2.3.2 below, as we believe they are in relation to our project the most closely related work in the field.

2.3 Existing Citizen Science platforms

In the past CS projects wanting to crowdsource data gathering activities with volunteers used to have to build their own interfaces from scratch. This is both expensive and difficult to maintain, as it requires an individual or team with the technical knowledge to create software for the web and/or mobile interfaces. In addition when support for mobile devices is required the complexity, time and cost of maintenance related activities increases as a different software project is often required for each platform.

[15] http://www.hummingbirdsathome.org

The issues outlined above are a stumbling block for the CS field. Finding alternatives to overcome them pushed some members of the CS community to devise platforms that simplify the process of CS project creation, making it both easier and cheaper. Two of the most well-known platforms follow:

2.3.1 CitSci.org: A Comprehensive Citizen Science Support Platform

CitSci.org[16] was developed and introduced by Newman et al. (2011). It is supported by the National Science Foundation and the Colorado State University in the United States. The authors identified the high cost and complexity around creating systems for each project and created what, initially, was a website designed to support multi-scale CS projects. Organisations may, via a program coordinator, create their own projects and customize them using a web GUI. The program coordinator may manage project members, who must register and login before being able to access the project. This results in a controlled crowd-sourcing platform, where participants and their contributions may not be of anonymous nature.

The platform has evolved over time, adding new features and increasing in use and adoption. It currently lists 164 projects spread across the globe (US, Mexico, Costa Rica, Jamaica, Brazil, Australia, Mongolia, India, South Africa, Cameroon, Sweden and Germany), and includes tools for building custom reports and data analysis. It also offers Android and iOS applications with an added photo upload functionality, although other features of the platform are not functional, such as custom defined select lists and a restriction to only support simple point observations. Even though there is no set date, they (CitSci.org Mobile Apps, 2015) expect to cover these short comings in the near future as newer versions of the apps are developed. Figure 2.2 contains three screenshots of the Android application of CitSci showing the flow of the user interaction with the application; on the left the user is required to log in beforehand, on the center the functionality to add observations is provided after selecting a project and data sheet and on the right a data collection form is displayed, with the location inputs being filled with the information provided by the mobile's GPS.

2.3.2 Sensr: A Flexible Framework for Authoring Mobile Data-Collection Tools for Citizen Science

Sensr[17] (Kim et al., 2013) was developed by the research team in HCI (Human Computer Interaction) on the Carnegie Mellon University. It is an authoring environment that enables people without programming skills to build mobile data collection and management tools for CS. The authors show how Sensr allows people without technical skills to create mobile applications. Findings from their case study demonstrate that their system successfully overcomes technical constraints and provides a simple way to create mobile data collection tools. The barrier imposed to small organisations by the required technical expertise and infrastructure requirements that underlie a data collection mobile application is part of the

[16] http://citsci.org
[17] http://www.sensr.org

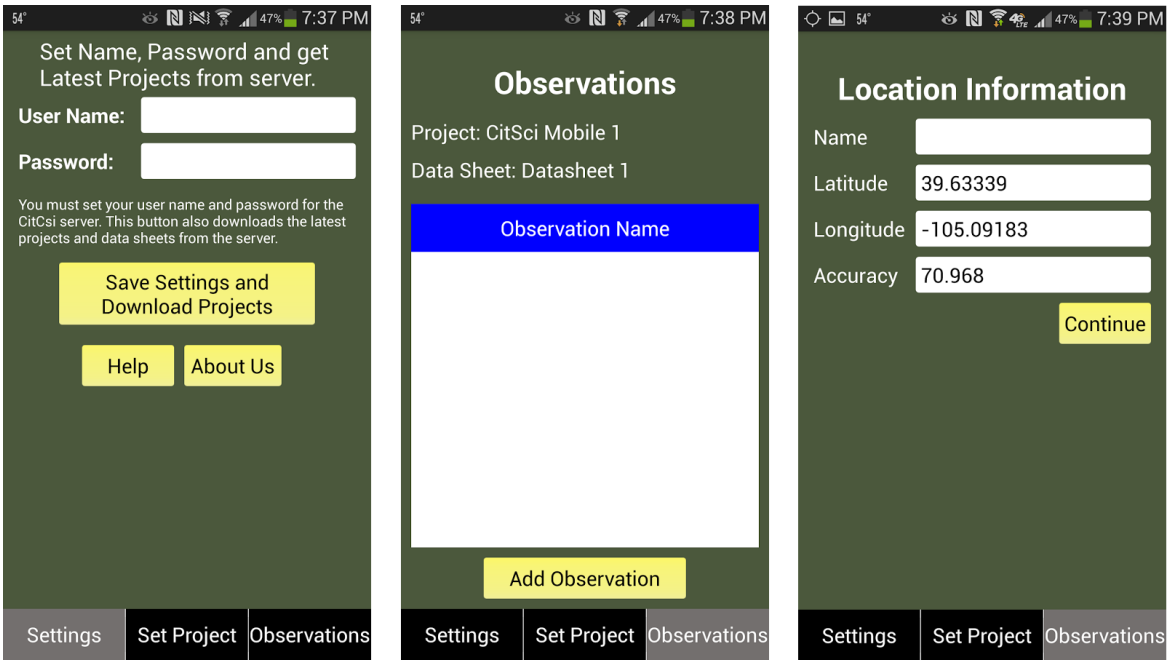


Fig. 2.2 CitSci's Android interface

motivation for the authors. There are, at the time of writing, over 48 projects created on Sensr in 16 different categories (e.g. Birds, Insects and Climate & Weather).

An organisation may use the Sensr website to *author* a campaign, which is then deployed on the –iOS only– Sensr mobile application. Volunteers can then find the campaign on the app and contribute data to it using it. The process has a low technical complexity and the authors approach the design of the web and mobile applications from a HCI and best user experience perspective. As a consequence usability is emphasized and the interface for creating the data collection view is based on dragging-and-dropping existing widgets that may later be customised. Unfortunately, these widgets are very limited in variety, with only three different types being provided: photo upload, radio buttons and text. Figure 2.3 is adapted from Kim et al. (2013) and it shows the interface for designing a campaign specific form in the Sensr application. The authors pose the development of an Android app as future work (Kim et al., 2013). Volunteer contributions are stored in a Sensr database and there are views to explore the data in the web and mobile applications. In addition, the web application offers the possibility to download a file with the collected data to the local computer.

Fig. 2.3 Mobile interface creation from Sensr's website

Chapter 3

Solution

3.1 Requirements specification

The requirements of our solution derive from the research questions, introduced in Section 1.3, and the current state of CS projects as found on the literature review we conducted.

R1 - The ownership of the data collection tool shall be given to the organisation

We believe, based on empirical evidence, that most CS projects have some infrastructure in which project specific applications could be deployed. Existing CS platforms (see Sections 2.3.1 & 2.3.2) simplify the creation of a data collection interface but the ownership is not given to the organisation and it is not possible for them to be independent. This requirement addresses this need.

R2 - The data model of the application shall be definable

CS projects differ in objectives and requirements. It would be the exception to find two different campaigns that have the exact same data needs. The solution we provide shall allow users to define their own data model, with different–but standard–types of input. CS projects can also require data of qualitative nature – text based reports or just images, audio or video (Wehn et al., 2015).

R3 - Mobile devices shall be supported including the use of camera and location sensors

Kim et al. (2013) found only 11% of the CS projects they surveyed offered mobile applications and the advantages that derive from them–e.g. portability, camera and GPS sensors. There is a (technical) gap that needs to be bridged for CS projects to leverage mobile technologies and that is covered by this requirement.

R4 - The data collection application shall allow the defined datamodel to be modified during a CS campaign

CS campaigns may run in tight timeframes that depend on uncontrollable events (such as season change or animal migration). This imposes a hard constraint on the availability of the data collection tools used. Project's specifications may change during the data collection stage. A new data field can be identified to be required or an existing one found to be confusing or no longer needed. A rigid solution that prohibits modification of the datamodel or that results in high complexity would be a drawback to the adoption of said solution. This requirement is set with that in mind.

R5 - The application shall prevent, so far as possible, bad quality data from being collected

The success of a CS project depends greatly on the data that is reported by its volunteers. Ensuring that the data is reliable, objective and accurate is one of the challenges of CS campaigns (Conrad and Hilchey, 2011). The application should address this issue providing the means to prevent *dirty* data from being collected. We use the term *dirty* to refer to data-type inconsistencies, like asking volunteers for a `tree's height` in meters, which is expected to be a number, and receiving contributions with values such as '`asd`'.

It is out of the scope of this project, and frankly challenging, to identify `false` but `clean` data, as it is more a semantic attribute of each field in the domain of the specific project. For example, a volunteer could submit a `false` contribution where an ash tree's `height` is '`50`', a value that is physically impossible for a tree of this kind. The application cannot stop volunteers from putting in false data, whether it is done purposely or by mistake. Recommended best practices for CS projects (Gouveia et al., 2004; Legg and Nagy, 2006) include having a large sample size and a stage where domain experts purge and refine their datasets.

R6 - The data collected by the application shall be easy to extract

Data collection is, as seen by Bonney et al. (2009), the sixth out of nine activities in the model for developing a CS project. Analysis and interpretation have to be done on the collected data before results can be obtained and disseminated. It is important to have the means to export the data that has been gathered in a standard format that is interoperable and recognised by different databases or other data-related programs (e.g. Microsoft Excel) and that multiple programming languages support. Examples of such format could be CSV or XML. This also serves backup purposes. Project experts can then use this data to perform scientific analysis and check for other issues identified by **R5**.

R7 - Open Source shall be a hard implementation constraint of the project

Providing a tool that is based on open source benefits the CS community as the code base can be further advanced by others or it can be forked and tailored to the specific needs of the project or organisation. This also derives from the need of new customisable, open-source

tools that have been previously expressed by others (Newman et al., 2012) Serrano Sanz et al. (2014)).

3.2 Overall project concept

To meet the requirements above we designed and implemented a system based on the concept shown on Figure 3.1. Answering **RQ1.1** (Section 1.3) the idea behind our solution is to provide a *configurable* data-collection application. Citizen scientists are able to use the application from different devices (computers, smart phones, tablets) to submit their contributions and the organisation can easily see and retrieve the data that has been collected. The *configuration* is performed to specify the expected fields that will be part of the data collection interface provided to the citizen scientists, which varies across campaigns, and thus should be defined by (or with) a scientific expert.

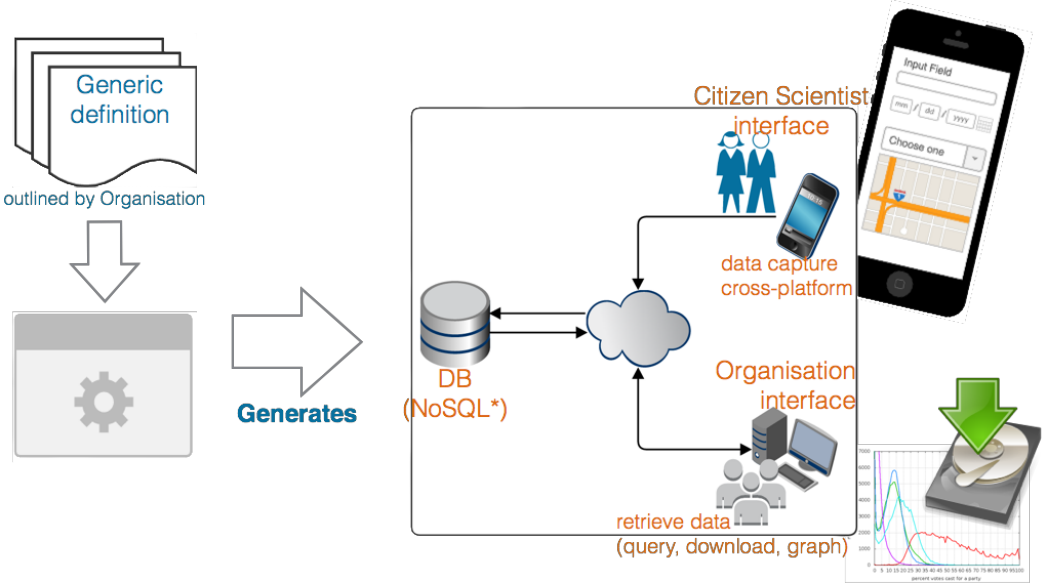


Fig. 3.1 Framework concept

We provide an example to better explain the concept of the framework: An organisation wants to execute a data-collection CS project where citizen scientists can contribute from their mobile devices and personal computers. They also want ownership of the collected data and to avoid so far as possible the complexity of development and technical activities. Our solution shall provide a mostly-ready platform for data collection which, given a generic definition of the desired data-model that is shown to the citizen scientists, generates the supporting systems that will allow the CS project to begin their data collection activities. The CS organisation is only concerned about defining the form that will be used during the campaign to which database and end user data-models adapt. Citizen scientists interact with a cross-platform data capture interface that matches the generic definition and their contributions are stored in a back-end database. The organisation can retrieve the collected data from the database and make use of it in other activities of the CS project.

3.3 System implementation

The following section is one of the larger sections in the thesis and it deals with the important phase of deciding upon a specific implementation for our proposed application. In the next paragraphs we will provide a summary of the key implementation decisions which were made. After this we shall go into greater detail on specific implementation decisions such as database type, front-end application frameworks, etc.

3.3.1 Short summary of our system implementation descisions

One of the key aspects of the design stage of this project was the investigation and evaluation of different implementation alternatives prior to the design and development of our final solution. Table 3.2 (on page 28) summarizes this analysis under our defined set of criteria and Section 3.4 shall contain more detailed discussion on the important design decisions. Figure 3.2 is a schematic diagram representing the overall system architecture. A CS organisation may make use of the external web application called *Form Builder* which we developed to simplify the creation of the configuration file. The *Form Builder* application provides a drag and drop interface where users can configure the fields they want to include in their data collection application by choosing from a predefined set of widgets or input fields.

We provide an open source web application for data collection which takes a JSON configuration file allowing for the definition of the schema/model of the form shown to the volunteers during the CS campaign. This application is carefully designed. It is not tightly coupled to the overall system to provide more opportunities for replaceable entities depending on the needs of the organisation during a CS campaign. Persistence is handled by the database. We chose MongoDB as the database and the reasons for this implementation design shall be explained in Section 3.4.1. As data is collected during a CS campaign the individual observations or records are stored as JSON documents within the database. The structure of these documents is defined implicitly in the JSON configuration file. Each field is represented as one property. To support better data management and data quality activities by default the application adds two additional properties namely the timestamp of submission of the contribution and the originating IP address.

To provide programmatic access to the database a back-end server manages access to the database and exposes the database via a REST API. We implemented and provided a back-end application using Node.js. The *GET* and *POST* methods are provided in our implementation. When using forms in the citizen scientist application which expect file upload submission the back-end application manages their upload. In the application files which are uploaded are stored on the same physical machine that is running the back-end application. The specific target upload directory/folder can be configured in an environment variable and can be easily adapted to store uploaded files in a different location or even a different machine.

We implemented a front-end application using Open Source components. The Angular.js JavaScript framework (Google, 2010), the Angular Formly module (Dodds, 2015) and the

Bootstrap framework (Twitter, 2011) were used to create and deliver the front-end application. Our front-end application provides a user interface for data collection which is generic and is built according to the data model specificed in the JSON file. With this design we answer **RQ1.1** as the common needs of organisations and CS campaigns have been abstracted in a way such that the only place where the project specific details have an impact is the configuration file that defines the desired fields for the CS campaign. Our application uses these web technologies in order to have a single codebase which is properly and effectively displayed across different devices and operating systems. A key technology involved in providing this functionality is Media Queries which are defined in W3C (2012). Figure 3.5 displays the data collection interface for an example application as displayed when accessed from a desktop and mobile system. This is directly related with **RQ1.2** (Section 1.3).

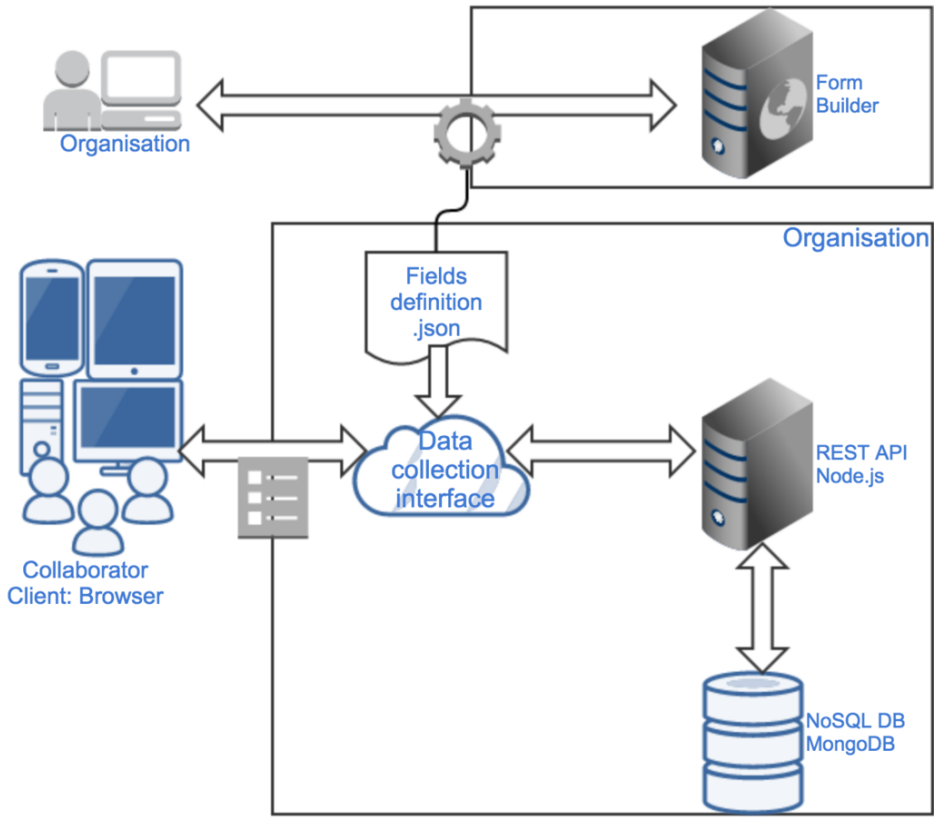


Fig. 3.2 A schematic diagram illustrating the overall system architecture and implementation details

3.3.2 The Form Builder

This application is based on the HCI for CS success principles laid down by Kim et al. (2013) and Preece and Bowser (2014). Figure 3.3 shows the layout of the form builder web application. The panel on the left contains the widgets that have been selected and serves as a preview of the form. The panel on the right contains the supported widgets which are

separated in two tabs. A widget can be dragged from the right panel and dropped on the one on the left. Widgets on the left can be reorganised by moving them on top (or below) each other. The widgets are grouped in two categories. *Default* contains the basic form input components and *Advanced* groups more advanced types of input that usually require some other configuration or permissions. All widgets (except the File input) have a **Column Name** attribute that has to be unique and represents the name of the property given to the value entered in the document that gets stored in the database. The input fields can be further extended to include frequently used fields such as a predefined list as the Ireland County Selector widget.

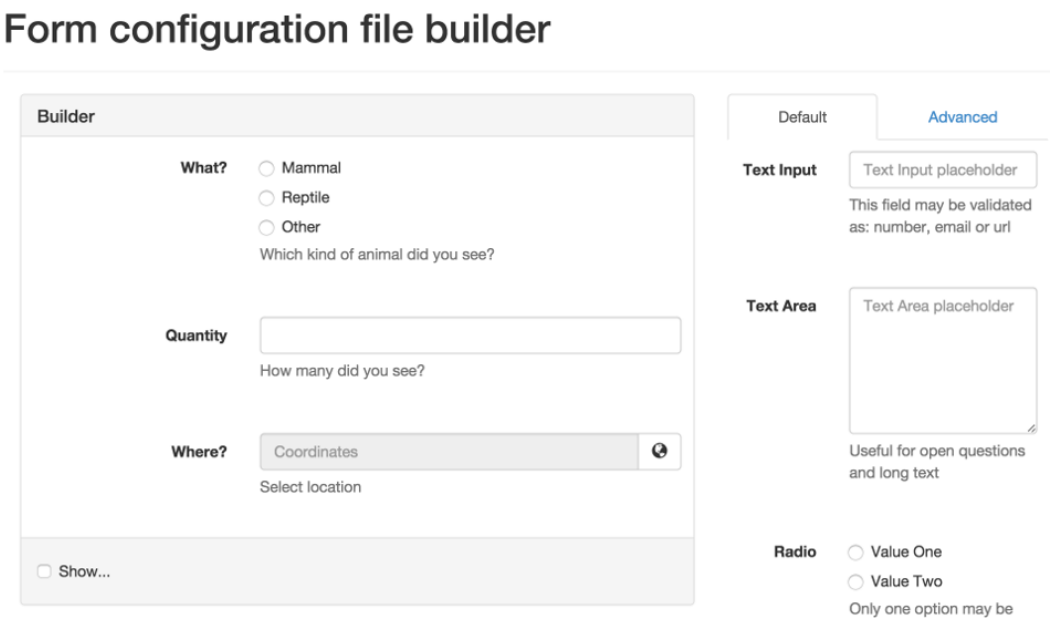


Fig. 3.3 A screenshot of the Form Builder Interface showing the form preview and the support widgets available for selection on the form

- **Default Widgets provided**
  - **Text** defines a text field for which the following attributes can be modified: label, description and placeholder. In addition, validation properties such as if it's required or not and the type of data it should contain (i.e. text, number, url, email) can also be specified.
  - **Text Area** defines a text area, useful for open questions and long free-form text. The following attributes can be modified: label, description and placeholder. A required validation may also be enabled for this type of input.
  - **Radio button** defines a radio select field, where only one option out of many can be chosen. The following attributes can be modified: label and description, as well as the different options with their value and label pairs.

  - **Select** defines a select drop down field, where only one option out of many can be chosen. The modifiable attributes are the same than the radio button: label and description, as well as the different options with their value and label pairs.
  - **Checkbox** defines a checkbox field, which behaves like a tick box with a boolean value. The following attributes can be modified: label and description. A required validation may also be enabled for this type of input.

- **Advanced Widgets provided**

  - **File** defines a file upload field. Any type of file can be uploaded as we decided not to include an image-only restriction considering it can be useful to upload sound files with recordings or videos, and other types of data. Mobile browsers give the option, when interacting with this type of field, to upload an existing file or create a new one from the camera or microphone. Unlike, all other types of input the column name may not be modified through the interface and it defaults to 'files', this is because of how the form submission with a file input is processed in the back-end application. The label and description attributes can be modified. A required validation may also be enabled for this type of input.
  - **Date** defines a field that validates if the value entered is a valid date and provides a date-picker in mobile devices to ease up the interaction. A restriction is set by default in the data collection application to only allow dates up to the current date, but this can be configured by the organisation. The following attributes can be modified: label and description. A required validation may also be enabled for this type of input.
  - **Location** defines a text field that cannot be modified and has a button that when clicked will obtain the current position of the user making use of the HTML5 Geolocation API and update the content of the text field with the coordinates. This is useful when the citizen scientists are expected to submit their observations in-situ and assists in controlling false or erroneous input of geographical location. The following attributes can be modified: label, placeholder and description. A required validation may also be enabled for this type of input.
  - **Map** defines a text field that cannot be modified and loads a map that defaults to the user position. The user may select a different location on the map which will be updated on the text field.A globe button will hide the map (until pressed again) to free up space and provide a better experience. This is useful when the citizen scientists are expected to submit their observations after some time or they are expected to give an specific location. The following attributes can be modified: label, placeholder and description. A required validation may also be enabled for this type of input.
  - **Ireland County selector** defines a select drop down field where only one option can be chosen. There are 32 options available one for each county in Ireland. Only the label and description attributes are modifiable. This widget is an example of how common fields can be defined in the form builder and reused across different

data collection applications. This type of widget can be replicated for other sets of input data (species list, regional administrative units, etc) which are required as code-lists by CS projects.

The *Form Builder* outputs a JSON document (Figure 3.4) that contains the definition of the data collection form, and complies with the Angular Formly specification. Angular Formly is an open source module for declaratively creating forms in Angular.js which is used in the front-end of the data collection application. It "brings unmatched maintainability to your application's forms" Dodds (2015).

3.3.3 Back-end application

The back-end application is the linkage between the front-end application and the back-end database. The back-end should expose two REST endpoints. One of these REST endpoints is used for processing new contributions and submissions. It also handles file upload. The other REST endpoint is used for retrieving basic statistics about the contributed data from the back-end database. The back-end application can feasibly be implemented in whichever language or technology the organisation decides. However for the purposes of demonstration and accessibility for organisations without specific IT skills we provide an implementation using JavaScript and the Node.js framework (Dahl, 2009). Node.js applications can be run on different architectures and operating systems (e.g. Windows, Linux, OS X and other Unix-like OSs). Our back-end application uses Restify (Cavage, 2012) which is a Node.js module designed to build REST API services. Our application exposes REST endpoints for `POST` and `GET` and requires some configuration to be done before being usable. These configuration details are as follows: specifying the desired target upload directory for files and the database details including the host address, port number, database name and collection and database user and password. The last two (authentication parameters) are expected to be set up in environment variables. This follows security best practices to keep this information secure and outside of the source code and configuration files. We advise that the database user is configured to only have read, update and insert privileges on the specific database used by the application.

When a contribution is `POST`ed to our back-end application the IP address is added to the contribution object on the `_ip_address_` property. In addition to this if the contribution contains a file it is automatically renamed adding the `timestamp` of the contribution as a suffix. This is done to avoid overwriting previously submitted files, which could happen, for example, from iOS devices that submit pictures with `image.jpg` as filename. It also provides an easy way to link the file uploaded to the specific contribution.

3.3.4 Front-end application

The front-end application is where the citizen scientists interact with the application to make their contributions and view statistics about the status of the current campaign. The front-end application displays a form for data collection which is built directly from the `form_fields.json` file which contains a declarative version of the expected form. The

contents of this file can be interactively created using the Form Builder application. The Form Builder application is outlined in Section 3.3.2 page 17. A form designed with the Form Builder application is shown in Figure 3.3. Its corresponding output is shown in Figure 3.4 and the generated data collection interface as rendered in two different browsers is shown in Figure 3.5a and Figure 3.5b.

The data collection application is implemented using Angular.js. Minar (2012) explains how Angular is not limited to any specific architectural pattern such as MVC (Model-View-Controller), MVVM (Model-View-ViewModel) or MVP (Model–View–Presenter). However the author does explain how this is closer to MVVM and it is this architectural pattern which we use in our application. MVVM was first introduced by Gossman (2005) as a pattern that derives from MVC (Model-View-Controller) and intends to provide a better separation of concerns between UI and business logic code through the use of data bindings.

When an Angular application starts it compiles the template in the browser and produces a live view that takes the role of the ViewModel. The template is simply a HTML document with Angular's annotations. The View becomes a projection of the model through the live view and every change that happens on the View is propagated to the Model and vice versa. This is shown in Figure 3.6 which is part of the Angular documentation. In the contribution section of our application the form is bound to the `formData` object in the model. Every field on the form is represented by a property of the `formData` object. As a result a binding to the model exists for each field and any change to them on the View will be propagated to the underlying property of the Model's `formData` object provided that it is a valid value.

How the form submission template page works

The contribution form is a webpage with two buttons at the bottom of the page. One of these buttons is used for submitting the contribution as form data and the other button is for resetting the form. The 'Contribute!' button is disabled if the form is not valid. A form is not valid when at least one of its child elements is not valid. An input element is invalid if it is required and it is not assigned a value or if it does not pass any other validation tests which have been set on it. Validation tests include assigning the correct type of data for text inputs (number, email, url) or if the value provided is outside the defined range for a date input. When a contribution is submitted a `timestamp` property is added to the `formData` object, which is then `POST`ed to the endpoint in the back-end application (Section 3.3.3). Both buttons are temporarily disabled while the response is received to avoid a mistake from the user and a spinner is shown to indicate that some processing is being done following interaction design recommendations explained in Cooper et al. (2014). If the contribution is successfully recorded the form is cleared and the user is notified. If an error occurs a notification is shown and the content of the form is not affected allowing the user to submit it again without losing the time and effort he/she spent entering the data.

The statistics view intends to display data about the project, which should be tailored to the detail that the organisation wants. It is a common practice in CS projects to have some public aggregated data given that it has been found to aid in keeping the citizen scientists engaged on the project while keeping specific (or important) data private. In our application the total number of contributions, the number of contributions that have been registered in

the last 30 days and the time elapsed since the last contribution are shown. In addition, the last 10 contributions are also displayed. We think that the level of detail shown on this view is very project specific, which is why both aggregated and raw data have been included in our implementation, for it is our intent that the organisations will further refine it to their needs.

It is required to configure the endpoints exposed by the back-end application (REST API) that the data collection application will use. This is done in the `restAPI.js` file which contains the service that handles all the interaction with the back-end REST application server and exposes two methods, one for submitting and one for retrieving data which are used by the contribution and statistics views, respectively. It is also recommended to use and configure a web analytics solution that gives an insight on user activity on the web application. We include in our implementation a Google Analytics (Google, 2005) tracking script that requires minimum configuration if the organisation decides to use this particular service.

3.4 Design decisions involved in our implementation

There are two major design decisions that we had to take given the large number of alternatives available. The first one is regarding the data storage layer as traditionally applications have used relational databases (RDBMS) for persistent data. Alternatives have emerged under the NoSQL model offering new characteristics and a new paradigm to consider. The second one concerns the UI model as different options exist given the variety of user devices and ways in which users can interact. These modes of interaction are mainly through an application or a website and there are different alternatives for each. We have evaluated some of these here.

3.4.1 Data storage design decision

NoSQL and the Document Store model

NoSQL databases are characterised by their flexibility when compared to relational databases. Sadalage and Fowler (2012) identify as common characteristics the following:

- Not using the relational model
- Running well on clusters
- Open-source
- Built for the 21st century web applications
- Schemaless

We chose to implement our solution by making the design assumption that the organisations will use a NoSQL database to store their data. This decision was mainly based on the open characteristic of NoSQL databases, their flexibility to handle dynamic data models, scalability and their increasing popularity in the web and enterprise contexts. Using a database that

Table 3.1 Google search result count for different Document Oriented DBs

DB name	Number of results
ArangoDB	19,800
Cassandra	1,070,000
Couchbase	374,000
CouchDB	454,000
MongoDB	1,110,000

does not enforce a schema structure on the data results is an important consideration. This removes concerns related to the data layer when a change in the model must be made. NoSQL databases are categorised based on the structure of the data they store in one of the following four ways: Key-Value, Document store, Wide Column and Graph. We decided to provide a solution that uses the Document store model because it matches nicely the type of information that is collected on CS campaigns. Document store databases define a **document** data structure as a complex unit that encapsulates data that belongs and is stored together. Table 3.1 contains an alphabetically ordered list of some document oriented databases in addition to the number of results returned on Google when searching for their name in addition to the term 'nosql', used to filter unrelated results.

The MongoDB NoSQL Database Option

We chose MongoDB as the default database for our proposed solution. This decision was made based on its good performance as found by Li and Manoharan (2013). MongoDB also provides support and facilities for replication (which provides redundancy, required for high availability). MongoDB is also very popular, as seen on Table 3.1. It supports different operating systems and cloud platforms and has a comprehensive documentation, found in The MongoDB Manual (2015). MongoDB is a document store database that stores documents as BSON (binary JSON) objects. BSON is specified in Dirolf (2010). JSON is defined in Crockford (2006) as "a lightweight, text-based, language-independent data interchange format". It is currently, next to XML, one of the most widely used formats for data exchange.

MongoDB runs JavaScript as a native language and fits nicely with the rest of the proposed architecture, being the 'M' in the **M**EAN web stack, Haviv (2014). Users of our solution can run an instance of MongoDB locally or in the cloud without any licensing costs; they can also decide to operate on a PaaS solution that hosts and manages the instance for them. We do not enforce any of these alternatives. It is up to the individual or organisation making use of our solution to evaluate according to their circumstances and their CS campaign requirements. The database details must be configured on the back-end server. The only physical restriction imposed is that if the database and the back-end application are on different machines it is required that a network connection can be established between them.

Data may be exported into JSON or CSV(comma-separated-values) files using the provided mongoexport utility (The MongoDB Manual, 2015). This addresses **R6** and may

be of use to organisations when they want to migrate or analyse their data as the CSV format is commonly used for importing data into relational databases and data processing tools (e.g. Microsoft Excel, MATLAB). It also provides a very easy way for experts in the organisations to access the data which has been contributed to a CS project and subsequently manipulate this data in their own software.

When we were developing and running the case-study CS campaign to test our solution we used mongolab (2011), a service that offers MongoDB-as-a-Service and whose free Sandbox tier may be good enough to suit most small campaigns. It has a capacity of up to 500MB and removes the need of setting up the database instance. However, the data gets stored remotely in one of their IaaS partners (i.e. Amazon Web Services, Google Cloud Platform or Microsoft Azure). This is not suitable for everyone but for CS organisations it is an option which should be investigated if there are no data location restrictions on the project.

3.4.2 User Interface model design decision

One of the first design decisions we had to make was which was the best approach to take for the user interaction. This was not exclusively from a user experience point of view but also from a software maintenance perspective. Native applications could have been generated for the desired mobile platforms as others have done before (Section 2). This would have the cost of maintaining different software projects. Hybrid applications, based on popular engines such as Cordova (Apache, 2009), offer the possibility to have a single code-base that uses web technologies (HTML, CSS, JavaScript) while having access to native functionalities on each device such as camera access or GPS location. Cross-platform frameworks, like the Xamarin Platform (2009) or Titanium (Appcelerator, 2009) give the possibility to have a single code-base from which native applications are created. Finally a mobile web application approach that differs from the other options in that it is and not an application in itself but a website running in the browser application. We have assessed these options under a point system evaluation method with a defined set of criteria considering:

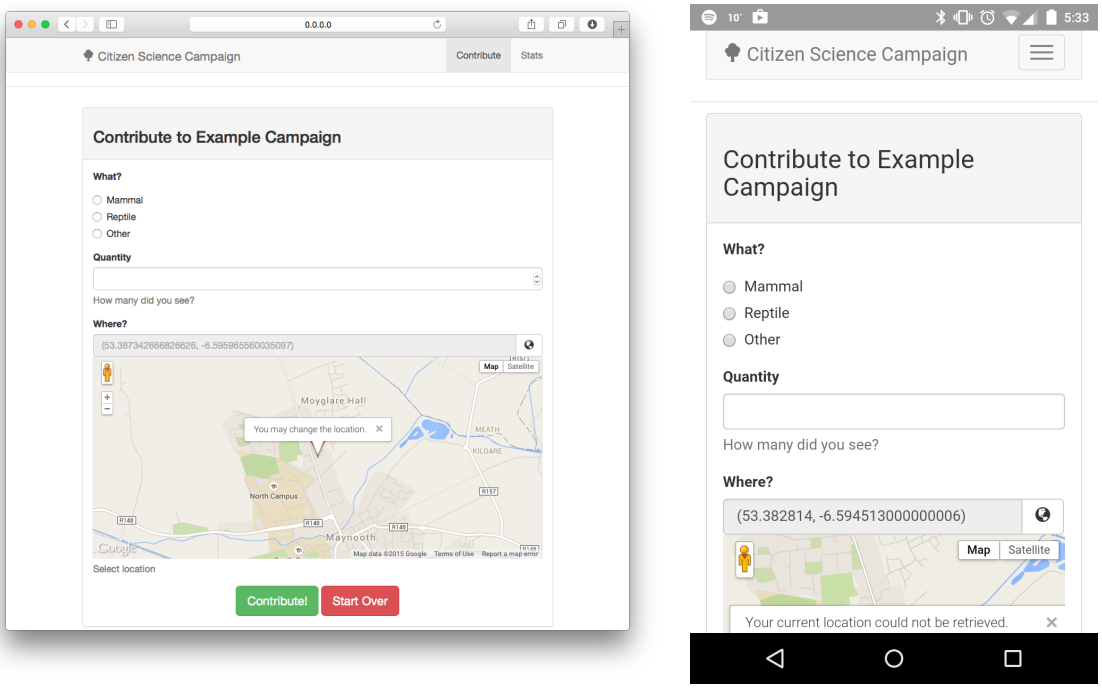
- Attributes of the software, such as performance and resource footprint
- Re-usability in terms of characteristics that reduce the effort of implementing for different devices such as different platforms supported from a single code-base
- Adaptability and extendibility which evaluate the complexity and cost of using the alternative such as how steep the learning curve can be and if its ecosystem is open source and free
- Technical aspects aimed at the development and testing activities such as the different programming languages that are involved, the tooling and debugging experience in the ecosystem and the independence of a marketplace.

Each concept was evaluated on a scale from one to three. One being the worst-case, meaning no support or drastically bad when compared to other alternatives; two being partial support

or no significant difference when compared to the rest; and three being fully supported or significantly better than the others. The result of this evaluation can be seen on Table 3.2 and a description of each alternative with its characteristics follows.

```
[
  {
    "key": "type",
    "type": "radio",
    "templateOptions": {
      "label": "What?",
      "description": "Which kind of animal did you see?",
      "required": false,
      "options": [
        {
          "name": "Mammal",
          "value": "M"
        },
        {
          "name": "Reptile",
          "value": "R"
        },
        {
          "name": "Other",
          "value": "O"
        }
      ]
    }
  },
  {
    "key": "quantity",
    "type": "input",
    "templateOptions": {
      "type": "number",
      "label": "Quantity",
      "placeholder": "",
      "description": "How many did you see?",
      "required": false,
      "options": []
    }
  },
  {
    "key": "location",
    "type": "mapLocation",
    "templateOptions": {
      "label": "Where?",
      "description": "Select location",
      "placeholder": "",
      "required": false,
      "options": []
    }
  }
]
```


Fig. 3.4 The JSON file output from the Form Builder application. This segment shows the creation of a set of radio buttons, an input text box and a map location selector



(a) Desktop browser (b) Mobile browser

Fig. 3.5 Screenshots of the data collection interfaces on desktop and mobile-based browsers

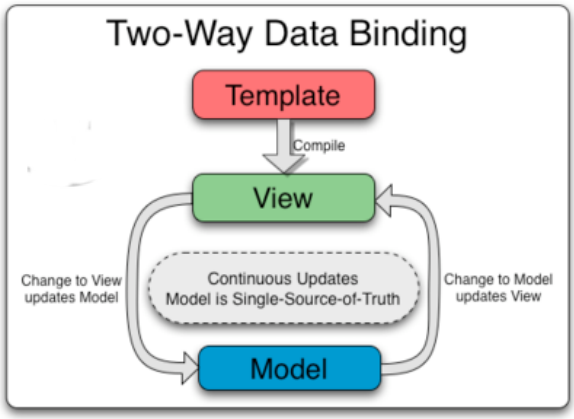


Fig. 3.6 An illustration of the concept of data binding in the Angular.js framework

Table 3.2 Evaluation of front-end alternatives

	Platforms Supported						Programming Languages						Technical aspects		Adaptability & extendibility				Alternative Attributes	
[**score**] Alternative	iOS	Android	Windows Phone	BB OS	Desktop	Marketplace independent	Java	Obj-C	C#	JS	JS Framework	Other	Debugging	Tooling	OpenSource	Free	Learning Curve	Single Code-base	Performance	Footprint
[**40**] Native application	3	3	3	3	1	1	1	1	1	2	3	2	2	2	2	2	1	1	3	3
Hybrid application																				
[**39**] Ember-cordova	3	3	1	1	1	1	3	3	3	1	1	3	2	1	3	3	1	2	1	2
[**43**] Ionic	3	3	2	1	1	1	3	3	3	1	1	3	2	2	3	3	2	2	2	2
Cross-platform framework																				
[**39**] Titanium	3	3	2	2	1	1	3	3	3	1	1	1	3	3	2	1	1	2	2	1
[**43**] Xamarin	3	3	3	1	2	1	3	3	1	3	3	3	2	3	1	1	2	2	2	1
[**51**] Mobile web app.	3	3	3	3	3	3	3	3	3	1	1	2	2	3	3	3	3	3	2	1

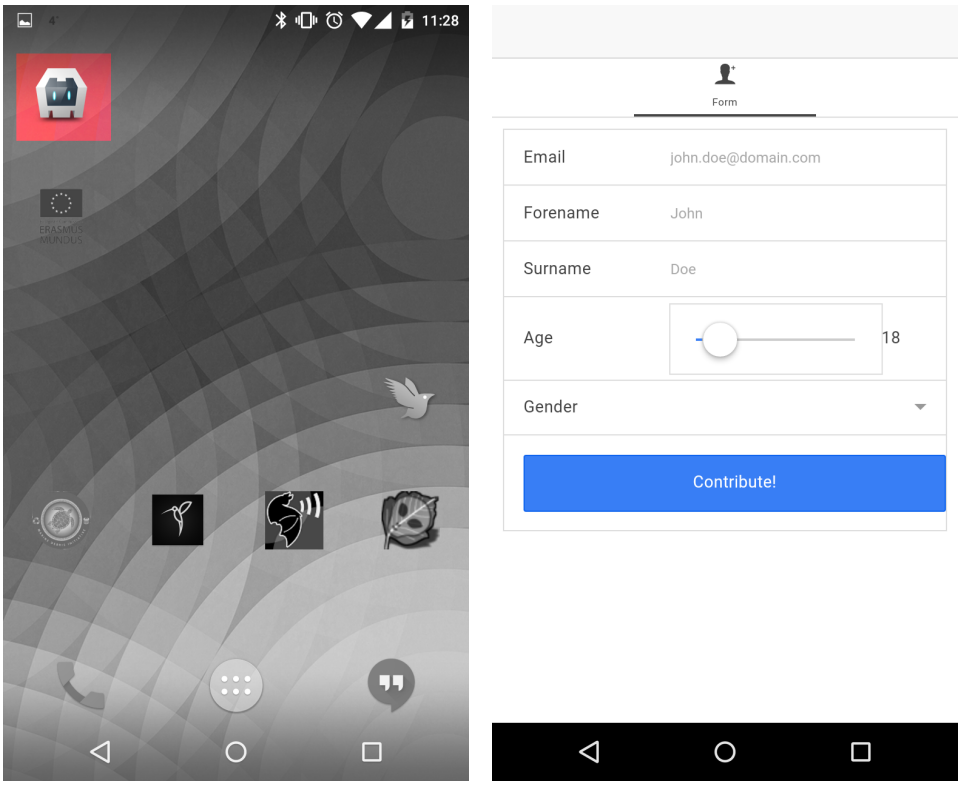
Native applications

Native applications must be developed for each platform and they offer the highest possible performance with the smallest storage footprint as the application package contains just the native code with minimal configuration files. It is very common to have independent software projects for each platform that is being supported. However in these situations not only does a different programming language have to be used for every platform but also the whole development stack and tools must change from one platform to the other. For example, iOS applications have to be developed on a Mac using Objective-C or Swift while the Windows Phone SDK requires Visual Studio, which is only available for Windows, and the recommended programming language is C#. Because of this it is hard to reuse code or implementation of functionalities under this alternative. The created applications are mostly distributed through the specific marketplace of each platform, App Store for iOS, Google Play for Android, the Windows Phone Store and the Blackberry App World. Ultimately, native apps are difficult and expensive to maintain for small teams as the programming languages and development stack and tools differ between platforms.

Hybrid application: Cordova

Hybrid applications use web technologies (HTML, CSS, JavaScript) with a layer of abstraction to access native APIs. The application which is installed on the mobile device is a web application with a native wrapper that runs on a WebView. A WebView can be understood as a browser without the URL bar. These applications must be distributed through the marketplaces of each platform. The main motivation of hybrid application development is code re-usability as the more code that can be written in a platform agnostic manner the less code that needs to be rewritten. Phonegap, now part of the Apache Software Foundation and known as Cordova (Apache, 2009), first appeared in 2009 and simplified the creation of applications for iOS, Android and Blackberry providing the means to access native APIs through JavaScript. This removed the dependence on different software stacks and programming languages. However it comes at a cost of a 'heavier' application in terms of disk usage and performance. We evaluated two alternatives that use Cordova together with JavaScript frameworks to develop hybrid mobile applications.

Ember-cordova is available at Poetic (2014). It is an open source project that exposes an addon for the Ember.js (2011) JavaScript framework. Ember is characterised by favouring convention over configuration imposing community agreed best practices and concepts on all projects. In addition to the Ember conventions development of hybrid applications under ember-cordova makes use of the provided tooling of `ember-cli` to deal with different environments and building activities. There is a big risk in that `ember-cli` is still under heavy development and the official repository even states that "*backward compatibility with older ember-cli versions will not be focused as it's moving too fast and the API is constantly changing*". Android and iOS platforms are supported and the required technical knowledge is mostly around JavaScript and Ember. The tooling is not ideal as the API is still not stable which also makes learning the tool complicated given the constant changes, new features and consequently outdated tutorials/documentation. We used two metrics to measure the



(a) Application icon on home screen (b) Data Collection Interface

Fig. 3.7 Ionic Proof of Concept

popularity and size of the community around an open source project: a project's star count on Github and the number of contributors. Table 3.3 contains these numbers for the projects we evaluated and they are plotted in Figure 3.9. Ember has 13850 stars on Github with 494 contributors while ember-cordova has 278 stars with 17 contributors.

The **Ionic (2014)** Framework is another hybrid approach that builds on Cordova. Ionic is open source and free and uses web technologies in addition to the patterns of the Angular.js framework. The only fully supported mobile platforms are iOS and Android while there is partial support for Windows Phone. Ionic focuses on good 'beautiful' design, performance and development tooling. Angular (Google, 2010) plays a big part on the development of hybrid applications with Ionic. Given the popularity of the framework, the more stable state of the project and the great quantity of available material around it we believe that the learning curve is not as steep as Ember's. Angular is, at the time of writing, the third most starred project on Github with a star count of 38959 and 1236 contributors. The Ionic project has 16826 stars with 174 contributors.

Ionic was second in our ranking and we had interest in its offerings. We decided to have an early proof of concept built in Ionic to better assess its suitability in our model. Figure 3.7 shows on the left the icon of the application after it is installed on an Android device. On the right the data collection UI displaying the Ionic app wrapped in a native application. The size of the Android application exceeded 4 MB which is large given the basic features that were

implemented. The dependence on a marketplace to be able to push changes discouraged us from using Ionic. For example when we wanted to change the form in our proof of concept application we had to build it again and deploy it on the device. This is troublesome and each platform has their own rules and regulations for how long this process takes and under which conditions an application can be updated on their marketplace.

Cross-platform Frameworks

Cross-platform frameworks are different in that they, unlike hybrid applications, do not wrap a web application inside a native one and display it on a WebView. Instead they are evaluated at runtime by an interpreter written in the platform's native language (Objective-C on iOS or Java on Android). Because of this applications also have to be distributed through platform-specific marketplaces and the footprint is increased as the runtime is bundled with each application. Another distinctive feature of this approach is that native views and components are preferred over Web UI technologies (HTML/CSS). The trade-off results in offering a more native-like experience at the expense of reduced code re-usability as this code has to be written for each platform.

The **Xamarin Platform (2009)** supports iOS, Android, Windows Phone and Mac applications. Using Xamarin is not entirely free, as a license has to be bought for each platform with a subscription fee (at the time of writing) of $25 per month. This license allows the use of an IDE that offers tools such as code completion, debugging and a built-in UI editor. Applications are written in `C#` and the API is similar to the one of `.NET`. On some platforms Xamarin applications must embed the runtime to function properly. The size of the runtime is approximately 2.5 MB. Given that Xamarin is not open source and not entirely free we had to decide against it as it strictly opposes requirement **R7** of our project (Section 3.1).

Titanium (Appcelerator, 2009) runs applications written in JavaScript as native executables. An IDE, Titanium Studio, is free and included for development activities. It also includes tools to build and publish the developed applications. Although not strictly required applications are recommended to be developed under Alloy, a MVC framework built on top of Titanium. Views are designed in XML documents. The latest version supports iOS and Android while Windows Phone and Blackberry are under development Appcelerator (2014). An interpreter is required which maps the JavaScript code to native code at runtime and has been found to have a perceivable performance impact. Alloy (2012) has 881 stars on Github with 43 different contributors while Titanium (2009) has 1857 stars and 130 contributors.

Alloy (2012) has 881 stars on Github with 43 different contributors while Titanium (2009) has 1857 stars and 130 contributors.

Mobile web application

Mobile web applications are web applications designed primarily to be accessed from mobile devices with relatively small screens. As such they are accessed via a URL instead of being distributed through a Marketplace. This means they are not bound to a specific platform and can be used from anywhere running a browser (e.g. Personal computers). Excluding caching and local storage optimizations considerations the content is downloaded from a

Table 3.3 Popularity of open source technologies on Github

Project	Contributors	Stars
Angular	1236	38959
Ionic	174	16826
Ember	494	13850
Ember-cordova	17	278
Titanium	130	1857
Alloy	43	881

remote server every time the application is accessed (which can be considered a performance weakness) and users feel and know that they are interacting with a web page instead of a native application. We have already mentioned that HTML, CSS and JavaScript are common web technologies that are also used by some of the other alternatives. In mobile web applications these technologies are used to optimize user experience in a responsive way based on the resolution of the view in which the application is being displayed. Through the use of CSS Media queries (W3C, 2012) the content layout can be changed based on the browser's dimensions. The HTML5 specification (W3C, 2014) was finished in October 2014. It includes new APIs that augment the possibilities of what can be done in the browser. Some of those APIs were decisive in our project. The Geolocation and File API are supported by mobile browsers meaning that web applications can use the GPS sensor to retrieve the location of the user and the camera or microphone to create a file for uploading. This is possible after the web application requests permission to perform these actions and the user authorizes it. This fulfils requirement **R3** from Section 3.1. Figure 3.8 shows an example of the prompt displayed by the Safari browser on Mac OS X when the web application requests access to the user's location through the Geolocation API. We found in web applications a way to answer **RQ1.2** (Section 1.3), as the web's architecture allows to deploy an application and its resources on a server that is then contacted by clients in different platforms and devices.

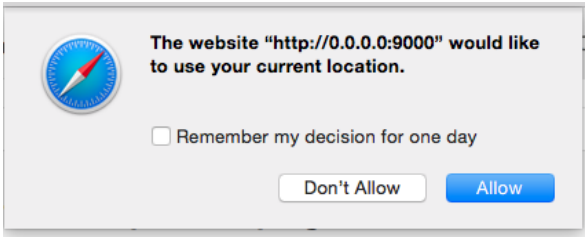


Fig. 3.8 An example of the user being asked to allow the application to access their location from their device

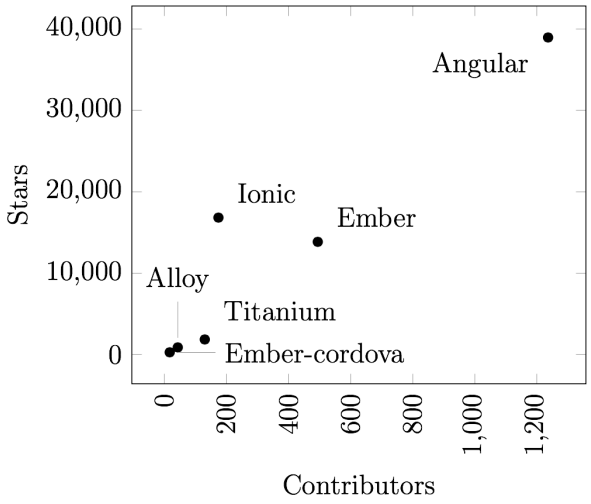


Fig. 3.9 Popularity of open source technologies on Github

3.4.3 Chosen UI model for implementation

We chose the mobile web application option. This option outscored all other alternatives on average by 25%. An important advantage is that it is the only marketplace independent alternative. This makes it more adaptable than the other options. When a change is required (e.g. a new field is added) the updated UI needs to be deployed only on the web application server. With all other alternatives installed as an application locally the application must be modified, uploaded to the different marketplaces and then downloaded again on each device before a CS project could update their applications to the new version. This might be something that they are reluctant to do or are unaware of. This overhead is non-existent in the web application approach. Another advantage is that the same application enables CS volunteers to contribute from any platform that is Internet connected and runs a browser. This includes personal computers extending single code-base support on multiple platforms outside of the boundaries of mobile.

3.5 Dependability attributes

Definitions for system dependability have been given by different standard bodies, such as ISO and IEC (Avižienis et al., 2004). The IFIP Working Group 10.4 defines the dependability of a computing system as “the ability to deliver service that can justifiably be trusted” and give the following as its attributes: availability, reliability, safety, confidentiality, integrity and maintainability (Avižienis et al., 2000). In our software application we focused on the availability, reliability and maintainability attributes intending to achieve a good level of dependability.

3.5.1 Dependability attribute: Availability

Availability is often seen as an attribute that does not pertain to the software itself (Meyer, 2006) but to the hardware or infrastructure in which it runs. Nevertheless we did not disregard it. It was taken into account when making some design decisions because as previously stated in Requirement **R4** in Section 3.1 the nature of CS campaigns is such that they might run in very narrow timeframes and the tools supporting the operation of the campaign are expected to be available at all times. The most common means to attain availability is redundancy. This reduces single point of failure dependencies and allows components to be hot swapped. This influenced our selection of MongoDB as the suggested data storage component given its replication facilities (see Section 3.4.1). In addition to this an update of a mobile web application for the front-end or an update to the data-collection component does not depend on the different marketplaces for each mobile platform and then on each of the users updating the application on their devices (see Section 3.4.2). The last point also has some reliability implications as the CS organisation can **rely** only on themselves to update the version of the data-collection interface to all citizen scientists. The provided ability to add or remove elements (input widgets) of the data model dynamically as the CS campaign is collecting data is effectively hot swapping the components of the data collection form. This is a direct result of the design and implementation of our provided application which would have been difficult under canonical web stacks such as LAMP (Wikipedia, 2015).

3.5.2 Dependability attribute: Reliability

We approach the reliability attribute through addressing each of the three factors listed by Meyer (2006). ***Correctness***, the performance of the system according to its specification; ***robustness***, the prevention of damage when accidentally used outside the specification; and ***security***, the prevention of damage when deliberately used outside the specification.

Data bias and false data

Data collection in CS projects is a human activity and, as such, is error prone. This concern has, and continues to be brought up in reports of different campaigns and is collected by Dickinson et al. (2010), who also convey the importance of including data quality assurance activities in CS projects. This should include providing adequate training to the volunteers and adopting rigorous protocols for data collection. Data errors in CS projects have been attributed to different factors such as lack of experience among volunteers, ineffective training and a complex data collection protocol Gardiner et al. (2012). In addition, there is the risk that volunteers might misbehave and deliberately provide false data with intentions different from contributing to the project. Quality assurance of the collected data is an activity that verifies that only accurate data is analysed and is done by trained experts on the CS team running the project. We are aware of how important, and sometimes hard, this activity can be and have taken some measures in our solution to better support it. In addition to the data specified for collection by the organisation, all contributions include a timestamp as of the time of the submission and the IP address of the device from which the submission was done.

We believe this will prove as useful information for whoever will have to filter and identify false or biased data.

We provide two different widgets for location. The **location** widget takes the position as given by the browser, whereas the **map** widget gives the citizen scientist the option to specify the location to submit. If concerned about false location data being provided organisations can use the **location** type of input over the **map**. This does not guarantee that a malicious user will provide false data but it does make harder. The data collection interface is designed in a way such that errors in data are found and informed to the volunteer before the contribution is done. A contribution cannot be submitted unless the mandatory fields have been provided and all fields that have any additional validation tests are passed. For example, the `Quantity` field in our example application was configured to be a number and if the citizen scientist provides a value that is not numeric (e.g. 'one') then they will not be able to submit their contribution while being notified of the data inconsistency as seen in Figure 3.10. Finally, we considered to include authentication in our application as a measure to identify an individual's contributions. We decided against this at this stage because it could be demotivating for some volunteer participation given the inherent tension between privacy and visibility (Erickson and Kellogg, 2000). Many people feel insecure about the data they submit specially if it is being tracked and connected to them when they are not using an anonymous platform. These insecurities are heightened when in addition to their identity the user's location and the time and date of their submission are also being recorded.

Fig. 3.10 An example of a numeric text field which has been supplied with an invalid value

Verification, Validation and Trusted Components

As the user interface for the data collection view varies for every project we provide a test harness with unit tests written using Jasmine. Jasmine is a behaviour-driven development framework for testing JavaScript code (PivotalLabs, 2010). We use Karma Test Runner (2012) that has adapters for different testing frameworks in case the organisation does not want to use Jasmine. The testing harness we provide includes unit tests for the functionality that is present in our implementation and is intended to be used as a guideline for extension by the organisation who can use the existing test cases as examples for testing any changes or new features they add. A `test` task is configured which when run will set up and launch the Karma test runner for running and reporting the results to the console. Figure 3.11 contains an example of the output of the `grunt test` command.

As stated previously Angular as a framework includes many features that ease on the task of testing an application. Through the use of dependency injection, a component's dependencies are passed in and they can be simulated to focus the unit-test on the behaviour of the component. Angular's documentation for unit testing provides a thorough explanation

```
➜  web_client git:(master) ✗ grunt test

Running "connect:test" (connect) task
Started connect web server on http://0.0.0.0:9001

Running "karma:unit" (karma) task
INFO [karma]: Karma v0.12.31 server started at http://localhost:8080/
INFO [launcher]: Starting browser PhantomJS
INFO [PhantomJS 1.9.8 (Mac OS X)]: Connected on socket P9v8yiF0gZAl6AMAsgJ2 with id 26969565
ALERT: 'Thanks for you contribution!'
ALERT: 'Thanks for you contribution!'
PhantomJS 1.9.8 (Mac OS X): Executed 17 of 17 SUCCESS (0.006 secs / 0.115 secs)

Done, without errors.
```

Fig. 3.11 An example of the output of the `grunt test` task

of how each component is most appropriately tested. When writing unit tests it is important to differentiate the application-specific features from those of the framework or the libraries used. Testing the framework or an external module is, in our opinion, a poor use of time and resources. We have made a conscious selection of *verified* modules that include robust test suite supporting their features. These modules are *validated* by the open-source process that guides them. As stated by Meyer (2006) the use of **trusted components** benefit the applications that rely on them directly improving their quality.

3.5.3 Dependability attribute: Maintainability

Maintainable software can be adapted economically to cope with new or changing requirements while having a low probability that such adaptations will result in new errors being introduced. Assessing the maintainability of a given software component is particularly difficult without using it for a long period of time (Sommerville, 2010). Our solution adheres to maintainable software best practices such as a loosely coupled architecture and the use of self-contained components.

Documentation

Special care was taken in properly producing documentation. Updated and accurate documentation allows future software maintainers to better understand and extend the software. Documentation is key in ensuring maintainability of the software. Code has been written and structured in a modular way aiming to make it easier to navigate. Comments have been included where we felt any clarification was useful and we provide a self-generating HTML documentation for the front-end application that is used for data collection. This is done through the use of **ngdoc** (Angular-Team, 2013). ngdoc is a flavour of JSDoc (2011) that generates API documentation for AngularJS projects. JSDoc is an API documentation generator for JavaScript, much like JavaDoc for Java or phpDocumentor for PHP. The code is documented with comments and annotations that are interpreted to generate hyperlinked documents that explain the intention, structure and behaviour of the software components. Figure 3.12 shows an example of the main view of the documentation of our application.

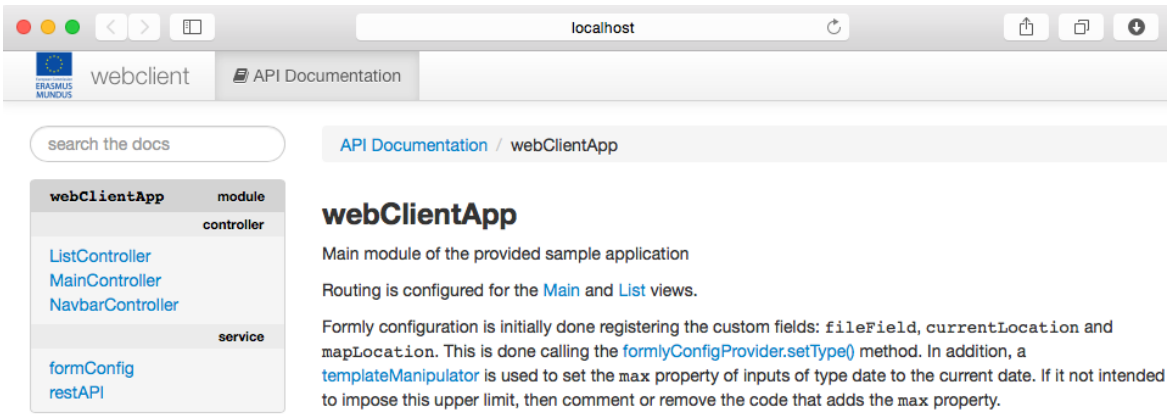


Fig. 3.12 HTML Documentation for the Data Collection app

A `docs` task is configured, which when run will parse the annotations in the source code and generate or update the HTML documentation. This task is also called from the `build` task which effectively updates the documentation whenever a new version of the software is getting 'production ready'. Figure 3.13 shows the relevant output of the `grunt docs` command. From our experience out of date documentation quickly becomes a hindrance in further maintenance efforts and it tends to be relegated from its original first-class citizen status exponentially decreasing its usability. Thus we highly encourage maintainers to keep documentation updated with any new or changing requirements.

```
➜  web_client git:(master) ✗ grunt docs
Running "ngdocs:all" (ngdocs) task
Generating Documentation...
DONE. Generated 6 pages in 64ms.

Done, without errors.
```

Fig. 3.13 An example output of the `grunt docs` task

Open Source

Requirement **R7** (in Section 3.1) imposes a hard constraint on the exclusive use of open source software. We believe that the open and free nature of open source components results in increased software maintainability. We chose self contained components that can be updated individually without affecting other parts of the software (while adhering to the Semantic Versioning specification by Preston-Werner 2011) or that can, in theory, be replaced for a different component that offers similar functionality. Tools and modules that have a widespread use also bring a community that is constantly creating content around these components. The output of this community is more support, better documentation, blog posts and tutorials, etc. All of these combined ultimately benefit the maintainers who can count on a pool of content and individuals to guide their efforts.

Chapter 4

Overall project evaluation

We assess the value of our contribution based on the successful delivery of key requirements, where we also address dependability, and a comparison with state-of-the-art platforms in the field and the benefits of our model over them.

4.1 Delivery of key requirements

In Section 3.1 the main requirements that guide our project were introduced. These requirements are derived from our research questions and the current state-of-the-art in the CS field. Consequently, by fulfilling them we'll be directly addressing our research objectives and a gap in the CS community. For convenience we repeat our requirements from Section 3.1 as follows:

R1 The ownership of the data collection tool shall be given to the organisation.

R2 The data model of the application shall be definable.

R3 Mobile devices shall be supported including the use of camera and location sensors.

R4 The data collection application shall allow the defined datamodel to be modified during a CS campaign.

R5 The application shall prevent, so far as possible, bad quality data from being collected.

R6 The data collected by the application shall be easy to extract.

R7 Open Source shall be a hard constraint of the project.

As a result of **R1** our tool differs greatly from existing platforms which is explained in greater detail in the next section. This requirement is successfully accomplished by providing an open source model which can be cloned and adapted to the needs of the organisation and the project storing their data where they consider best. **R2** was a difficult requirement and based on it we designed the architecture of our solution around a data-model definition file

to which both back and front ends are linked. To assist in the creation of this data-model definition file we include a user-friendly form builder (Section 3.3) that reduces the technical nature of the task. **R3** was easy to fulfill given the adoption and features offered by novel web technologies. It was met with the use of Bootstrap as a mobile-first framework for Web interfaces and the HTML5 Geolocation and File APIs that give access to the GPS and camera respectively. **R4**, like **R2**, was also a difficult requirement concerned with organisations being able to make changes to the data-model once the project is running. It was successfully delivered by the design of our solution as with only changing the configuration file on the Web server subsequent clients can access the modified version. **R5** is successfully delivered by mandatory and data-type validation checks that prevent incomplete or low quality data from being accepted. Biased data can still be submitted. It is up to the domain experts to validate the data which we help by complementing each contribution with metadata. Further validations could be added to our framework to better fulfill this requirement which we pose as future work (Section 6.2). **R6** was completely fulfilled with our choice of MongoDB as a database as it natively stores BSON objects that can be exported to common formats such as CSV. Finally **R7**, while being a difficult requirement, was delivered successfully which led to some design alternatives being ruled out because of it. All components used on our applications are Open Source as is the framework itself.

In Section 3.5 we mentioned Availability, Reliability and Maintainability as the attributes that were taken into account to attain dependability. Some requirements are intertwined with the attributes. The availability of our framework is improved by our choice of a NoSQL database. NoSQL databases are by design intended to support replication and the ability to hot-swap the definition of the data-model with the autonomy of the Web architecture. As previously stated we include data checks and metadata. This improves the reliability of the data collected by the tool. In addition a test harness with automated unit tests can better identify when a breaking change occurs and the use of some *trusted* open source components augments the visibility and coverage of the code that we are using. The use of open source also has an effect on maintainability as the community resources around open source continue to expand. We complement these resources by providing a roadmap into the data collection application with proper documentation.

4.2 Comparison to previous work

In Sections 2.3.1 and 2.3.2 we introduced CitSci.org and Sensr, two existing platforms that, like our framework, seek to support CS projects that crowdsource data collection. In Table 4.1 we present a comparison of the features offered by each platform and the improvements of our approach. The concept behind CitSci.org and Sensr is similar as they both offer a platform where CS projects are created and accessed within. They enclose each campaign and offer limited 'branding' and identity creation capabilities to the campaign. Our concept is different, given that we intend to keep tools and data ownership on the organisation's end, offering the opportunity to use the infrastructure they choose and tailor the appearance of the data collection interface to their needs. This also results in our model being the only model which can offer privacy of the collected data. Volunteers are required to login and

register before being able to contribute to projects hosted on CitSci, which may be useful for uses such as ranking the contributors, linking each contribution to its source and maybe even attempting to reduce false/biased contributions. However, it has been noted that there are privacy concerns regarding the amount and type of personal information obtained from contributions (Krontiris and Maisonneuve, 2011). Like Sensr, our model uses anonymous contributions to ease this issue on the citizen's side (Cohen, 2008), but with the added value of metadata that proves useful for data quality control.

Table 4.1 A comparison with previous platforms

System attributes	Our model	CitSci	Sensr
Data collected is private	✔		
Does not require login to contribute	✔		✔
Customisable data collection forms	✔	✔	✔
Interactive form designer	✔		✔
Diverse and extensible types of input	✔		
Unrestricted use on Mobiles	✔		
Unrestricted use on Desktops	✔	✔	
Add images to contributions and use of camera	✔	✔	✔
Use of GPS – obtain current location	✔	✔	
Use of map – specify desired location	✔	✔	
Dynamic charts and graphs		✔	✔
Data flagging to aid in QC	✔		

While in all systems the data collection form for the project can be designed there are major differences in how this is done and what is possible between them. CitSci does not offer an interactive way to do it, while Sensr has a drag and drop form designer for iOS and we have the form builder (Section 3.3.2) to generate the data model specification. The range of available types of input is limited in both CitSci and Sensr, even more so in the latter, where only text, two-and-three options radio buttons and photo upload can be used (Kim et al., 2013). Our model is designed in a way such that once the data collection interface has been defined volunteers can contribute from mobile or desktop devices without restriction on features. Contributors on CitSci's web application have access to all features of the platform, while on the mobile applications some are not supported. Sensr only offers a data collection application for mobile devices running iOS and the web application is restricted to data visualization features without any means to contribute. A key feature required by the CS community is the use of camera and GPS sensors on mobile devices. All three systems support attaching a file to the contribution and the use of the camera from the device; however, this is not the case with the GPS sensor and the ability to indicate a location as Sensr's limited types of input do not include one that captures the device's position. In contrast, CitSci and our model both offer two alternatives: (*i*) fetch the current location of the device using the GPS sensor and (*ii*) allow the citizen scientist to specify a location on a map.

Both of the other systems have dynamic data charts and graph creation capabilities. CitSci creates charts based on the collected data, while Sensr offers a map view with the location of each contribution. The current version of our model does not include such features, but it is one of the things we have considered for future work (Section 6.2) as data visualization can be of great use to a campaign.

Our framework makes use of novel web technologies to fill the gaps experienced and encountered by previous approaches in the CS field. By extending and adapting our framework organisations gain ownership of the data and tools being used on their projects while having a platform for contributions whose features are seamless across user devices and designed to best support the needs of CS campaigns that crowdsource data collection.

Chapter 5

Case Study: Signs of Spring campaign

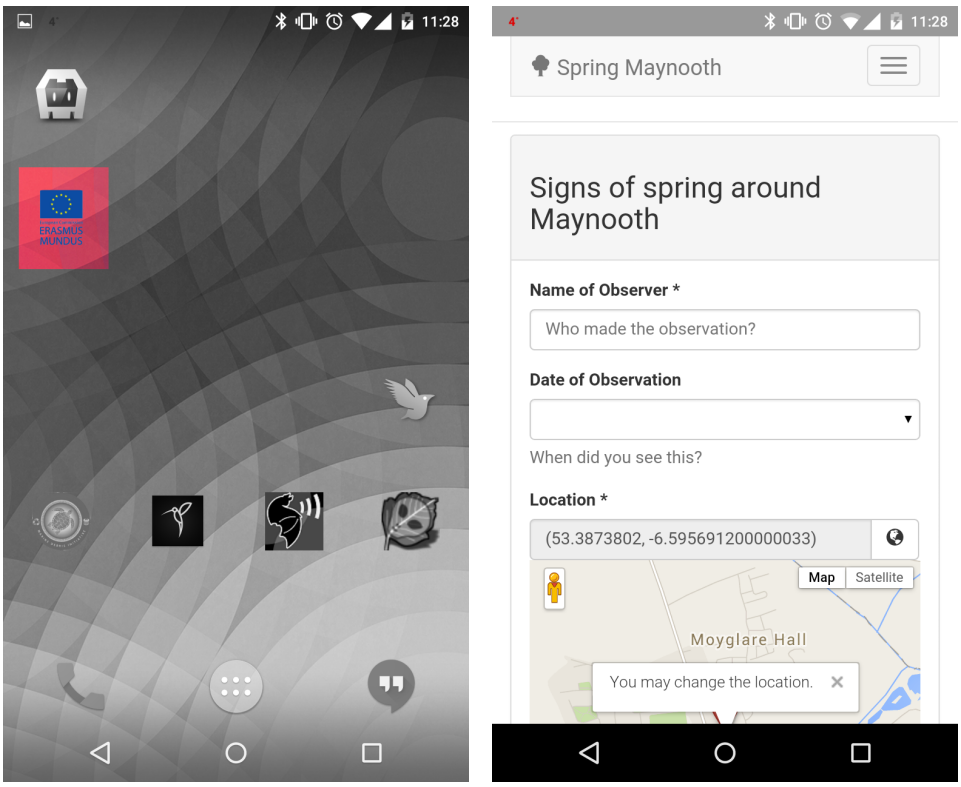
In this section, we present a case study in which we organised a CS campaign that begun on early April and intended to collect the signs of Spring that volunteers could identify around Maynooth. Nine volunteers agreed to participate, but contributions to the project came only from seven of them. Each of the volunteers was instructed on the objectives of the campaign and given the URL[1] to access the data collection application. They were also asked to provide any feedback they had as early as possible instead of waiting until the end of the campaign so that we could react quickly to their concerns thereby improving the overall experience for all participants. With our case study we wanted to: First, see how feasible it was to adapt and deploy a concrete application using our framework. Second, have external users interact with the application from different platforms and devices, and uncover any potential flaws arising from field usage of the application. Thirdly, understand how users interacted with the application, collecting feedback about the perceived usability and any suggested improvements that could be done.

5.1 Case study setup and configuration

We used the Form Builder interface (Section 3.3.2) to design the data collection form for our volunteers to use. The form had seven fields, each of a different category. The first field was of type `text` and inquired the name of the observer, we decided to make this field mandatory to be able to link each contribution to its respective volunteer and get a better understanding of usage and involvement. The second field was of type `date` and asked for the date of the observation. Then a `map` field where an approximate location of the observation was expected. This field was also mandatory. The fourth field was of type `radio button` and offered options about the category of the observation with options: Singing birds, flying birds, flowers or plants, environmental observations (e.g. mud, temperature), mating animals or none of the rest. Fifth, a `text` field of type `number` asked about the perceived temperature at the time of the observation. Next, a `text area` field was available for any remarks. And finally, a field of type `file` provided the possibility to upload a file with the contribution.

[1] `http://maynoothspring.herokuapp.com/`

The front-end application used the output of the Form Builder for the configuration of the form. We modified the default application, personalizing it to the objective of our campaign and updating the endpoints of our back-end application. Then, using the `'grunt build'` task we generated the production application that we later deployed through `git` to the Heroku (2007) PaaS, "a cloud platform that lets companies build, deliver, monitor and scale apps". Given the size of our campaign, the free tier of Heroku offered enough resources to run it without the need of scaling or incurring in any costs. Figure 5.1 shows: On the left, how bookmarking the URL on Android can be used as a shortcut from the home screen, resembling the behaviour of an installed application. On the right, the application being accessed on the same Android device, properly scaling the contents of the app to the screen in which it is being rendered.



(a) Bookmark on home screen (b) Data collection Interface

Fig. 5.1 Signs of Spring Campaign on Android mobile

We also deployed our back-end application to Heroku, where we only had to update the configuration file and set the required environment variables. No changes were needed on the source code. Like with the front-end application, the free tier provided enough to host this application without incurring in any charges.

Our database tier was hosted on mongolab (2011), a MongoDB-as-a-Service platform whose free sandbox level offered a capacity of up to 500MB, allowing us to remotely host a database instance on external infrastructure. A word of caution must be given to organisations interested in doing the same as the data gets stored remotely in one of mongolab's IaaS

Table 5.1 Usage of optional fields in Signs of Spring

Input field	`Type`	Percentage (%)
Remarks	`textarea`	20
Image	`file`	80
Temperature	`number`	90
Date of Observation	`date`	100
Category	`radio`	100

partners (i.e. Amazon Web Services, Google Cloud Platform or Microsoft Azure), which might not fit the boundaries of some projects.

Observations

Most of the recruited volunteers contributed to the project (77.7%), with an average of two contributions per volunteer. All contributions were split between two categories flowers or plants and singing birds, with 80 and 20 percent respectively. Other possible categories such as flying birds or environmental attributes (like temperature change) were not perceived by our volunteers as strong signs of the spring season taking over. Every contribution in the flowers or plants category had an image attached to it, which we found to reflect the social media aspect of sharing pictures over sounds. It is usual to see somebody taking a picture of a plant or flower while it is uncommon to come across someone recording sounds. Table 5.1 contains the relative usage of the optional input fields in the data collection form of the Signs of Spring campaign, calculated for every input field f as

$$percentage_f = \frac{contributions_f}{contributions_t} \times 100\%$$

where $contributions_f$ is the total number of contributions for which a value was assigned to field f and $contributions_t$ is the total number of contributions. These values coincide with previous observations from Kim et al. (2013) who noted that citizen scientists using a data-collection mobile interface, like ours, will prefer to interact with *easy* to use fields. A field is *easy* with respect to the time and number of taps necessary to give it a valid value. The statistics of field usage in our case study (see Table 5.1) are a clear example of this as the `radio button` field was largely favoured over the `textarea` field. We believe this is an important form design consideration for future CS campaigns as a smart design that uses *easy* fields is likely to get more detailed contributions than one that ignores this fact.

5.2 Feedback and commentary

Most of the comments we got from volunteers were regarding the `map` component. Initially, we did not do any configuration on it and had it set to show all Ireland and default the position

to Dublin. One of our volunteers said "*I find confusing that the location marker is set to Dublin, isn't the purpose of your project finding signs of spring on Maynooth?*". Another had similar concerns and said "*I think the default zoom level is too small. It would be helpful if the map loaded on Maynooth*". In this case, little effort on our side meant removing a nuisance on the experience of our volunteers. We changed the default map configuration, setting the default location somewhere near the campus of Maynooth University, a location all of our volunteers were familiar with, and adjusting the zoom level from 5 to 14.

Another volunteer expressed how having a placeholder value on the temperature field (of type `number`) made her think this was a default value. We asked other volunteers on their perception and the general consensus was that it was indeed confusing. We decided to adjust the form configuration file and remove the placeholder property from this field. Besides the previous, feedback was very positive and no usability issues were reported. On the contrary, some expressed that the form felt intuitive and that it "*looked nice*". Aesthetic features are usually undervalued in some projects but they prove valuable in retaining user interest and motivation. Volunteers submitted their contributions from devices running different versions of Windows, Mac OSX, Android and iOS and there were no compatibility or stability problems reported, besides a small hiccup from Heroku, the platform were we deployed our applications, that was unfortunately out of our control.

From what we learned on our case study, we recommend new CS projects, regardless of if they are using our data collection framework or not, to carefully design the form that will be used by their volunteers and to spend sensible time adapting each field to the use that it is intended for. An example of which is our oversight on initially configuring the `map` component, which ended up offering a less than optimal user experience. A benefit of our framework is that this was easy to correct and cheap, in terms of time spent. Another recommendation would be to conduct a *closed beta* trial, akin to that of the software release life cycle, where few volunteers are selected to engage with the tool and give early feedback on the usability and overall experience with the application, resulting in a tighter feedback loop that is likely to spot issues before releasing the campaign to a wider CS audience.

Chapter 6

Conclusions and future work

In this thesis we have identified and addressed a gap in the CS literature that arose from new technological inventions and the increasing popularity of 'smart' devices being acquired by citizens. Even though mobile and web technologies have been proven advantageous for data-collection CS projects their adoption rates are still very low. The CS community needs an alternative that, using these technologies, supports the definition of the data-model without relinquishing ownership of the tools and data. Our perception is that this gap could be moulded into an opportunity for the CS community to benefit tremendously, and consequently we designed and developed the *bridge* – an open-source software tool that makes data collection for CS projects easier and more accessible. In this Chapter we provide some overall conclusions on the success of the research and development work presented in this thesis.

6.1 Conclusions

In Chapter 2 we presented our survey of the state of the art in the CS domain, highlighting the most relevant contributions to date. We introduced CitSci and Sensr as the most closely related platforms to our project. In Chapter 3 we defined the key requirements and described the design and implementation phases of this project as well as the dependability attributes considered. Chapter 4 contained the evaluation of the value of our work in two main areas: Fulfillment of key requirements and a comparison with existing platforms. In Chapter 5 we presented a case study, a CS campaign that made use of our framework and asked volunteers to contribute with any signs of spring that they found on the town of Maynooth.

To answer **RQ1** (Section 1.3), we intended to find how novel technologies could empower the CS community in a way such that: **RQ1.1**-multiple campaigns and organisations could make use of the same data collection model, which we addressed by designing and implementing a data-collection framework that can be cloned and extended to fit the campaign specific requirements. **RQ1.2**-The model can be consistently useful across different platforms and user devices, which we answered affirmatively using novel web technologies (e.g. responsive web design and HTML5 APIs) that are supported across platforms and devices without restraining the features available. Modifiability and a clean and simple user

interface are some of they main characteristics of our framework. It reduces the existing inertia that troubles new CS projects who could not previously control and own their data and applications under existing alternatives (Kim et al., 2013; Newman et al., 2011). This was discussed in Section 4.2.

Through a case study, we demonstrated the suitability of our framework in the CS domain. The operation of our campaign, in which volunteers participated from different devices and platforms successfully contributing data to the project, was driven by an application created from cloning and making minor modifications to our proposed model.

Our tool will be of great interest to CS projects which do not have access to significant IT and financial resources to develop their own data collection tools and applications. As the framework will be made available as free open source software we envisage that others will feel capable of adding new features and wish to adapt it to the needs of their projects. Consequently, the community will be able to drive the evolution and maturity of the tool.

6.2 Future work

The set of features included in our model is not definitive. These could be further extended by anyone that wanted to take the project one step further. For example, we provide required field and basic data-type validations and it would make sense to also have type-specific validations available, such as range for numeric inputs or length for alphanumeric text. The current version only allows volunteers to upload one file with each contribution. This could also be extended to support multi-file upload.

During our case study, some volunteers expressed having difficulties identifying the exact location of their observation on the map, specifically when not making their contribution at the time of capture. We think a useful addition would be a map-like component that supports area delimitation with polygons instead of a specific point in the map. This issue also raises data quality concerns on single point location observations which could be mitigated by the proposed component. These proposed extensions are not comprehensive. They are just an example of simple changes or additions that can be done to improve our model. There are other activities that would require more effort but could be equally carried out. We included a UI specifically for volunteers as their engagement and interaction are more likely to be influenced by simple and clear interfaces. However, implementing an ‘organisational’ view that has complete access to the collected data and offers visualization and manipulation options, such as queries, ordering, filters, flagging, chart/diagram generation and download, could be of interest for some organisations. We concentrated on the data collection branch of CS projects during our project. Alternatives to support other models of participation in CS, for example classification, would also be a possibility to extend our work in the field.

Providing implementations of the back-end REST API in different programming languages is another route of future work that can be explored. Our initial version on JavaScript offers consistency and the possibility of using one single language across all layers of the model. But still, having readily available versions written in popular web languages (e.g. C#, Java, Python, Ruby, ...) would increase the options and possibilities available to organisations interested in adopting our model. Ensuring dependability to a greater extent in our framework

is also important as future work. This branch could be followed with activities such as using static analysis tools on the sources as a step of the `build` task, to identify any correctness issues; assessing dependability of the system through metrics and considering other software attributes for evaluation such as modifiability and usability.

As future work a conference or journal paper version of this thesis is planned for later in 2015. The paper will be made available under an open access license to ensure that the entire CS community can access the content and evaluate our work in the context of their own.

References

Abbott, B., Abbott, R., Adhikari, R., Ajith, P., and Allen, B. (2009). Einstein@home search for periodic gravitational waves in ligo s4 data. *Phys. Rev. D*, 79.

Alloy (2012). *Alloy MVC on Github*. https://github.com/appcelerator/alloy.

Angular-Team (2013). *ngdoc*. https://github.com/angular/angular.js/wiki/Writing-AngularJS-Documentation.

Apache (2009). Cordova. https://cordova.apache.org/.

Appcelerator (2009). Titanium. http://www.appcelerator.org/#titanium.

Appcelerator (2014). Titanium Compatibility Matrix. http://docs.appcelerator.com/platform/latest/#!/guide/Titanium_Compatibility_Matrix-section-29004837_TitaniumCompatibilityMatrix-MobileDevelopment.

ARC Centre of Excellence for Environmental Decisions (CEED) (2014). *Let's hear it for citizen scientists!*

Avižienis, A., Laprie, J.-C., and Randell, B. (2004). Dependability and its threats: a taxonomy. In *Building the Information Society*, pages 91–120. Springer.

Avižienis, A., Laprie, J.-C., Randell, B., et al. (2000). *Fundamental concepts of dependability*.

Bonney, R. (1996). Citizen science: A lab tradition. *Living Bird*, 15(4):7 – 15.

Bonney, R., Cooper, C. B., Dickinson, J., Kelling, S., Phillips, T., Rosenberg, K. V., and Shirk, J. (2009). Citizen science: A developing tool for expanding science knowledge and scientific literacy. *BioScience*, 59(11):pp. 977–984.

British Trust for Ornithology (1932). *British Trust for Ornithology*. http://www.bto.org/.

Cavage, M. (2012). *Restify*. https://github.com/mcavage/node-restify.

Chubb, A., O'Brien, K., Bianchin, A., Lavin, P., and Mooney, C. (2010). Fight-Malaria@Home. http://www.fight-malaria.org/. [Online; accessed 30-April-2015].

CitSci.org Mobile Apps (2015). *CitSci.org Mobile Apps*. http://citsci.org/cwis438/websites/CitSci/APPs.php?WebSiteID=7.

Cohen, J. E. (2008). Privacy, visibility, transparency, and exposure. *The University of Chicago Law Review*, pages 181–201.

Conrad, C. and Hilchey, K. (2011). A review of citizen science and community-based environmental monitoring: issues and opportunities. *Environmental Monitoring and Assessment*, 176(1-4):273–291.

Cooper, A., Reimann, R., Cronin, D., and Christopher, N. (2014). *About Face: The Essentials of Interaction Design*. John Wiley & Sons, Inc., New York, NY, USA, 4th edition.

Cooper, S., Khatib, F., Treuille, A., Barbero, J., Lee, J., Beenen, M., Leaver-Fay, A., Baker, D., Popović, Z., et al. (2010). Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760.

Crockford, D. (2006). The application/json media type for javascript object notation (json). RFC 4627, RFC Editor. http://www.rfc-editor.org/rfc/rfc4627.txt.

Dahl, R. (2009). *Node.js*. https://nodejs.org/.

Dickinson, J. L., Zuckerberg, B., and Bonter, D. N. (2010). Citizen science as an ecological research tool: challenges and benefits. *Annual review of ecology, evolution, and systematics*, 41:149–172.

Dirolf, M. (2010). Bson Specification. http://bsonspec.org/spec.html. [Online; accessed 20-April-2015].

Dodds, K. C. (2015). *angular-formly* . https://github.com/formly-js/angular-formly.

Ember.js (2011). *Ember.js*. https://github.com/emberjs/ember.js.

Erickson, T. and Kellogg, W. A. (2000). Social translucence: an approach to designing systems that support social processes. *ACM transactions on computer-human interaction (TOCHI)*, 7(1):59–83.

Evans, C., Abrams, E., Reitsma, R., Roux, K., Salmonsen, L., and Marra, P. P. (2005). The neighborhood nestwatch program: Participant outcomes of a citizen-science ecological research project. *Conservation Biology*, 19(3):589–594.

Gardiner, M. M., Allee, L. L., Brown, P. M., Losey, J. E., Roy, H. E., and Smyth, R. R. (2012). Lessons from lady beetles: accuracy of monitoring data from us and uk citizen-science programs. *Frontiers in Ecology and the Environment*, 10(9):471–476.

Google (2005). *Google Analytics*. www.google.ie/analytics/.

Google (2010). *Angular.js*. https://github.com/angular/angular.js.

Gossman, J. (2005). Introduction to Model/View/ViewModel pattern for building WPF apps. http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx. [Online; accessed 10-May-2015].

Gouveia, C., Fonseca, A., Câmara, A., and Ferreira, F. (2004). Promoting the use of environmental data collected by concerned citizens through information and communication technologies. *Journal of Environmental Management*, 71(2):135–154.

Haklay, M. E., Antoniou, V., Basiouka, S., Soden, R., and Mooney, P. (2014). Crowdsourced Geographic Information Use in Government. Report, Global Facility for Disaster Reduction & Recovery (GFDRR), World Bank, London, UK.

Haviv, A. (2014). MEAN Web Stack. http://meanjs.org/.

Hecht, J. and Rice, E. S. (2014). Citizen science: A new direction in canine behavior research. *Behavioural Processes*.

Hennon, C. C., Knapp, K. R., Schreck, C. J., Stevens, S. E., Kossin, J., Thorne, P., Hennon, P., Kruk, M. C., Rennie, J. J., Gadea, J.-M., Striegl, M., and Carley, I. (2014). Cyclone center: Can citizen scientists improve tropical cyclone intensity records? *Bulletin of The American Meteorological Society - (BAMS)*.

Heroku (2007). *Heroku*. https://www.heroku.com/.

Ionic (2014). *Ionic*. http://ionicframework.com/.

JSDoc (2011). *JSDoc3*. http://usejsdoc.org/.

Karma Test Runner (2012). *Karma Test Runner*. http://karma-runner.github.io/0.12/index.html.

Kim, S., Mankoff, J., and Paulos, E. (2013). Sensr: Evaluating a flexible framework for authoring mobile data-collection tools for citizen science. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 1453–1462, New York, NY, USA. ACM.

Krontiris, I. and Maisonneuve, N. (2011). Participatory sensing: The tension between social translucence and privacy. In Salgarelli, L., Bianchi, G., and Blefari-Melazzi, N., editors, *Trustworthy Internet*, pages 159–170. Springer Milan.

Larson, S. M., Snow, C. D., Shirts, M., and Pande, V. S. (2002). Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*.

Lauriault, T. P. and Mooney, P. (2014). Crowdsourcing: A Geographic Approach to Public Engagement. SSRN Scholarly Paper ID 2518233, Social Science Research Network, Rochester, NY.

Legg, C. J. and Nagy, L. (2006). Why most conservation monitoring is, but need not be, a waste of time. *Journal of environmental management*, 78(2):194–199.

Li, Y. and Manoharan, S. (2013). A performance comparison of sql and nosql databases. In *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, pages 15–19.

Lintott, C., Schawinski, K., Bamford, S., Slosar, A., Land, K., Thomas, D., Edmondson, E., Masters, K., Nichol, R. C., Raddick, M. J., et al. (2011). Galaxy zoo 1: data release of morphological classifications for nearly 900 000 galaxies. *Monthly Notices of the Royal Astronomical Society*, 410(1):166–178.

MacLean, D. (2013). Changing the rules of the game. *eLife*, 2.

Meyer, B. (2006). Dependable software. In *Dependable Systems: Software, Computing, Networks*, pages 1–33. Springer.

Minar, I. (2012). *AngularJS is a MVW framework*. https://plus.google.com/+IgorMinar/posts/DRUAkZmXjNV.

mongolab (2011). *Mongolab*. https://mongolab.com/.

Mooney, P. and Corcoran, P. (2014). Has OpenStreetMap a role in Digital Earth applications? *International Journal of Digital Earth*, 7(7):534–553.

National Audubon Society (1905). *National Audubon Society*. http://www.audubon.org/.

Newman, G., Graham, J., Crall, A., and Laituri, M. (2011). The art and science of multi-scale citizen science support. *Ecological Informatics*, 6(34):217 – 227.

Newman, G., Wiggins, A., Crall, A., Graham, E., Newman, S., and Crowston, K. (2012). The future of citizen science: emerging technologies and shifting paradigms. *Frontiers in Ecology and the Environment*, 10:298–304.

OpenScientist (2011). Finalizing a Definition of 'Citizen Science' and 'Citizen Scientists'. http://www.openscientist.org/2011/09/finalizing-definition-of-citizen.html. [Online; accessed 03-November-2014].

PivotalLabs (2010). Jasmine. http://jasmine.github.io/2.0/introduction.html. [Online; accessed 09-May-2015].

Poetic (2014). https://github.com/poetic/ember-cli-cordova.

Preece, J. and Bowser, A. (2014). What hci can do for citizen science. In *Proceedings of the Extended Abstracts of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI EA '14, pages 1059–1060, New York, NY, USA. ACM.

Preston-Werner, T. (2011). Semantic Versioning. http://semver.org/.

Sadalage, P. J. and Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edition.

Serrano Sanz, F., Holocher-Ertl, T., Kieslinger, B., Sanz García, F., and Silva, C. G. (2014). Citizen science for europe. White Paper.

Shamir, L., Yerby, C., Simpson, R., von Benda-Beckmann, A. M., Tyack, P., Samarra, F., Miller, P., and Wallin, J. (2014). Classification of large acoustic datasets using machine learning and crowdsourcing: Application to whale calls. *The Journal of the Acoustical Society of America*, 135(2):953–962.

Sickler, J., Cherry, T. M., Allee, L., Smyth, R. R., and Losey, J. (2014). Scientific value and educational goals: Balancing priorities and increasing adult engagement in a citizen science project. *Applied Environmental Education & Communication*, 13(2):109–119.

Silvertown, J. (2009). A new dawn for citizen science. *Trends in Ecology & Evolution*, 24(9):467 – 471.

Sommerville, I. (2010). *Software Engineering (9th Edition)*. Pearson.

Stainforth, D., Kettleborough, J., Martin, A., Simpson, A., Gillis, R., Akkas, A., Gault, R., Collins, M., Gavaghan, D., and Allen, M. (2002). Climateprediction.net: Design principles for public resource modeling research. *Computing in Science and Engineering*, 4.

The MongoDB Manual (2015). *The MongoDB Manual*. http://docs.mongodb.org/manual/.

Titanium (2009). *Titanium Mobile on Github*. https://github.com/appcelerator/titanium_mobile.

Tweddle, J., Robinson, L., Pocock, M., and Roy, H. (2012). *Guide to citizen science: developing, implementing and evaluating citizen science to study biodiversity and the environment in the UK*. Natural History Museum and NERC Centre for Ecology & Hydrology for UK-EOF. www.ukeof.org.uk/.

Twitter (2011). *Bootstrap*. https://github.com/twbs/bootstrap.

W3C (2012). *Media Queries*. http://www.w3.org/TR/css3-mediaqueries/.

W3C (2014). *HTML5*. http://www.w3.org/TR/html5/.

Wehn, U., Rusca, M., Evers, J., and Lanfranchi, V. (2015). Participation in flood risk management and the potential of citizen observatories: A governance analysis. *Environmental Science & Policy*, 48(0):225 – 236.

Werbach, K. and Hunter, D. (2012). *For the Win: How Game Thinking Can Revolutionize Your Business*. Wharton Digital Press.

Werthimer, D., Cobb, J., Lebofsky, M., Anderson, D., and Korpela, E. (2001). Seti@home–massively distributed computing for seti. *Computing in Science and Engg.*, 3(1):78–83.

Wikipedia (2015). http://en.wikipedia.org/wiki/LAMP_%28software_bundle%29. [Online; accessed 21-May-2015].

Willett, K. W., Lintott, C. J., Bamford, S. P., Masters, K. L., Simmons, B. D., Casteels, K. R., Edmondson, E. M., Fortson, L. F., Kaviraj, S., Keel, W. C., et al. (2013). Galaxy zoo 2: detailed morphological classifications for 304 122 galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, page stt1458.

Xamarin Platform (2009). *Xamarin Platform*. http://xamarin.com/.