CAPhone: A Collective Awareness Platform for expressing privacy concerns and expectations on Smartphone Applications

Konstantinos Kreatsoulas

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Dimitrios Plexousakis*

This work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

CAPhone: A Collective Awareness Platform for expressing privacy concerns and expectations on Smartphone Applications

Thesis submitted by
Konstantinos Kreatsoulas
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Konstantinos Kreatsoulas

Committee approvals: _____
Dimitrios Plexousakis
Professor, Thesis Supervisor, Committee Member

_____
Evangelos Markatos
Professor, Committee Member

_____
Giorgos Flouris
Principal Researcher, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, February 2019

CAPhone: A Collective Awareness Platform for expressing privacy concerns and expectations on Smartphone Applications

Abstract

In today's highly digitalized modern world, internet users produce colossal amounts of data, with most of them being valuable and personal. Yet, when it comes to controlling who can collect it, what they can do with it, and deciding how best to protect it, users remain in the dark. A socio-technical approach established on collective awareness and informed consent is the objective of the platform presented in this thesis. With this approach, data collection and use by mobile applications are driven by the expectations and needs of the consumers themselves, through a collaborative participatory process. CAPhone will create a new innovative model that will complement existing top-down approaches to data protection, which mainly rely on technical or legal provisions.

This thesis produces a semantic repository of consumer and developer generated content regarding the privacy behaviour of mobile applications. CAPhone will help consumers understand the privacy policies induced in smartphones and their implications via crowd sourced approaches. The ambition behind the platform is to strengthen the trust bond between service developers and users, encouraging innovation and empowering the individuals to promote their privacy expectations as a quantifiable, community-generated request.

CAPhone: Μια πλατφόρμα συλλογικής επίγνωσης για την προστασία προσωπικών δεδομένων στις εφαρμογές έξυπνων κινητών συσκευών

Περίληψη

Στη σημερινή σύγχρονη και ιδιαίτερα ψηφιοποιημένη εποχή, οι χρήστες του Διαδικτύου παράγουν τεράστιο όγκο δεδομένων, το μεγαλύτερο μέρος του οποίου, αποτελείται από πολύτιμα προσωπικά στοιχεία. Ωστόσο, αν τεθεί το ερώτημα, ποιος μπορεί να ελέγξει και να συλλέξει αυτές τις πληροφορίες, τι μπορεί να κάνει με αυτές, καθώς και ποιος είναι ο καλύτερος τρόπος για την προστασία αυτών των δεδομένων, οι περισσότεροι από τους χρήστες δηλώνουν πλήρη άγνοια. Στην παρούσα διατριβή παρουσιάζεται μια κοινωνικό-τεχνική προσέγγιση, η οποία βασίζεται στη συλλογική ευαισθητοποίηση και την εν επιγνώσει συναίνεση των χρηστών. Με βάση αυτή την προσέγγιση, η συλλογή και η χρήση δεδομένων από ψηφιακά προϊόντα, καθοδηγείται από τις προσδοκίες και τις ανάγκες των ίδιων των καταναλωτών, μέσω μιας συνεργατικής συμμετοχικής διαδικασίας, η οποία έχει ως κύριο γνώμονα τη διαμόρφωση των συλλογικών προτύπων προστασίας της ιδιωτικής ζωής. Αυτή η πλατφόρμα θα δημιουργήσει ένα νέο καινοτόμο μοντέλο, το οποίο έχει ως στόχο να συμπληρώσει τις υπάρχουσες προσεγγίσεις για την προστασία των δεδομένων, οι οποίες βασίζονται κυρίως σε τεχνικές ή νομικές διατάξεις.

Αυτή η εργασία θα παρέχει στους χρήστες ένα παγκόσμιο αποθετήριο σημασιολογικού περιεχομένου, το οποίο θα δημιουργείται από τους καταναλωτές και τους προγραμματιστές και θα αφορά τη συμπεριφορά απορρήτου των εφαρμογών κινητής τηλεφωνίας. Η πλατφόρμα θα βοηθήσει τους καταναλωτές να κατανοήσουν τις πολιτικές απορρήτου που θεσπίζονται για τις έξυπνες κινητές συσκευές και τις επιπτώσεις που αυτές έχουν, μέσω προσεγγίσεων που προέρχονται από το πλήθος. Η φιλοδοξία πίσω από τη δημιουργία της πλατφόρμας είναι η ενίσχυση του δεσμού εμπιστοσύνης μεταξύ των παρόχων αυτών των υπηρεσιών και των χρηστών, ενθαρρύνοντας την καινοτομία και ενδυναμώνοντας τα άτομα να γνωστοποιήσουν τις προσδοκίες τους για την προστασία των ιδιωτικών δεδομένων ως ένα ποσοτικοποιήσιμο αίτημα το οποίο απορρέει μέσω μίας κοινότητας.

Acknowledgements

First of all, I would like to thank my supervisor, Professor Dimitris Plexousakis for his continued support during the time of my undergraduate and postgraduate studies and for giving me the chance to work with his team in a very interesting topic of my choice.

Next, I want to express my deepest gratitude to my co-supervisors, Theodore Patkos, post-doctoral researcher at ICS-FORTH, and Giorgos Flouris, principle researcher at ICS-FORTH, for their continuous guidance, cooperation and support. Despite their workload, their office door was always open for advice and directions, throughout the whole process of my work for this thesis.

I would also like to express many thanks to Professor Evangelos Markatos for his willingness to participate as member of my dissertation committee.

My special thanks also go to the Institute of Computer Science (ICS), Foundation of Research and Technology Hellas (FORTH) and more specifically the members of the Information Systems Laboratory (ISL) for providing the necessary support in terms of knowledge and materials. Also, I would like to thank Ioannis Chrysakis and Giannis Roussakis for their useful advice.

This thesis would not be possible without the support from my childhood friends, who despite the distance, were always by my side, encouraging me every step of the way. I greatly value their friendship and I deeply appreciate their belief in me.

During my time in Heraklion, I have been fortunate to enjoy the friendship of a number of special people. Among them, I would like to especially thank my friends Giannis Angelogiannis, Rafail Tsirbas, Mixalis Vassalos, Manos Porfyrakis and Nikos Stefanou for the great moments we had together during all these years.

Last but not least, I would like to deeply thank my parents, Andreas and Katerina, along with my sister Aggeliki, for their love, encouragement and support throughout all these years.

To my parents,
Andreas and Katerina

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The problem

The previous year was dominated with news about privacy and anonymity issues in the digital world. After the exposure of millions of Facebook's user accounts was brought to light, the number of users getting aware of the fact that they may have involuntary supplied their privacy management to businesses and corporations that are primarily driven by a profit motive, is increasing on a daily basis. It is publicly accepted that businesses regularly offer digital products and services, that are conflicting with consumer values and are utilizing personal data in harmful ways.

In spite of the above, the majority of users continue to give their consent to untrustworthy software which can collect, store and process their data without knowing by whom these data will be handled or for what purpose. This lack of awareness leaves consumers vulnerable. If they do not realize that their data is being accessed and shared, they are unlikely to try to look for controls to set their preferences. This also encumbers technical countermeasures from achieving a broad, society-wide impact to consumers privacy protection.

Moreover, company's privacy statements frequently cover multiple services, resulting in endless and vague documents that make it difficult for consumers to have concrete knowledge about the company's information practices. Control of personal information in the digital space, and particularly on mobile devices, presents a unique design challenge. Privacy policies are increasingly posted separately from users' interactions with a system. For instance, websites link to policies at the bottom of pages, mobile apps link to policies in the app store and the privacy policy of smart devices or wearables is likely posted somewhere on the company's website, making it really hard for consumers to keep up with all of that information and probable changes in the content.

The consumers are divided into two separated categories. Those who do not give the appropriate attention to all of those dialogues about terms and conditions that appear, when they launch an application for the first time and those who despite being willing to get informed are going to struggle to understand the major

elements of the content, due to the vast amount of information and very often the requirement of college-level reading skills. In 2008 a study estimated that for the average person to completely read every privacy policy accompanying the websites they visit, the person would need to spend about 250 working hours each year, or about 30 full working days [1].

It is easily foreseen that eventually as the number of privacy violations rises, the contrasting views among what consumers need and what companies offer will substantially hurt the industry. And as in most cases small businesses will be the first to face the consequences of consumers 'ignoring' privacy suspicious products. The urgency to shape strict laws which will monitor business policies for data protection is considered of imperative importance by most stakeholders in the digital market. Europe is already, since 2018, in the process of establishing data protection regulations.

1.2 Our Approach

The core idea behind this thesis is that the society itself can also power data protection. We want to implement a socio-technical solution based on collective awareness and informed consent, whereby data collection is configured by the consumers themselves, through transparent, collaborative and participatory processes, by strengthening the trust bond between digital product developers and users.

The Collective Awareness Platform CAPhone, aims to provide consumers with a platform for smartphone applications, where they can be informed about which personal data they share, and how much privacy friendly an application is. Consumers are able to express their concerns and expectations for the personal information a developer requires access to. By acting as members of a community, users are better informed, and also give somehow feedback to developers to act accordingly.

CAPhone is part of CAPrice - Community [1] initiative, a suite of mechanisms to facilitate community interaction, enabling the explicit declaration of consumers' privacy expectations of the various digital products [5]. The outcome could be a new innovation model that may allow customers to collectively and cooperatively specify their worries, and developers to embrace more privacy-friendly practices and respond to the requirements of customers with novel products and services.

The rest of this thesis is structured as follows. In Chapter 2 we elaborate on some preliminaries required for understanding the rest of this thesis. Then in Section 3 we present related work, and in in Section 4 we describe our methodology for the ontology description, and quantification of users' expectations. Afterwards, Section 5 presents our implementation and some use cases of the platform. Finally Section 6 concludes this thesis and presents directions for future work.

[1] http://www.caprice-community.net

Chapter 2

Preliminaries

The present Chapter provides background material, necessary to follow the main concepts and theories of this Thesis. For the backend part of our platform, we are going to see the main concepts of Semantic Web, Resource Description Framework (RDF), Web Ontology Language (OWL) and Binomial proportion confidence interval. For the frontend part, we will see information about React, Redux and Material UI.

2.1 Collective Awareness Platforms - CAPS

The acronym CAPS stands for Collective Awareness Platforms for Sustainability and Social Innovation. Collective Awareness Platforms (CAPS) are applications based on Internet or mobile communication, scaffolding on social networking for supporting communities by delivering new services, building innovative knowledge, promoting collective intelligence. The final goal of CAPS is to promote more sustainable lifestyles and to induce transformative social innovation [20]. Often such "voluntary model" is conceptualized as a collaborative commons paradigm as it bypasses the capitalist markets and relies on zero marginal cost [18]. The CAPS initiative aims at designing and piloting online platforms to create awareness on sustainability problems and putting in place collective solutions. It fosters collaborative solutions based on networks (of people, of ideas, of sensors), enabling new forms of digital social innovation. CAPS projects are expected to support environmental awareness, grassroots processes and practices enabling citizens to:

- share knowledge,
- make better informed decisions as consumers,
- nudge collective environmentally-savvy behavioural changes, and
- set up more participatory democratic processes.

2.2 Semantic Web

The word semantic itself implies meaning or understanding. As such, the fundamental difference between Semantic Web technologies and other technologies related to data (such as relational databases or the World Wide Web itself) is that the Semantic Web is concerned with the meaning and not the structure of data. This fundamental difference engenders a completely different outlook on how storing, querying, and displaying information might be approached. Some applications, such as those that refer to a large amount of data from many different sources, benefit enormously from this feature. Others, such as the storage of high volumes of highly structured transactional data, do not.

The Semantic Web includes the common formats for integration and combination of data drawn from diverse sources, whereas on the original Web they are mainly concentrated on the interchange of documents. It is also about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing [24].

It is an extension of the current Web, in which information is given well defined meaning, better enabling computers and people to work in cooperation. It is driven by the World Wide Web Consortium (W3C) and builds on W3C's Resource Description Framework (RDF), and is usually designed with syntaxes that use Uniform Resource Identifiers (URIs) to represent data. These syntaxes are known as RDF syntaxes. The inclusion of data to RDF files enables computer programs or Web spiders to search, discover, collect, assess and process the data on the Web.

Semantic Web consists primarily of three technical standards [19] :

- **RDF (Resource Description Framework)**: The data modeling language for the Semantic Web. All Semantic Web information is stored and represented in the RDF.

- **SPARQL (SPARQL Protocol and RDF Query Language)**: The query language of the Semantic Web. It is specifically designed to query data across various systems.

- **OWL (Web Ontology Language)**: The schema language, or knowledge representation (KR) language, of the Semantic Web. OWL enables you to define concepts composably so that these concepts can be reused as much and as often as possible. Composability means that each concept is carefully defined so that it can be selected and assembled in various combinations with other concepts as needed for many different applications and purposes.

The key goal of the Semantic Web is to trigger the evolution of the existing Web to enable users to search, discover, share and join information with less effort. Humans can use the Web to execute multiple tasks, such as booking online

tickets, searching for different information, using online dictionaries, etc. Even so, machines are not able to carry out any of these tasks without human intervention, because Web pages are made to be read by humans, not machines. The Semantic Web can be considered a vision for the future in which data could be quickly interpreted by machines, allowing them to carry out numerous tedious tasks related to discovering, blending, and taking action on the information available on the Web [23].

2.2.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is an infrastructure that enables the encoding, exchange and reuse of structured meta-data. RDF is an application of XML that imposes needed structural constraints to provide unambiguous methods of expressing semantics. RDF additionally provides a means of publishing both human readable and machine processable vocabularies designed to encourage the reuse and extension of meta-data semantics among disparate information communities. The structural constraints RDF imposes to support the consistent encoding and exchange of standardized meta-data provides for the interchangeability of separate packages of meta-data defined by different resource description communities [13].

The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines the semantics of any application domain. The definition of the mechanism should be domain neutral, yet the mechanism should be suitable for describing information about any domain.

The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph. This can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link (hence the term "graph"). This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations. The subject indicates the resource that is described by the statement. The predicate describes the relationship between the subject and the object. Finally, the object can be either uniquely identifiable by a Uniform Resource Identifier (URI) or a literal (e.g text). There are many different formats that RDF statements can be represented in, but in this master thesis is given on RDF/XML format.

2.2.2 SPARQL

SPARQL [3] is the official W3C recommendation for querying RDF graphs, and is based on the concept of matching patterns against the RDF graph. Thus, a SPARQL query determines the pattern to seek for, and the answer is the part(s) of the RDF graph that match this pattern.

More specifically, SPARQL defines triple patterns which resemble an RDF

triple, but may have a variable (prefixed with character ?) in any of the subject, predicate, or object positions in the RDF triple. Intuitively, triple patterns denote the triples in an RDF graph that have the form specified by the triple patterns. SPARQL graph patterns are produced by combining triple patterns through the join, optional and union SPARQL operators. Graph patterns may contain filters, using the FILTER expression that specify conditions on the triple patterns.

2.2.3 OWL Web Ontology Language

Web Ontology Language (OWL) is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology.

An ontology is a formal specification of a conceptualisation, that is, an abstract and simplified view of the world that we wish to represent, described in a language that is equipped with a formal semantics. In knowledge representation, an ontology is a description of the concepts and relationships in an application domain. Depending on the users of this ontology, such a description must be understandable by humans and/or by software agents. In many other fields (such as in information systems and databases, and in software engineering) an ontology would be called a conceptual schema. An ontology is formal, since its understanding should be non ambiguous, both from the syntactic and the semantic point of views [6].

The W3C recommendation presents three versions of OWL, depending on the degree of expressive power required [25].

- **OWL Lite** supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL, see the section on OWL Lite in the OWL Reference for further details.

- **OWL DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.

- **OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full, a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

OWL's core idea is to enable efficient representation of ontologies that are also amenable to decision procedures. It checks an ontology to see whether it's logically consistent or to determine whether a particular concept falls within the ontology. OWL uses the linking provided by RDF to allow ontologies to be distributed across systems. Ontologies can become distributed, as OWL allows ontologies to refer to terms in other ontologies. In this way OWL is specifically engineered for the Web and Semantic Web. OWL is seeing increased adoption but still needs tools and software development environments to support its production and application. These are starting to appear but as yet we have few means to routinely and effortlessly generate Semantic Web annotations using this or other languages at the point of content use or creation [15].

OWL has been designed to meet this need for a Web Ontology Language and takes part in the growing stack of W3C recommendations related to the Semantic Web [25].

- **XML** provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.

- **XML Schema** is a language for restricting the structure of XML documents and also extends XML with datatypes.

- **RDF** is a datamodel for objects ("resources") and relations between them, provides a simple semantics for this datamodel, and these datamodels can be represented in an XML syntax.

- **RDF Schema** is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.

- **OWL** adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

2.2.4 Ontologies as a specification mechanism

A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are presumed to exist in some area of

interest and the relationships that hold them (Genesereth & Nilsson, 1987). A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.

An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an ontology is a systematic account of Existence. Ontologies are also like conceptual schemata in database systems. A conceptual schema provides a logical description of shared data, allowing application programs and databases to interoperate without having to share data structures [7].

2.3 React and Redux

React is an open-source declarative, efficient, and flexible JavaScript library, which is used for building user interfaces specifically for single page applications. It was created by Jordan Walke, a software engineer at Facebook, who was influenced by XHP, an HTML component framework for PHP. At first it was deployed on Facebook's news feed in 2011 and later on Instagram.com in 2012. [9] and afterwards it became open-sourced at JSConf US in May 2013. Now it is maintained by Facebook and a community of individual developers and companies.

React is used for handling view layer for web and mobile apps and allows us to create reusable UI Components. Developers are able to create large web applications which can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple.

The reason why it is characterized declarative is because React makes it painless to create interactive UIs. It designs simple views for each state in the application, and React will efficiently update and render just the right components when the data changes. Declarative views make the code more predictable, simpler to understand, and easier to debug.

A developer can build encapsulated components that manage their own state and then compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, developers can easily pass rich data through their app and keep state out of the DOM.

Aside from React, Redux state container was used to maintain global application store. Redux is a predictable state container. This means that the cause of state change, can easily be identified. Redux uses a single store instead of many different stores like other flux inspired libraries do. This is often called "single source of truth", as the store is the only place where state can be accessed from (Abramov, 2016).

As applications become more complex, they have to manage more state. Redux makes state changes predictable. It achieves this by setting several rules. The first of which being that state must be immutable. Secondly the information about state changes must be contained by actions. And thirdly the use of reducers, which are pure functions, specifying how actions transform the state. References to multiple states can be stored, as the state in a Redux application is immutable.

2.4 Material UI

Material UI is a design language that was first introduced by Google in 2014. It is a visual language that makes use of grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows. The goal for Material Design is down to three things: **Create**, **Unify** and **Customize**.

With **Create**, Material UI aims to provide a visual language that synthesizes the classic principles of good design. With **Unify**, it aims to develop a single underlying system that unifies the user experience across platforms, devices, and input methods and with **Customize**, it provides a visual language and a flexible foundation for innovation and brand expression. It is inspired by the physical world and its textures, including how they reflect light and cast shadows. Material surfaces re imagine the mediums of paper and ink.

2.5 Binomial proportion confidence interval

In statistics, a binomial proportion confidence interval is a confidence interval for the probability of success calculated from the outcome of a series of success - failure experiments (Bernoulli trials). In other words, a binomial proportion confidence interval is an interval estimate of a success probability p when only the number of experiments n and the number of successes n_s are known.

There are several formulas for a binomial confidence interval, but all of them rely on the assumption of a binomial distribution. In general, a binomial distribution applies when an experiment is repeated a fixed number of times, each trial of the experiment has two possible outcomes (success and failure), the probability of success is the same for each trial, and the trials are statistically independent. Because of the fact that the binomial distribution is a discrete probability distribution (i.e., not continuous) and difficult to calculate for large numbers of trials, a variety of approximations are used to calculate this confidence interval, all with their own trade offs in accuracy and computational intensity [21].

A commonly used formula for a binomial confidence interval relies on approximating the distribution of error about a binomially - distributed observation $\hat{p}$, with a normal distribution [26]. This approximation is based on the central limit theorem and is unreliable when the sample size is small or the success probability is close to 0 or 1 [11].

Using the normal approximation, the success probability p is estimated as

$$\hat{p} \pm z\sqrt{\frac{\hat{p}\,(1-\hat{p})}{n}},$$

where $\hat{p} = n_s/n$ is the proportion of successes in a Bernoulli trial process, measured with n trials yielding n_s successes and n_F = n - n_s failures, and z is the $1 - \frac{\alpha}{2}$ quantile of a standard normal distribution corresponding to the target error rate

α. For a 95% confidence level, the error $\alpha = 1 - 0.95 = 0.05$, so $1 - \frac{\alpha}{2} = 0.975$ and $z = 1.96$.

An important theoretical derivation of this confidence interval involves the inversion of a hypothesis test. Under this formulation, the confidence interval represents those values of the population parameter that would have large p-values if they were tested as a hypothesized population proportion. The collection of values, θ, for which the normal approximation is valid can be represented as

$$\left\{ \theta \;\middle|\; y \leq \frac{\hat{p} - \theta}{\sqrt{\frac{1}{n}\hat{p}\,(1-\hat{p})}} \leq z_{\frac{\alpha}{2}} \right\},$$

where y is the $\frac{\alpha}{2}$ quantile of a standard normal distribution. Since the test in the middle of the inequality is a Wald test, the normal approximation interval is sometimes called the Wald interval, but it was first described by Pierre-Simon Laplace in 1812 [12].

2.6 Wilson Score Interval

The Wilson score interval is an improvement over the normal approximation interval in that the actual coverage probability is closer to the nominal value. It was developed by Edwin Bidwell Wilson (1927) [27].

Using the Wilson score interval, the success probability p is estimated as

$$\frac{\hat{p} + \frac{z^2}{2n}}{1 + \frac{z^2}{n}} \pm \frac{z}{1 + \frac{z^2}{n}} \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}$$

or the equivalent

$$\frac{n_S + \frac{z^2}{2}}{n + z^2} \pm \frac{z}{n + z^2} \sqrt{\frac{n_S n_F}{n} + \frac{z^2}{4}}.$$

Intuitively, the center value of this interval is the weighted average of $\hat{p}$ and $\frac{1}{2}$, with $\hat{p}$ receiving greater weight as the sample size increases. This interval has good properties even for a small number of trials and/or an extreme probability. The Wilson interval can be derived from Pearson's chi-squared test with two categories. The resulting interval,

$$\left\{ \theta \;\middle|\; y \leq \frac{\hat{p} - \theta}{\sqrt{\frac{1}{n}\theta(1-\theta)}} \leq z \right\},$$

can then be solved for θ to produce the Wilson score interval. The test in the middle of the inequality is a score test.

The Wilson interval may be modified by employing a continuity correction, in order to align the minimum coverage probability, rather than the average probability, with the nominal value. Just as the Wilson interval mirrors Pearson's chi-squared test, the Wilson interval with continuity correction mirrors the equivalent Yates' chi-squared test.

The following formulae for the lower and upper bounds of the Wilson score interval with continuity correction (w^-, w^+) are derived from Newcombe (1998) [14].

$$w^- = \max\left\{0, \frac{2n\hat{p} + z^2 - \left[z\sqrt{z^2 - \frac{1}{n} + 4n\hat{p}(1-\hat{p}) + (4\hat{p}-2)} + 1\right]}{2(n+z^2)}\right\}$$

$$w^+ = \min\left\{1, \frac{2n\hat{p} + z^2 + \left[z\sqrt{z^2 - \frac{1}{n} + 4n\hat{p}(1-\hat{p}) - (4\hat{p}-2)} + 1\right]}{2(n+z^2)}\right\}$$

However, if $p = 0$, w^- must be taken as 0 and if $p = 1$, w^+ is then 1.

Chapter 3

Related Work

The new dynamics brought about by digitization has raised many privacy concerns, attracting the interest of various stakeholders around the world that study the implications of this development and generate solutions toward privacy protection. Privacy related activities are being conducted in the context of numerous EU-funded projects, but are also being supported by initiatives of industrial consortia, as well as by groups of individuals in research institutes and organizations.

Since there is no other platform that offers the functionality that we want to provide in CAPhone platform, we would like to benefit from the progress other related projects have in the following two basic dimensions **Privacy**, and **Collective Intelligence**, as well as the **Community Building** dimension that CAPrice - Community aims to endorse. In the next paragraphs, some of those related projects will be cited, along with their prospect.

A privacy related work is **SafeCloud** [1], that re-architects cloud infrastructures to ensure that data transmission, storage, and processing can be partitioned in multiple administrative domains that are unlikely to collude, so that sensitive data can be protected by design and also entangled with inter-dependencies that make it impossible for any of the domains to tamper with its integrity. Like that, users will control the choice of non-colluding domains for partitioning and the trade-offs between entanglement and performance, and thus will have full control over what happens to their data. This will make users less reluctant to manage their personal data online due to privacy concerns and will generate important benefits for privacy-sensitive online applications such as distributed cloud infrastructures and medical record storage platforms.

CAPhone platform has as its main motive the power of privacy, as consumers should be assured that their personal sensitive information will not be exploited by any company for profit and deception. Thus, we embrace projects like SafeCloud that have, as their main purpose, to protect the human rights and their sensitive data. Bearing this in mind, the platform designed gives the consumers the opportunity to express their concerns and expectations for a specific application and

[1] http://www.safecloud-project.eu

also to denote evidence if they suspect they are being deceived.

Of particular interest are projects that lie in the intersection of the Privacy and Community Building dimensions, which focus on raising awareness on issues related to privacy and trustworthiness. A project like that is **saferInternet4Kids** [2], an effort led by Foundation for Research and Technology and more particularly the Institute of Informatics, in which someone can learn about safe use of the Internet and social networks.

Moreover, there is a number of projects that exist in the intersection of the Privacy and the Collective Intelligence dimensions. **WikiRate** [3], has strived to foster collective awareness on corporate social responsibility, made possible through the open, editable, wiki platform that connects a global community. The collaborative platform enables academics, non-profits, standard bodies, investors, companies, and the general public to research, discuss and rate company performance. As a collaborative site, WikiRate fosters and empowers communities of individuals and organizations to push for greater transparency through open, quality research that enhances the WikiRate platform as a tool for advocacy, education, research and analysis.

Another, crowdsourced attempt for privacy is **Privacy Flag** [4] project, which combines crowdsourcing, ICT technologies, and legal expertise to protect citizen privacy when visiting websites. Its purpose is to enable citizens to monitor and control their privacy with user friendly solutions made available as smartphone applications, web browser add-ons, all connected to a shared knowledge database. Privacy Flag is co-financed by the European Commission and the Swiss State Secretariat for Education, Research, and Innovation.

Apart from the aforementioned EU-funded projects, our focus is also placed on relevant initiatives by industrial, research or other organizations. **The Digital Standard** [5] is an ambitious, open, and collaborative effort, to create a digital privacy and security standard to help guide the future design of consumer software, digital platforms and services, and Internet-connected products. The perspective of that is to enable consumer organizations to test, evaluate, and report on whether new products protect consumer security and privacy, and to empower consumers to make smarter choices about the products they buy.

Similarly, the study of consumers' concerns and expectations on digital privacy is attracting the interest of other fields. From the psychological perspective, a recent study considered for the first time the notion of collective privacy concerns on social media, suggesting a conceptual model to measure privacy concerns [8]. In [16], the authors introduced a formal model of privacy preferences on the Social Web, revealing for instance that the high variance in users' preferences can be grouped into profiles [10], and developing the personalized **Privacy Assistant** [6]

[2] https://saferinternet4kids.gr
[3] http://wikirate.eu
[4] https://privacyflag.eu
[5] https://www.thedigitalstandard.org
[6] http://www.privacyassistant.org

for learning the privacy preferences of users over time. These efforts are relevant for achieving our objectives, offering conceptual models for the CAPrice Semantic Privacy Wiki or providing further insights on how our notion of collective privacy norms can facilitate the interaction of consumers with digital technology.

From all the above projects, one can easily understand that the importance of data protection and transparency is more imperative than ever. Consumers must be aware of the data they provide companies with, as well as, where applications require access. CAPhone was created for this specific purpose, to help consumers realize which applications are trustworthy, through a collective awareness process. The developers also can take advantage of this motion, as they can understand more effortlessly what expectations consumers have and what they need to change. Such projects and platforms should be reinforced, so that consumers feel more confident and secure about the applications they use.

Chapter 4

Methodology

In this Chapter focus, is given on the system that was implemented. Firstly, the overall idea of this collective awareness platform will be described in detail, for expressing privacy concerns and expectations on smartphone applications, its features and the main goal we try to achieve through it. Secondly, we will analyze the ontology schema, its entities and their connections. Finally, in order to quantify the privacy friendliness of an application and the honesty of a developer, functions were defined to correlate the relations between the developers' requirements and users' expectations.

4.1 The approach of CAPhone

The Collective Awareness Platform CAPhone is part of the CAPrice - Community [1] initiative and aims to implement a socio-technical solution based on collective awareness, where consumers will have a general view of which personal data they share and how privacy friendly each android application is. Developers, in order to submit their applications to the platform, are asked to specify in a more comprehensive form the permission groups that they want access to and to provide a justification for them. In this way, consumers will be better informed about the personal data they share, where they give access to and for what purpose.

In addition, consumers are able to express their concerns and expectations for each permission group. Taking into account users' opinions and expectations we defined the scoring functions that will be described in detail in section 4.3 and as a result we evaluate how privacy friendly an application is based on the outcome of **CAPhone Community Score**.

Moreover, if consumers find out that a developer's requests are not in accordance with his submitted justifications or that he uses them for different purposes, they can take actions to inform the community. Specifically, they can provide a link that explains with clear evidence how the application violates its own claims.

[1] http://www.caprice-community.net

As this is a critical aspect both for the developer and for the community, crowdsourcing mechanisms are deployed to rate the credibility of the evidence. Only highly accurate application violations affect the **Honesty Score** of a developer.

The goal through these collaborative and participatory processes, is to make users more aware about the privacy policies and the data they share. Developers can use the concerns and expectations of the users as feedback and act accordingly. It is essential for us to make consumers and developers feel as members of a community, where they can interact with each other for the benefit of all.

4.2 Ontology Design Description

In this section, we are describing the basic concepts of the platform. These are modeled as an ontology, which captures their meaning and relations in detail, including all hierarchies (classes, subclasses and properties) that have arisen. Our approach, represents an ontology that captures all the aspects of a collective awareness platform.

As previously mentioned (Chapter 1), our collective awareness platform will give users the ability to express their opinion about the permission groups that each application wants to have access to. In general, when an application is published to a web store like Google Play Store and App Store, the developer has to justify where does the application need access and for what reason. Afterwards, the users can express their expectations for each privacy policy and if they find out that an application uses their personal data for other purposes, they can submit an evidence providing the source.

In the following subsections, we will examine more closely the entities we defined and the connections between them. Classes are demonstrated as ellipses, their subclasses with a solid green arrow and the relations between them with a blue arrow and the predicate on top of it. Each time we are describing an entity, we fill the corresponding ellipsis with grey color. The entity Resources refers to alphabetical strings and numbers.

4.2.1 Person

There are two different types of end users inside the platform. On the one hand, we have the Developers and, on the other hand, the basic Users. Both of them share some common characteristics, with the addition of some extra. For that reason, we define an entity Person for the common attributes and, below it, the Developer and User entities are defined.

For a Person (Figure 4.1), we need some basic personal information, like first and last name, email and age. Also, we need a password, in order for the user to sign in to the platform. All these predicates refer to literals, thus they are connected with the entity Resources. The additional information that we need for the Developer, help the purpose of being compatible with the details that exist already in Google Play Store. These are the address, website and a link to the privacy policies.

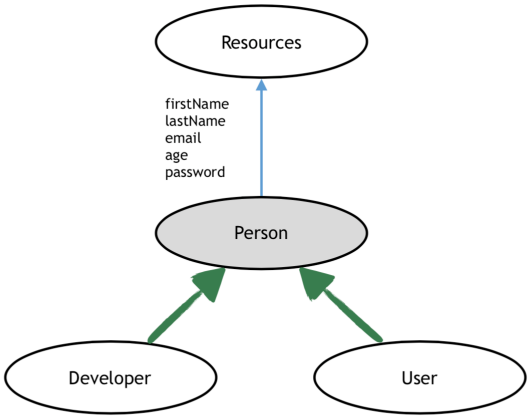

Figure 4.1: The Person Entity

The Developer entity (Figure 4.2), is connected with the Application entity with predicate "developedBy", for the applications that the Developer has developed. Also, it has the predicates "website", "policy" and "address", which refer to literals, thus they are connected with the entity Resources.

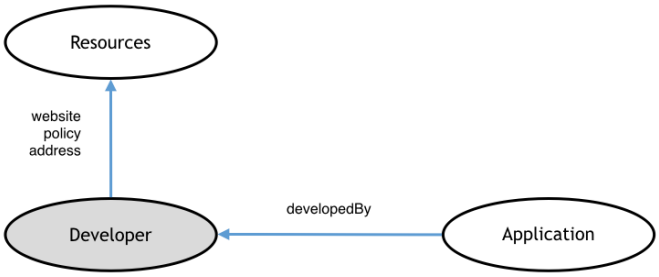

Figure 4.2: The Developer Entity

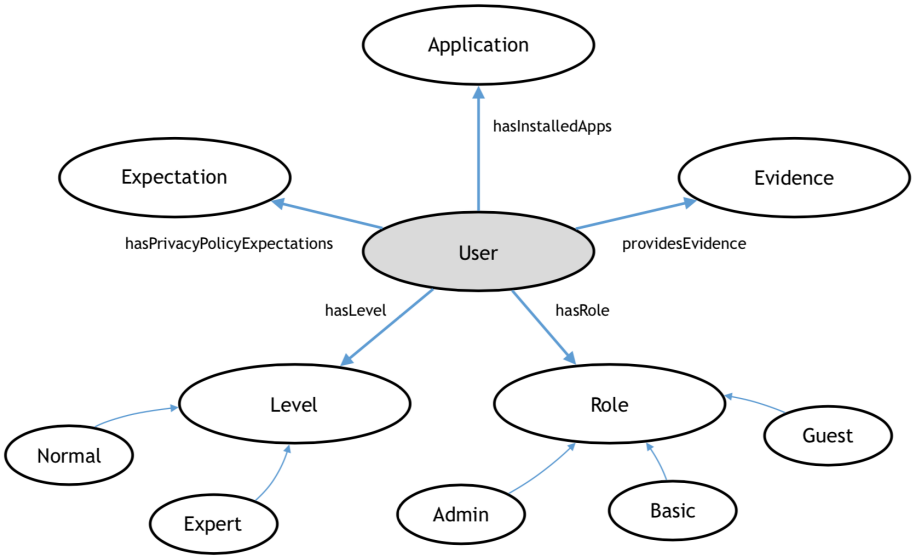

Figure 4.3: The User Entity

The User entity (Figure 4.3), is connected with the Application entity with predicate "hasInstalledApps", for the applications that the User has installed. More than that, a User is connected with the Evidence and Expectation entities, that we are going to describe in subsection 4.2.6, with the predicates "providesEvidence" and "hasPrivacyPolicyExpectations", respectively.

The users of the platform are separated in three different categories, based on their role. We define an **Admin**, who is going to be responsible to maintain the platform and the proper management of the system. He will be eligible to remove a user that does not operate conscientiously or tries to confuse the overall view of the community. We also define the **Basic** users, who can see the applications that exist, review possible evidence that other users have found for a specific privacy policy and express their opinion. The last role is the **Guest**, who can only take a look at the available applications in the platform and check how privacy friendly they are.

For the time being, we have implemented the Developer and the Basic user. In Chapter 5, we are going to see the differences in the interface between those two and how they interact with the system. Also, in Chapter 6, we will describe some additions that we want our platform to provide, in order to motivate and reward participation and loyalty of the members of the community. Thus, we add in the Schema of our ontology the Level entity, which separates users in **normal** and **expert**, based on their interaction history with the system.

4.2.2 Digital Products

Over the last decade, technology has progressed in every field imaginable. Companies release new smart devices, in order to satisfy user desires. Some of the most commonly used smart devices are smartphones, tablets, Smart TVs and other wearable gadgets. Developers release applications for all these Digital Products. Though, very often the same application between different Digital Products requires access to different permission groups i.e. Camera, Microphone, etc. For instance, a web application may not require access to your GPS, as opposed to smartphone applications [17]. In order to handle this occasion, we give the flexibility to the administrator of our collective awareness platform to create different sections of Digital Products as Platforms.

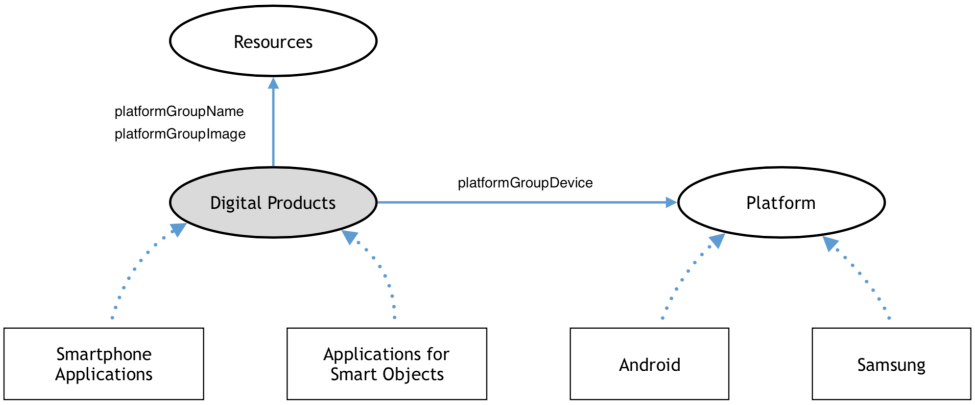

Figure 4.4: The Digital Products Entity

As illustrated in Figure 4.4, there are two instances of Platform, Smartphone Applications and Applications for Smart Objects. This thesis will focus on Smartphone Applications and more specifically on Android applications, which are an instance of Smartphone Applications. All information that we store in the platform is described in the ontology. Thus, Digital Products have as predicates "platformGroupName" and "platformGroupImage", which are connected with the entity Resources. Both of them are literals. Moreover, there is another predicate, the "platformGroupDevice", which connects Digital Products with Platform.

4.2.3 Platform

The entity Platform (Figure 4.5) has as predicates, "platformName", "platformGroupDegitalDevices" and "platformImage", which are literals and give us information about the platform itself. For example, the Android section inside the smartphone applications. Finally, we have the predicate "platformApp", to connect platforms with the Entity Application.

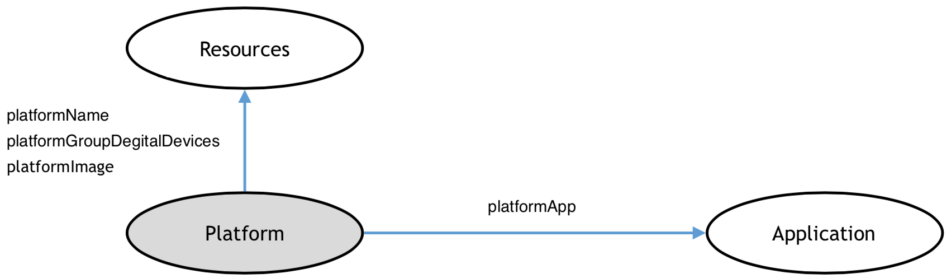

Figure 4.5: The Platform Entity

4.2.4 Application

The general idea of this platform is all the available applications of different digital platforms to be stored in one place, so the community will have the opportunity to express its opinion and specify if it is safe to use it or not.

As we will see in Chapter 5, we inserted around 250.000 applications from the Google Play Store in our platform. Note though that Android applications, as they have been published, do not provide all the appropriate information that we need. In the official website of Google Play Store [2], we can see a list with all applications. When we choose one of them we can see more details about it. At the bottom of the details page, there is a section named "Additional Information". From there, the user can find more details about the developer, the number of installations, but more importantly about the permissions the application wants to have access to.

For demonstration purposes, we present as an example the Android version of Skype as it appears in Google Play Store. In Figure 4.6, we can see the additional information that the developer provides and also the permissions that he wants access to. When we select View Details we get the following list as presented in Figure 4.7. Permissions will be further discussed in subsection 4.2.5.

[2] https://play.google.com/store/apps/

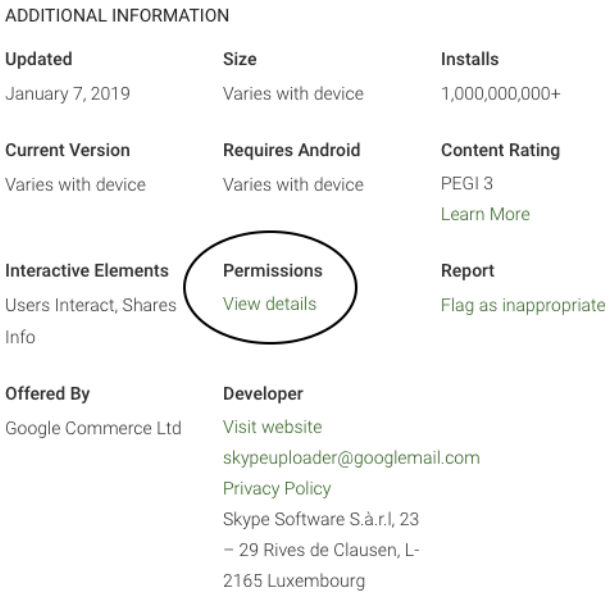

Figure 4.6: Additional information of Skype

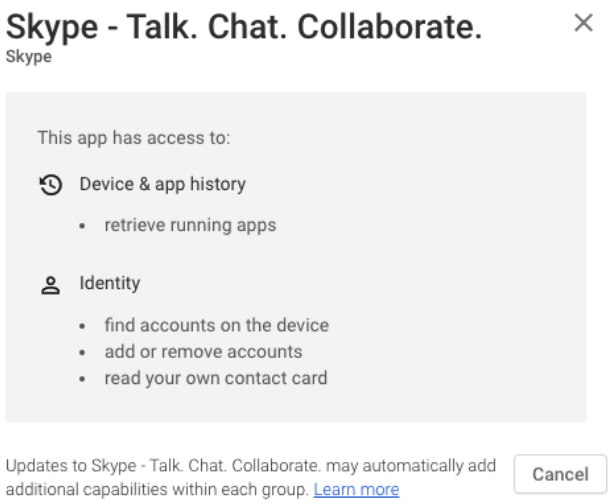

Figure 4.7: Sample permissions of Skype

The design of our ontology is based on the information we obtain from the Google Play Store, but we also add some fields that we need for the community review process, like the exact justification of a permission (Subsection 4.2.6.1), expectation of users and maybe the evidence they have found (Subsection 4.2.6).

Figure 4.8 demonstrates the connections between Application, Developer, Permission and User. Firstly, as already mentioned, for each application we keep information retrieved from the Google Play Store. This information is represented as predicates, which are connected with the Resources. Secondly, an Application is connected with a Developer with the predicate "developedBy" and with Permission through the predicate "hasPermission". Finally, an Application is linked to a User with the predicate "hasInstalled".

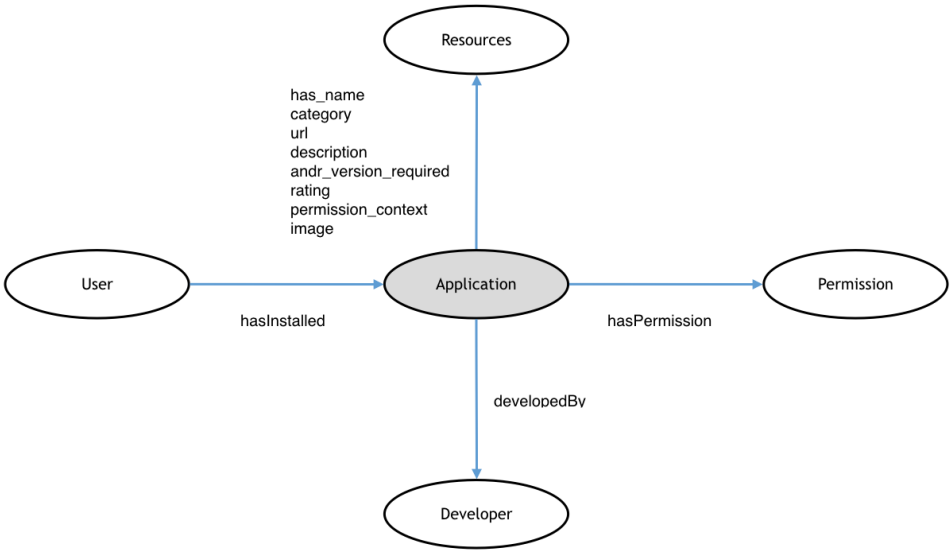

Figure 4.8: The Application Entity

4.2.5 Permission

Developers ask permission from users, in order for their applications to work properly and to offer a better experience to them. A lot of applications provide different interactions with the users and that is the reason why they want access in various categories. Some of them are mandatory, while others are not.

In general, all permissions in Android have the following structure. There are Permissions Groups and Permissions. Figure 4.7 illustrates a sample of where Skype wants access to. For example, "Device & app history" is a permission group and "retrieve running apps" is the permission. This is also the structure that we use in our ontology. More specifically, we capture all permission groups and their permissions that exist in Google Play Store (until January 2019).

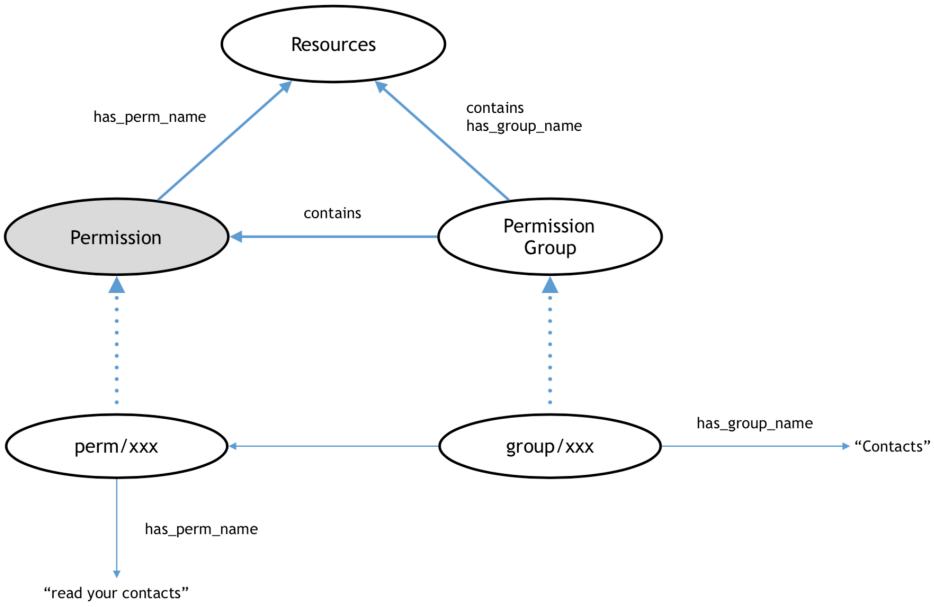

Figure 4.9: The Permission Entity

We can see the structure of a permission group in Figure 4.9. At this point, we have to clarify that the main reason we define as our main entity the Permission instead of Permission Group is that we want to be compatible with the Google Play Store. There are sixteen permissions groups (January 2019). Developers may add new or change the permissions of them, but the groups remain the same or at least they will not be modified as regularly. For simplicity, we use those permission groups as our permissions in our platform.

4.2.6 Privacy Policy

As the most important entity of our platform we define Privacy Policy. This is the main class that the other entities depend on. A Privacy Policy is composed of a Permission Group and a Justification. When a developer wants to submit his application to the platform, he has to specify from a given list which permission groups the application wants access to and to justify his line of argumentation.

4.2.6.1 Justification

There are four types of Justifications that the developer can choose from, in order to specify the reason why he wants access to a specific Permission Group. Those are the following: **Purpose**, **Period**, **Storage Method** and **Shared With**. In Figure 4.10 we can see the different possible values of each type. The developer can inform the community for the purpose of each permission group that he wants access to, the duration that he wants it, how he is going to store the information that he will get and, finally, with whom he is going to share this information. The Permission Groups and their Justifications, constitute the Privacy Policies.

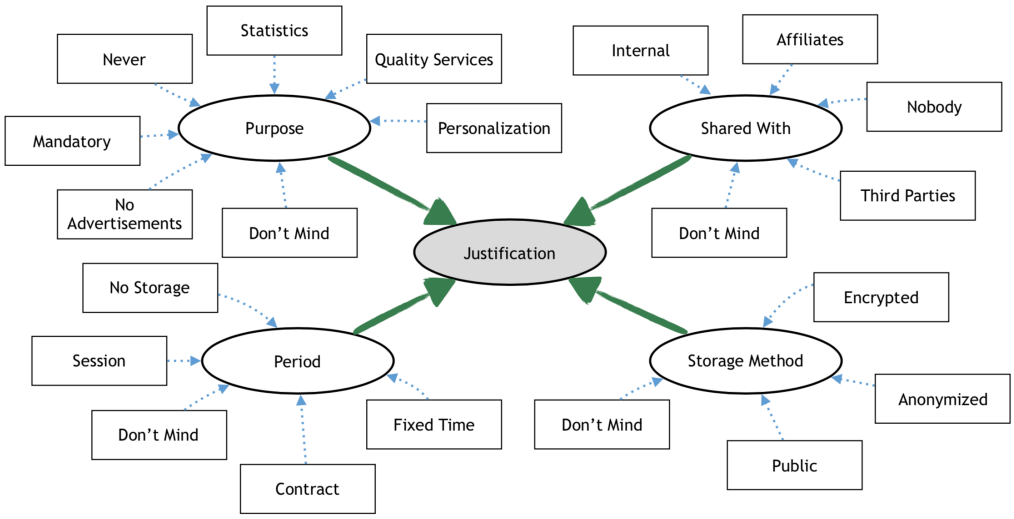

Figure 4.10: The Justification Entity

4.2.6.2 Expectation

The main purpose of our platform is to enable users to express their opinion about the privacy policies developers have presented. Sometimes the submitted privacy policies are not accurate or the users find usage of their personal information that they consider unreasonable. Thus, we give them the opportunity to express their expectation for each privacy policy, by permitting or refusing it. Additionally users can also submit the appropriate justifications for each permission group that we will make them give access to it. There are three types of Expectations the consumers can choose, in order to specify their preference. Those types are the following: **Permitting**, **Refusing** and **OnlyIf** (Figure 4.11).

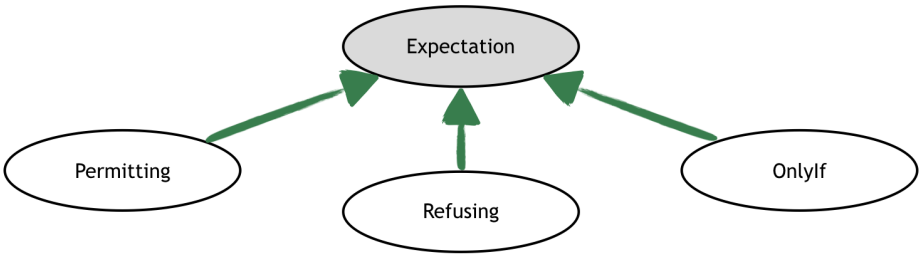

Figure 4.11: The Expectation Entity

4.2.6.3 Evidence

Moreover, in a lot of cases users find some evidence that substantiate the mistreatment of their personal data. Then, they can submit the evidence that they have found in our platform, so the other members of the community will get informed and vote for the validation of it. Consumers can confirm or report an evidence. Thus, Evidence has two types, as we can see in Figure 4.12, the **Confirm** and **Report**. The most accurate evidence is presented at top of the list. Based on the expectations and feedback by users, we calculate a privacy friendliness score for the application that we are going to describe in the following Chapter 4.3.

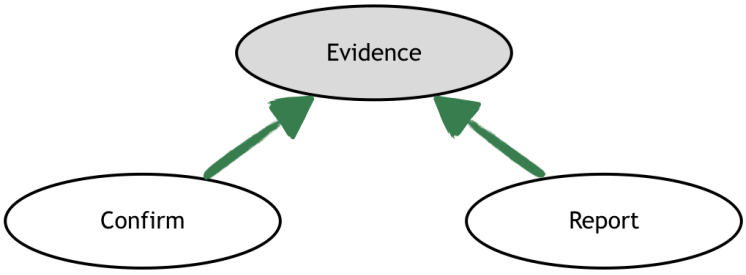

Figure 4.12: The Evidence Entity

According to the above, in Figure 4.13 we see the connections between Privacy Policy and the other entities. It is connected with Application via "hasApplication", with Permission Group via "hasPermissionGroup" and with Justification via "hasJustification". A User is connected with Expectation via the predicate "hasPrivacyPolicyExpectations" and with Evidence via "providesEvidence".

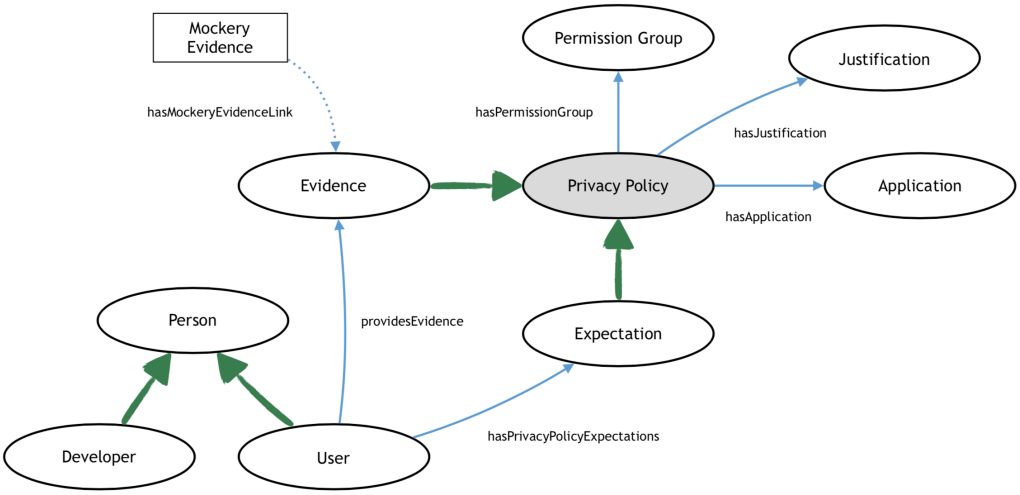

Figure 4.13: The Privacy Policy Entity

Since all the entities of our ontology have been defined, we can take a look at an example based on the Skype application. As we can see in Figure 4.7, the application requires access to *"Device & app history"* and *"Identity"*. In each one of them, the developer gives a sort description in a bullet list form, nevertheless without clarifying exactly what will be used and for what purpose. For our ontology, Device & app history and Identity correspond to Permission Groups, and the information in bullets the Permissions. In CAPhone only the Permission Groups are used to specify where an application tries to gain access to. Moreover, we denoted four Justification categories (Purpose, Period, Storage Method and Shared With) that developers who want to submit their applications to our platform should fill in. A Privacy Policy is a triple of entities Digital Product, Permission Group and Justifications, i.e. PrivacyPolicy(Skype, Identity, QoS). When the consumers express their opinion by permitting or refusing a privacy policy, they actually define a triple Expectation(Skype, Identity, QoS). If they would permit a privacy policy under conditions, then they will express a new Expectation with the justification they agree the most with i.e. Expectation(Skype, Identity, Analytics).

4.3 Quantifying Users' Expectations

The objective we seek to accomplish with CAPhone is for users to be informed about the personal data that they share, while they are using a specific application and also to have a general idea of how privacy friendly this application is. While the community interacts with the system, outcomes an intention of how safe is to use an application and how honest are the developers with the customers. In order to evaluate how privacy friendly an application is, we implement score functions that are described in the following sections. These functions measure community's trends; users express their concerns and expectations about the permission groups the developer wants access to and, as a result, they generate general perspective about the application's trustworthiness.

Our computations perspective is as follows. For each privacy policy of a specific application, users can express their expectation for a specific justification of a permission group, that the developer has defined. They can permit or refuse access to permission groups and also, there is an option where they can denote under what conditions they will give access and point out their preferred justifications. Based on that perspective, we formally define our domain ingredients (objects, concepts etc). Next, we define our domain behavior (functions, relations, etc.) and finally we define our targeted behavior, which is the quantification of users' expectations through a score function (CAPhone Community Score). Additionally, for the submitted community's evidence we follow the same approach, in order to measure how valid they are. Users can confirm or report an evidence and through that interaction, we measure an honesty score for them.

We divided the entities we are going to use in our computations in three main sections: Digital Product Types (4.2.2), Developer (4.3.2), User (4.3.3). Afterwards, we defined our score functions (subsections 4.3.4 and 4.3.5). In each section, we provide all the necessary denotations that we are going to use in our functions.

4.3.1 Digital Product Types

First of all, let sort ***PT*** denote the set of digital product types. As an example, we have PT = { SmartphoneApps, Smart TVs, Social Media }. In our case, as Digital Product Type we have Smartphone Applications. So, for simplicity, we will assume that all definitions in the sequel concern the same PT, (rather than adding a sub/superscript).

Moreover, each Digital Product Type includes Digital Products. In our case, these concern Android applications. Each application has some permission groups that require access to, along with their justifications. Their denotations are formalized in the following list.

- **Digital Product**

Let sort ***DP*** denote the set of digital products (of a particular Product Type).

▷ Variables: $DP_1, DP_2, etc.$

▷ **Example:** DP = { Viber, WhatsApp, Skype }

- **Permission Groups**

Let sort $\boldsymbol{P}$ denote the set of permission groups (of a particular Product Type).

▷ Variables: $P_1, P_2, etc.$

▷ **Example:** P = { Camera, Contacts, Microphone, Calendar }

- **Justifications**

Let sort $\boldsymbol{JT}$ denote the type of Justification (Purpose, Period, Storage Method, Shared With) and $\boldsymbol{J}$ denote the set of justifications optionally given by developers, in order to ask for a permission group (of a particular Product Type).

▷ Variables: $J_1, J_2, etc.$

▷ **Example:** J = { Undefined, Mandatory, QoS, Analytics }

So far, we defined all the denotations that are relevant to an application. We are going to use them to express developer requests (section 4.3.2) and user expectations (section 4.3.3).

4.3.2 Developers' Claims

A developer requests access to certain permissions, providing some justification.

Let the partial function **Request: DP** $\times$ **P** $\times$ **JT** $\rightarrow$ **J** return the justification for a request for a particular permission group (of a particular Product Type). The function is partial (i.e., not all values map to a specific J), since a DP_i may not ask for certain permission groups, therefore it does not make sense to have a justification for them. As an example, consider the following request:

$$\textit{Request(Viber, Contacts, Purpose)} = \textit{QoS},$$

which means that the Developer of Viber requests access to contacts for improving the quality of services.

4.3.3 Users' Concerns and Expectations

On the other hand, a user can declare his privacy expectations and privacy concerns regarding a particular access request. We are going to define those as privacy expectations function and privacy concerns function.

Let the function **Expectation : DP** $\times$ **P** $\times$ **J** $\rightarrow$ $\left[\mathbf{0,1}\right]$ return the percentage of users that find the justification for a request for a particular permission (of a particular Product Type) reasonable. As an example, assume the following expectations:

Expectation(Viber, Camera, QoS) = 0.1
Expectation(Viber, Camera, Mandatory) = 0.9

This example demonstrates that, 90% of users find reasonable access to Camera mandatory and only 10% have no objection to grand the permission for quality of service purposes.

At this point, we have to mention that for a given Digital Product X (DPX) and a specific Permission Y (PY), the percentage of users that find the justification for a request reasonable should sum up to 1. In other words, we need to have $\sum_{j \in J}$ Expectation(DPX, PY, j) = **1**.

Regarding the concerns, we let function **Concern : DP** $\times$ **P** $\times$ **J** $\rightarrow$ $\left[\mathbf{0,1}\right]$ return the percentage of users that find the justification for a request for a particular permission (of a particular Product Type) suspicious. As an example, let the following concerns:

Concern(Viber, Contacts, Mandatory) = 0.0
Concern(Viber, Camera, QoS) = 0.9

As we can see, there are two concerns related to application Viber for two different permission groups. In the first concern, all the users found the request for contacts as mandatory reasonable and that is why the percentage of concern is 0, but on the second concern 90% of users believe the request of camera for quality of service is suspicious.

Now that we have defined the domain ingredients (objects, concepts etc), as well as their behavior (functions, relations), we can define the functions that we use to compute the quantification of users' expectations as CAPhone Community Score and Developers' Honesty Score.

4.3.4 CAPhone Community Score

As already mentioned, with CAPhone Community Score we try to measure how close the users' expectations are, with respect to the developers' requirements and justifications. Our function CAPhoneScore will return values between $\left[0,1\right]$. The ranges we have set, are from 0.0 to 0.35 bad, 0.36 to 0.75 good and 0.76 to 1.00

very good. In this way the community can get immediate idea of how trustworthy an application can be. At the beginning all the applications that do not have an expectation from the community, have a CAPhone Community Score equal to 0.75.

Keep in mind that for the computation of the CAPhone Community Score we take only into account the expectations of the users. In subsection 4.3.3, we saw that the expectations include the permission groups that the developer requires access to. However, some permission groups may be more critical than others. Thus, inside the function, we denote a variable **WoI**, which can take values from $[0,1]$ and returns how critical in terms of privacy a given permission group is. For our implementation, we assumed that all permission groups have the same weight of importance score (WoI) and that is 1.

Another key aspect that we have to handle is that users may submit their expectations for some permission groups of a specific application, but there is a possibility to leave some of them undefined. We assume that a permission group is undefined, if a user has mentioned at least one expectation for a permission group of an application and has left the rest. On the one hand we could leave out undefined permission groups and compute the privacy friendliness score based on the expressed expectations. Though, on the other hand, there are a lot of possible interesting explanations, for not submitting their opinion and that is the reason why we decided to include them in our computations.

- **The Privacy Function**

A simple approach would be to calculate the **weighted sum** of the users' expectations that are in agreement with the developers' justifications. Then the Privacy Score function will be:

$$\textbf{CAPhoneScore(dp)} = (\sum_{i}^{m} [Expectation(dp, p_i, j) + Expectation(dp, p_i, Undefined)] * WoI(p_i))/m \tag{4.1}$$

such that Request(dp, p_i) = j (with j $\neq$ Undefined), and where m is the number of p_is for the given product.

A more intuitive approach is to calculate the **Mean Squared Error (MSE)**, a very popular and commonly used cost function.

$$\textbf{CAPhoneScore}^{MSE}\textbf{(dp)} = 1-(\sum_{i}^{m} [1-Expectation(dp, p_i, j)-Expectation(dp, p_i, Undefined)]^2 * WoI(p_i))/m \tag{4.2}$$

such that Request(dp, p_i) = j (with j $\neq$ Undefined), and where m is the number of p_is for the given product. This is the approach that we use in our implementation.

In Function 4.2 for a given set of users' expectations for a certain permission, i.e. n_1 users prefer justification j_1, n_2 users prefer justification j_2, etc, we are using the **Wilson Score Interval (WSI)**, with confidence level 95%, to calculate the percentage of people that prefer a specific justification j_x. The mean of WSI for some n_x gives the expected value that we want in order to determine how probable it is that the next preference will also be j_x. The upper and lower values give our confidence interval, that is the estimated deviation from our expected value. The shorter the interval, the more confident we are on the estimation.

For the undefined expectations, we have considered that users in general have the attitude to condemn situations where they believe they are being defrauded. When they totally agree with a statement, they are willing to express their opinion. The same happens when they have the impression that somebody tries to spoof them. Taking this into account, we handled undefined expectations as partially negative votes with weight of half a vote.

In other words, for the computation of CAPhone Community Score MSE based on Function 4.2, we gathered the aggregation of the expectations of the users for a specific application, including the undefined ones, and we subtracted them from 1, which denotes the developers' justification. We squared the result and we divided it with m, which is the number of permission groups for the given application, so that we get the average of the squares of the difference between the actual observations and those predicted. The value of Mean Squared Error represents the gap between developers' and consumers' opinion that we are going to subtract from 1 to get the final score. The higher the gap between them, the lower is the CAPhone Community's Score for the application. We have denoted that the CAPhone Score can take values between $[0,1]$. When the score is closer to 0, it means that the application is suspicious and when it is closer to 1 that is trustful. At the beginning, the CAPhone Community Score is equal to 0.75.

Finally, an even more elaborate function approach, would be to assign weights of importance to justifications as well. For example there could be a partial ordering among justifications, i.e.Undefined is the least preferred, QoS is more preferred than analytics etc. Then, we can calculate the MSE considering the distance of the developers' justifications to the users' expectations, where "unpreferred" justifications would penalize more the cost. In this way, the developer can have a hint as to how to minimize costs by changing the privacy policy.

4.3.5 Developers' Honesty Score

With this score we try to measure how honest a developer is with the consumers in a particular application. Developers require access to users' personal data, but a lot of times they use this data for different purposes or they have access to permission groups that users have not given their permission to. Thus, we use the Honesty Function to quantify how honest a developer is.

- **The Honesty Function**

When users of the community catch suspicious actions of the developers, they can submit an evidence and discredit them. Thus, in order to determine how reliable the evidence is about what the digital product is doing, with respect to what the developers claim it is doing, we use the Honesty function. With this function, we determine how valid an evidence is, based on the votes (confirm or report) of the community and will return values between $\big[0,1\big]$. We can use the previous approaches to establish it, but we will keep the highest suspicious evidence for simplicity. Afterwards, we subtract from 1 the score of the most valid mockery evidence, to get the final honesty score for a developer. When the score is closer to 0, it means that the developer is not honest with the users and when it is closer to 1 that they can trust him. At the beginning, the Developers' Honesty Score is equal to 1.

For all p_is of the given product, the resulting function is as follows:

$$\mathbf{DevHonestyScore(dp)} = 1 - max_i\big[Concern(dp, p_i, Request(dp, p_i)) * WoI(p_i)\big], \tag{4.3}$$

Chapter 5

Implementation & Use Cases

This chapter contains details related to the system implementation. It provides descriptions about the requirements and functionalities of the system and the basic architecture of CAPhone platform. Last but not least, we present use-case scenarios, appropriate for the demonstration of the proposed system.

5.1 Architecture of the System

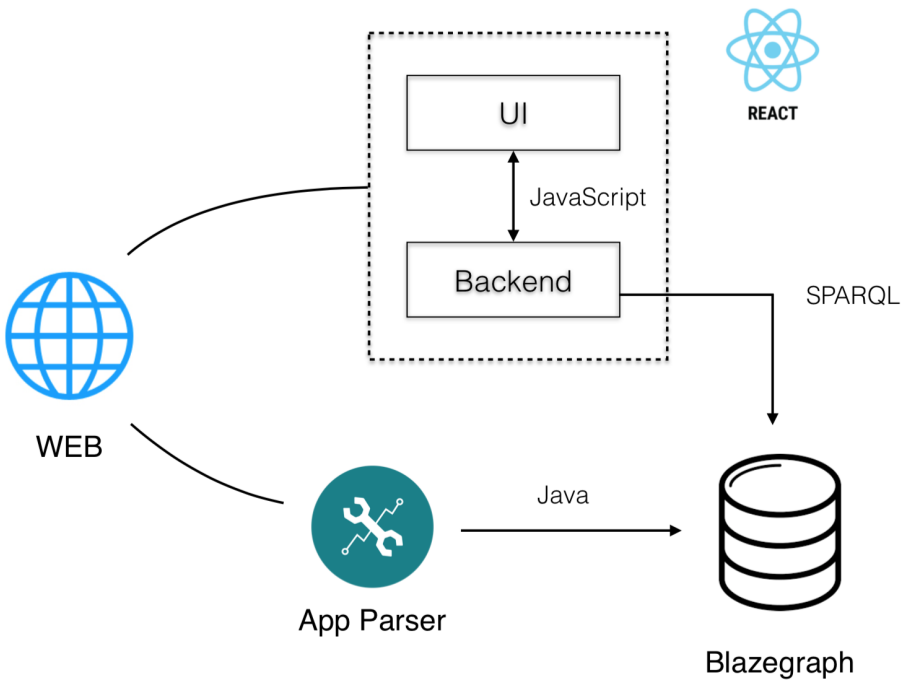

Figure 5.1: Schema of our Architecture

In Figure 5.1, one can see the schema of the architecture implemented on the system. For the UI of CAPhone platform React has been used, which is a JavaScript library. We also used JavaScript for the construction of the queries that we made to Blazegraph endpoint. The queries are expressed in SPARQL. Moreover, we made a Java Application Parser, in order to collect all the information that we need for the applications in Google Play Store.

5.1.1 The connection with Blazegraph

The designed system stores its data using the Resource Description Framework format (RDF) (2.2.1), which is a standard model for data interchange on the Web. It has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to change. Also, extends the linking structure of the Web to use URIs, to name the relationship between things as "triples". Each triple is composed of a subject (the resource), the predicate (the property name of the resource) and the object (the property value). Using this simple model, means allowing structured and semi-structured data to be mixed, exposed, and shared across different applications.

In order to retrieve or manipulate the data, we are using the semantic query language SPARQL (2.2.2), which is an official W3C Recommendation to work with RDF [3]. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. The results of SPARQL queries can be results sets or RDF graphs.

As SPARQL endpoint we are using Blazegraph, where we can remotely explore, access, and download the data stored using the SPARQL language. Blazegraph is a standards-based, high-performance, scalable, open-source graph database written entirely in Java. It supports Blueprints and RDF/SPARQL 1.1 family of specifications, including Query, Update, Basic Federated Query, and Service Description. Moreover, it supports novel extensions for durable named solution sets, efficient storage and querying of reified statement models, and scalable graph analytics. The database supports multi-tenancy and can be deployed as an embedded database, a standalone server, a highly available replication cluster, and as a horizontally-sharded federation of services similar to Google's Bigtable [1], Apache Accumulo [2], or Cassandra [3] [22].

At the beginning we had to create a namespace at Blazegraph, where we could load and query our data. Our namespace is called **"caprice"** and inside it, we have placed all the graphs that we used in our Ontology Schema. In Figure 5.2 we can see an example of how we can create a namespace in Blazegraph.

[1] https://cloud.google.com/bigtable/
[2] https://accumulo.apache.org/
[3] http://cassandra.apache.org/

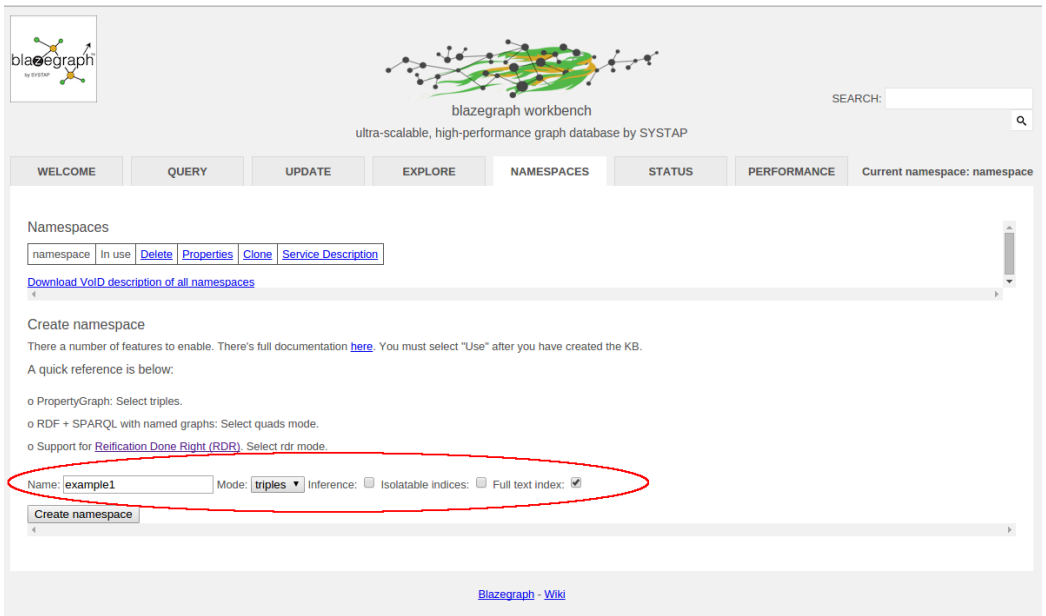

Figure 5.2: Creating a namespace in Blazegraph

Afterwards, in order to store in Blazegraph our ontology triples, we have a function in JAVA called SchemaGenerator, which based on the guidelines of the Blazegraph REST API, generates the relations of our entities that we have declared in separated classes. In those classes we define for each entity the name of the class and its properties. As an example we will see the structure of Privacy Policy in Figure 5.3. The name of the class is *"PrivacyPolicyClass"* and the properties are *"hasApplicationProp"*, *"hasPermissionGroupProp"* and *"hasJustificationProp"*. Furthermore, in Figure 5.4 we can see how Privacy Policy is connected with the other classes, according to the Ontology Schema we have seen in Figure 4.13.

```
public class PrivacyPolicy {

    public static final class SchemaProperties {
        public static final String hasApplicationProp = "PR1_has_application";
        public static final String hasPermissionGroupProp = "PR2_has_permission_group";
        public static final String hasJustificationProp = "PR3_has_justification";
    }

    public static final class SchemaClasses {

        public static final String PrivacyPolicyClass = "C9_PrivacyPolicy";
    }
}
```

Figure 5.3: Privacy Policy Class

```
// ============================================================ Privacy Policy ============================================================


private static void enrichPrivacyPolicyGraph(String privacyPolicyGraph, BlazegraphRepRestful blaze) throws UnsupportedEncodingException {

    TripleOps.addSchemaClass(blaze,  className: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaClasses.PrivacyPolicyClass, privacyPolicyGraph);

    TripleOps.addSchemaProperty(blaze,
            propertyName: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaProperties.hasApplicationProp,
            domain: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaClasses.PrivacyPolicyClass,
            range: NAMESPACES.get("caprice") + Application.SchemaClasses.ApplicationClass,
           privacyPolicyGraph);

    TripleOps.addSchemaProperty(blaze,
            propertyName: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaProperties.hasPermissionGroupProp,
            domain: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaClasses.PrivacyPolicyClass,
            range: NAMESPACES.get("caprice") + Permission.SchemaClasses.PermissionGroupClass,
           privacyPolicyGraph);

    TripleOps.addSchemaProperty(blaze,
            propertyName: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaProperties.hasJustificationProp,
            domain: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaClasses.PrivacyPolicyClass,
            range: NAMESPACES.get("caprice") + Justification.SchemaClasses.JustificationClass,
           privacyPolicyGraph);

    TripleOps.addTriple(blaze,
            s: NAMESPACES.get("caprice") + Evidence.SchemaClasses.EvidenceClass,
            p: NAMESPACES.get("rdfs") + "subClassOf",
            o: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaClasses.PrivacyPolicyClass,
           privacyPolicyGraph);

    TripleOps.addTriple(blaze,
            s: NAMESPACES.get("caprice") + Expectation.SchemaClasses.ExpectationClass,
            p: NAMESPACES.get("rdfs") + "subClassOf",
            o: NAMESPACES.get("caprice") + PrivacyPolicy.SchemaClasses.PrivacyPolicyClass,
           privacyPolicyGraph);
}
```

Figure 5.4: The Schema of Privacy Policy Entity

5.1.2 The structure of queries

The next part is the connection between our platform and Blazegraph endpoint. We wish to be able to access data from the triple store and present them to our system, as well as to compose data, generate new data, and store data back to the repository. Thus, we are using **Axios**, a promise-based HTTP client for JavaScript which can be used in our frontend application and in our Node.js backend. By using Axios, we can send asynchronous HTTP requests to our REST endpoint.

In Figure 5.5 we can see our class SparqlClient, which achieves the desired connection that we described before. There are two functions that we can see in the middle of the screen. The first one, **"executeQuery"**, is used when we want to make a **"SELECT"** query to retrieve data from Blazegraph. With the second function **"executeUpdateQuery"** we can make an **"UPDATE"** query to change our data. The next paragraph will present the structure of queries and a demonstration of them.

```
import axios from "axios";
import qs from 'qs' ;

export default class SparqlClient {
    static host = `http://${window.location.hostname}:9999/blazegraph/namespace/caprice/sparql`;


    static serializeTriplesToObjects = (response) => {

        const results = [];

        const triples = response.results['bindings'];
        triples.forEach(item => {...});

        return results;
    };

    static *executeQuery(query) {
        return yield SparqlClient.execute( queryType: "query", query);
    }

    static *executeUpdate(query) {
        return yield SparqlClient.execute( queryType: "update", query)
    }

    static *execute(queryType, query) {
        console.log(query);

        const data = qs.stringify( object: {
            [queryType]: `PREFIX cpr: <http://caprice/> ${query}`
        });

        const headers = {
            "Content-Type": "application/x-www-form-urlencoded;charset=UTF-8"
        };

        const response = yield axios.post(SparqlClient.host, data,  config: {
            headers,
            responseType: queryType === "query" ? "json" : "text",
            transformResponse: [data => {
                if (queryType !== "update") {
                    return SparqlClient.serializeTriplesToObjects(data);
                } else {
                    return {
                        "status": data.includes("totalElapsed") ? "success" : "failed"
                    }
                }
            }]
        });

        return response.data;
    }

}
```

Figure 5.5: The connection of CAPhone with Blazegraph

For the implementation of the frontend part of our platform, we use React which is a JavaScript library. As we mentioned in Section 2.3, React uses components that enable the splitting of the UI into independent, reusable pieces, and handling each piece in isolation. Conceptually, components are like JavaScript functions. They accept arbitrary inputs, called "props" and return React elements describing what should appear on the screen.

While interacting with the platform, certain elements may need to change. For instance, when a user wants to submit his expectation about a permission group, we have to make an UPDATE query. Apart from the visual changes of UI, that a user selected a button and the re computation of privacy friendliness score, all the other components should remain the same. This means that we want to access data that are related to a specific component. For that reason we are using **Redux**

Saga, a library that aims to make application side effects (i.e. asynchronous tasks like data fetching and impure like accessing the browser cache) easier to manage, more efficient to execute, simple to test, and better at handling failures [4].

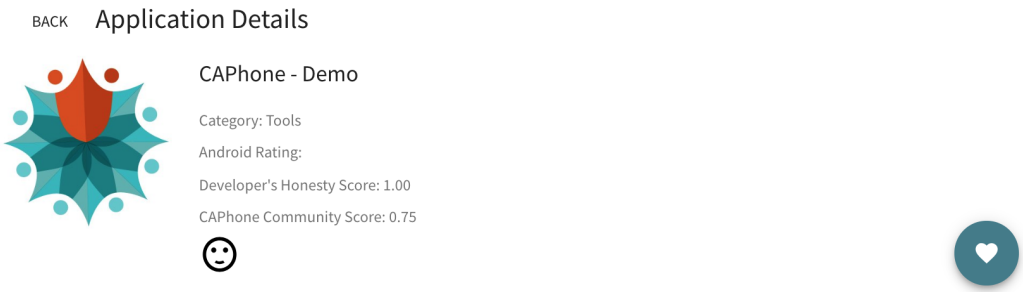

Figure 5.6: Basic details of an application

As an example of how we fetch and submit data using sagas, we will describe the section of the application details and the option of adding it to our favorite applications. In Figure 5.6 we see some basic information about the application like name, image, category and the Developer's Honesty Score, which presents how honest a developer is as we described in Section 4.3.5, and the CAPhone Community Score, which is the privacy friendliness score that we described in Section 4.3.4.

In Figure 5.7 we can see the query that we used to SELECT the desired data, as well as the use of function **"executeQuery"**. Moreover, in Figure 5.8 we can see the query that we used to INSERT this application to our Favorites and the use of function **"executeUpdateQuery"**.

```
import { takeLatest, put, call } from "redux-saga/effects"
import Types from "../types";
import Actions from "../actions";
import SparqlClient from "../../../../../rdf/SparqlClient";


function* fetchAppDetails(action) {
    const { appID } = action.payload;

    yield put(Actions.fetchPending(appID));

    const query = 'SELECT ?uri ?name ?image ?category ?description ?rating ?developer WHERE {' +
            `<${appID}> <http://caprice/AP2_has_name> ?name;` +
                        '<http://caprice/AP3_category> ?category;' +
                        '<http://caprice/AP9_developed_by> ?developer;' +
                        '<http://caprice/AP5_has_description> ?description.' +

            `OPTIONAL {<${appID}> <http://caprice/AP13_image> ?image. }` +
            `OPTIONAL {<${appID}> <http://caprice/AP7_has_rating> ?rating. }` +
            `values ?uri { <${appID}> }`+
        '}';

    try {
        const result = yield call(SparqlClient.executeQuery, query);
        yield put(Actions.fetchSuccess(result));
    } catch (error) {
        yield put(Actions.fetchFailed(error));
    }
}

export default function* watchFetchAppDetails() {
    yield takeLatest(Types.FETCH, fetchAppDetails)
}
```

Figure 5.7: Query for fetching basic details of an application

```
import { takeLatest, put, call } from "redux-saga/effects"
import Types from "../types";
import Actions from "../actions";
import SparqlClient from "../../../../../rdf/SparqlClient";


function* addToMyApps(action) {
    const { appUri } = action.payload;

    yield put(Actions.addToMyAppsPending(appUri));

    const userUri = `<${localStorage.getItem( key: "user")}>`;

    const query ='INSERT DATA {' +
            `GRAPH <http://caprice/personGraph/> { ${userUri} <http://caprice/UP7_has_installed_apps> <${appUri}> }`+
        '}';

    try {
        const result = yield call(SparqlClient.executeUpdate, query);
        yield put(Actions.addToMyAppsSuccess(result));
    } catch (error) {
        console.log('got error');
        console.dir(error);

        yield put(Actions.addToMyAppsFailed(error));
    }
}

export default function* watchAddToMyApps() {
    yield takeLatest(Types.ADD_TO_MY_APPS, addToMyApps)
}
```

Figure 5.8: Query for inserting an application to our favorites

5.2 System Requirements

At the beginning of this section we are describing the features and behavior of our platform, based on the system requirements we have set. Because this is the first version of our platform, our requirements depend on the essential functionalities that should be provided. In Chapter 6, a more detailed approach on them will be provided, in order to gain a more complete picture of the whole concept we want to produce. Afterwards, we will give a presentation of the whole platform and demonstrate use case scenarios, to explain how a user can interact with the system. The basic requirements of CAPhone platform are the following:

1. The user should be able to register and login into the platform.

2. The Authentication should be based on email and password.

3. There must be a difference for the registration of the basic users and the developers.

4. There must be a difference in the UI between users and developers when they log in to the platform.

5. There must be a page with the developers' applications.

6. There must be a page with the users' favorite applications.

7. There must be a page with all the available applications.

8. There must be a basic search page for the applications.

9. Developers should be able to submit their applications inside the platform.

10. Developers should specify same basic information about the application like name, image and a small description.

11. Developers should specify the permission groups that their application wants access and the justification for that.

12. Users (and Developers) should be able to express their willing to permit or deny access to a developer's privacy policy.

13. Based on the total expectations of the community to exist a privacy friendliness score.

14. If the users find anything suspicious for the application they should be able to submit the evidence as a link.

15. Users should be able to check the mockery evidence that the community has found.

16. Users should be able to rate the evidence that the community has found.

17. There must be an honesty score for the developer in each application, based on the most valid mockery evidence the community has found.

5.3 Presentation of the CAPhone Platform

In this section, we present the CAPhone platform using three use cases, taking into account the requirements of the system. At the beginning, we describe a scenario with a developer, then with a simple user and finally with a more advanced user.

5.3.1 A Use Case for the Developers

A Developer is willing to use CAPhone platform, in order to clarify which permission groups his application wants access to and for what purpose, so that consumers have full knowledge of which personal data they share. His application is called "CAPhone - Demo" and requires access to the following permission groups with their justifications, as we can see in Table 5.1. Finally, the developer wants to be able to see all the applications he has added to the platform.

Permission Group	Purpose	Period	Storage Method	Shared With
SMS	Never	-	Encrypted	-
Camera	-	Session	-	-
Location	-	Fixed Time	Anonymized	Third Parties

Table 5.1: Permission Groups and Justifications of "CAPhone - Demo"

First of all, the developer in order to enter the platform, has to register. In Figure 5.9, we see the **Login** page of CAPhone (Requirement 1), where he has to submit his email and password (Requirement 2). The developer does not have an account yet, so he has to select the "Register" button on the bottom right of the page. Next, in the **Registration** page, he has to submit the following information (Figure 5.10): ***First*** and ***Last name, Email, Age, Password***. Afterwards, because he is a developer (Requirement 3), he should select the corresponding button to define it. Then, three additional fields appear that he has to fill in, about his physical ***Address, Website URL*** and a url with his general privacy policy, ***Policy URL***. The last three fields exist as information in Google Play Store, thus we require them. Finally, he has to enter the "Submit" button to submit his information, and he will be redirected to the Login page, where he has to fill in his email and password.

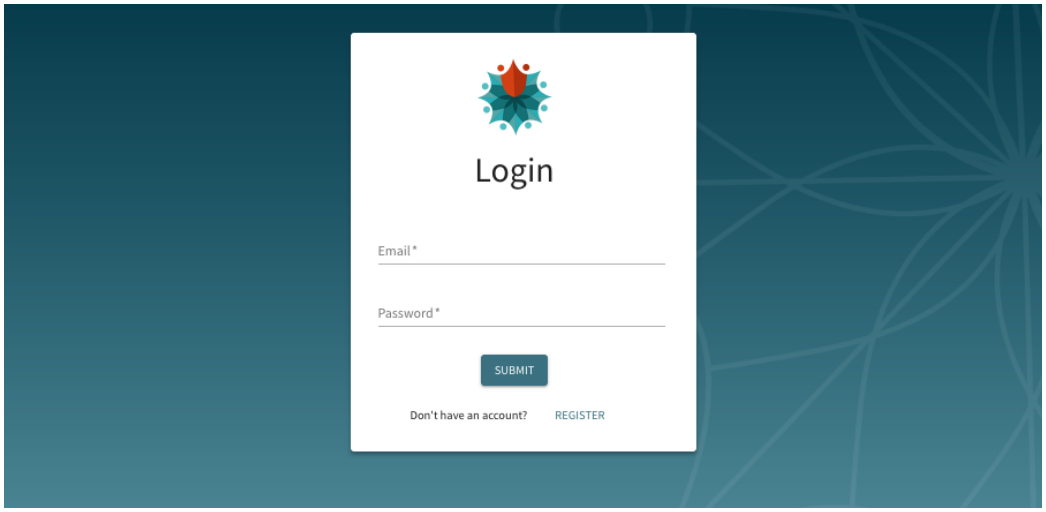

Figure 5.9: The Login page of the CAPhone Platform

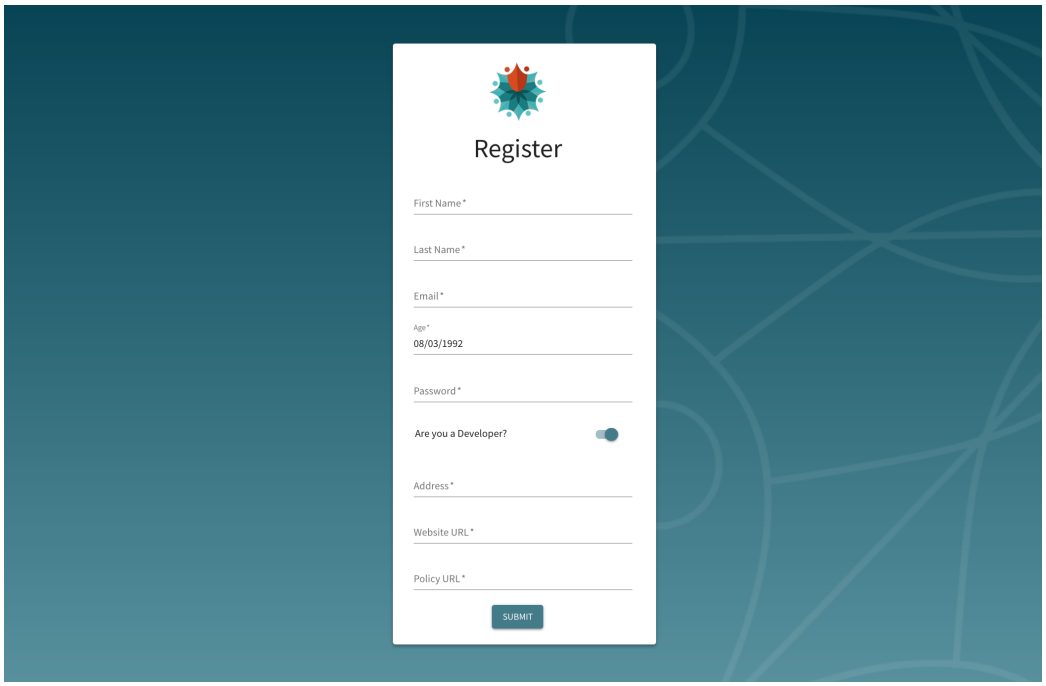

Figure 5.10: The Registration page for the Developers

When the developer enters the platform, he sees the main Dashboard, which has the Digital Products that are available. In our case, which concerns **Smartphone Applications**, this is what he sees. On the left side menu, there are five options, which are the following: ***Search, Platforms, Favorite Applications***, which are common for both Developer and Users, ***My Applications, Add Application*** (Requirement 4). All these options are accessible from everywhere in the platform, so they can easily navigate to the sections they are interested in. The developer in order to submit his application to the platform, has to select "Add Application".

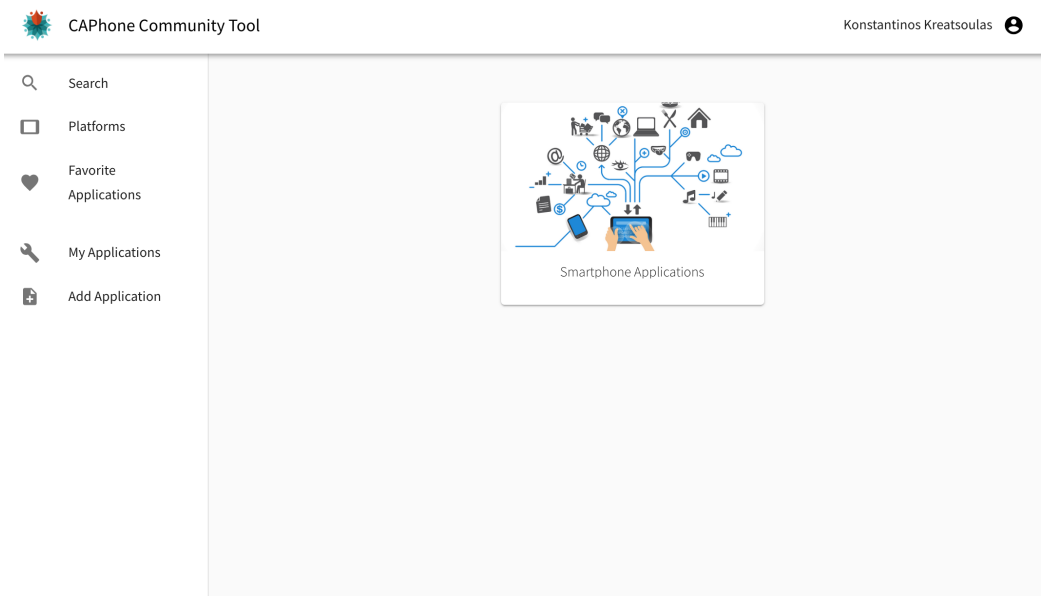

Figure 5.11: The Dashboard page of the CAPhone Platform

Then, the developer is moved to a new page, where he has to specify the appropriate details for his application (Requirement 9). On the top half of the page (Figure 5.12), there are the basic details about the application (Requirement 10) such as the ***Application Name***, an ***Image URL***, the ***URL of the Application***, the ***Category*** and ***Description*** that the developer has to fill in. On the bottom half (Figure 5.13), there is a table with permissions that he has to specify where the application wants access to and to justify the ***Purpose, Period, Storage Method*** and to who those information will be ***Shared With*** (Requirement 11). For all of these justifications there is a drop down list where the developer can select his preferred value.

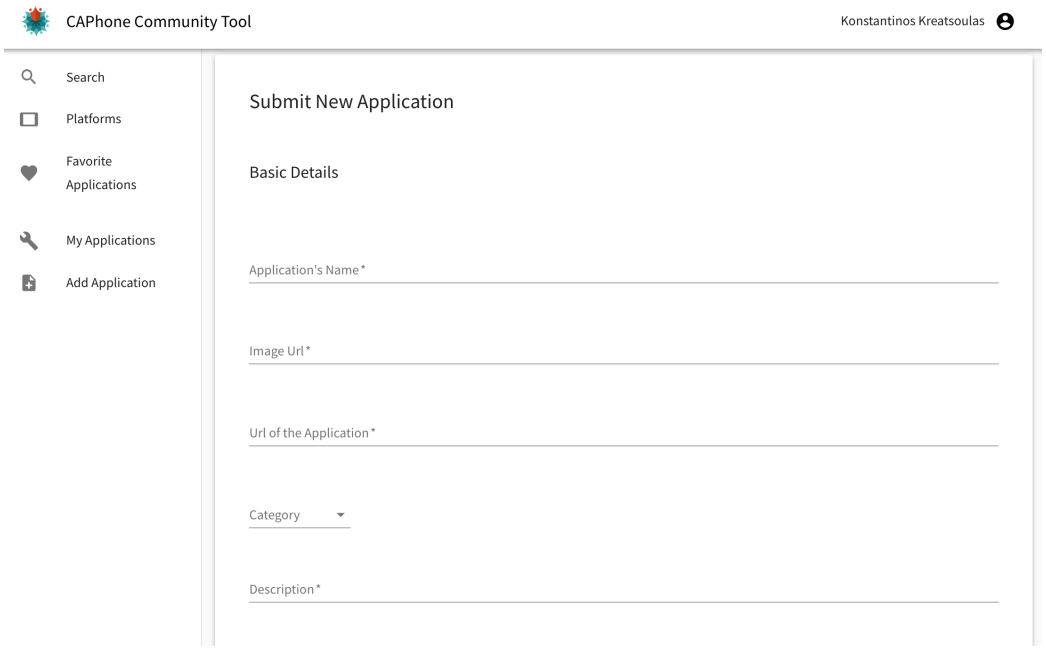

Figure 5.12: The Basic Details of a new application

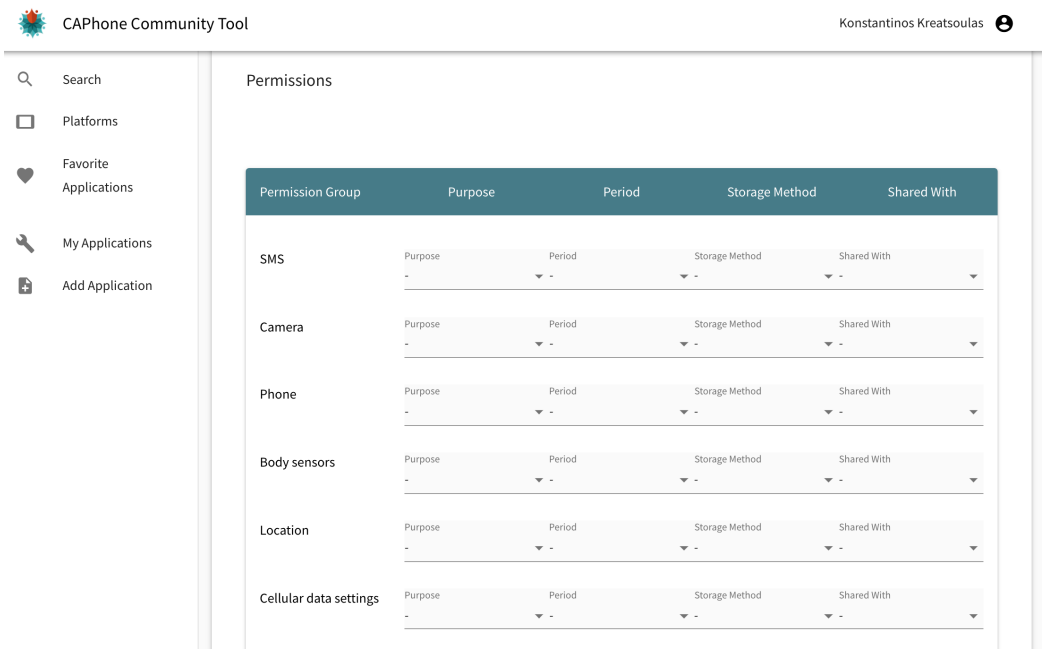

Figure 5.13: The Permission Groups and their Justifications of the new application

After all compulsory fields are filled in, he can submit his application to the platform. As we have said in the introduction of this subsection, the developer submitted an application called **"CAPhone - Demo"**, which requires access to SMS, Camera and Location and specified their justifications, as we can see in Table 5.1. All of them are randomly set for demonstration purposes. Also, the application has been added in the section of *"My applications"* on the left side menu (Figure 5.14), so the developer can have in one specific place his own applications (Requirement 5).

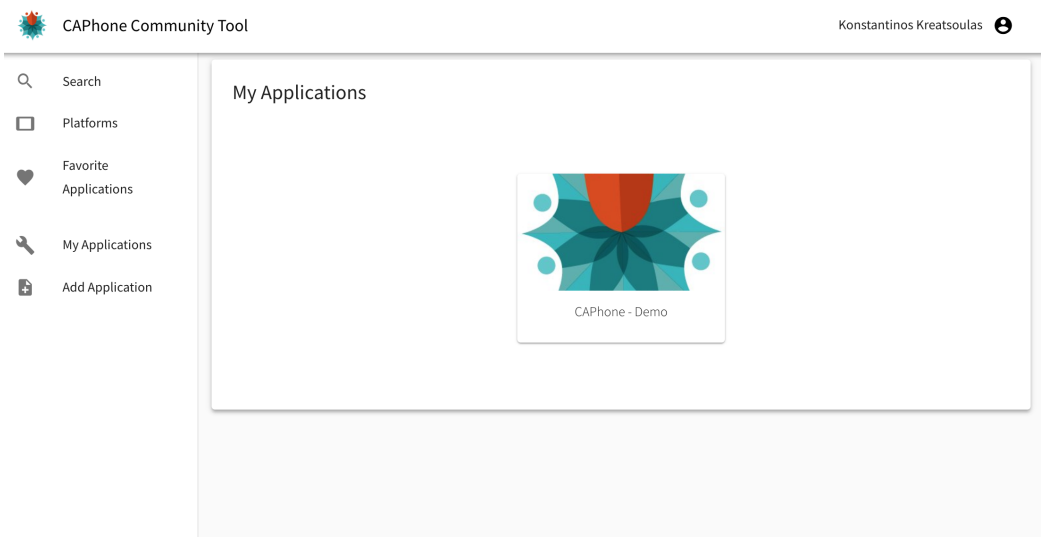

Figure 5.14: A sample of Developers' applications

5.3.2 A Use Case for the Basic Users

A basic User wants to look for Android Smartphone Applications in the CAPhone platform. Afterwards, he wants to search for the application "CAPhone - Demo", to learn some information about it i.e. description, permissions and more importantly the specifications that the Developer has given about the permission groups that the application requires access to. Finally, he wants to express his expectations about the permission groups of the application.

At the beginning, the User has to register on the platform. The registration page is the same as for the developers, with the only difference, that the last three fields (Address, Website URL, Policy URL) are missing (Requirement 3). When the User enters the platform the first thing he sees is the Digital Products that are available, which in our case, are the Smartphone Applications, similar to the Developer Use Case. By entering the Smartphone Applications section, the User sees two platforms, Android and iOS (Figure 5.15) and he is going to select the

Android one. The iOS platform exists there for demonstration purposes. In Figure 5.16 the User sees a list of Android applications (Requirement 7), where he can find what is he looking for. Another way to search for an application, is to use *"Search"* on the left side menu. In this menu, the sections "My Applications" and "Add Application" are missing, because the users are not able to submit applications to the platform.

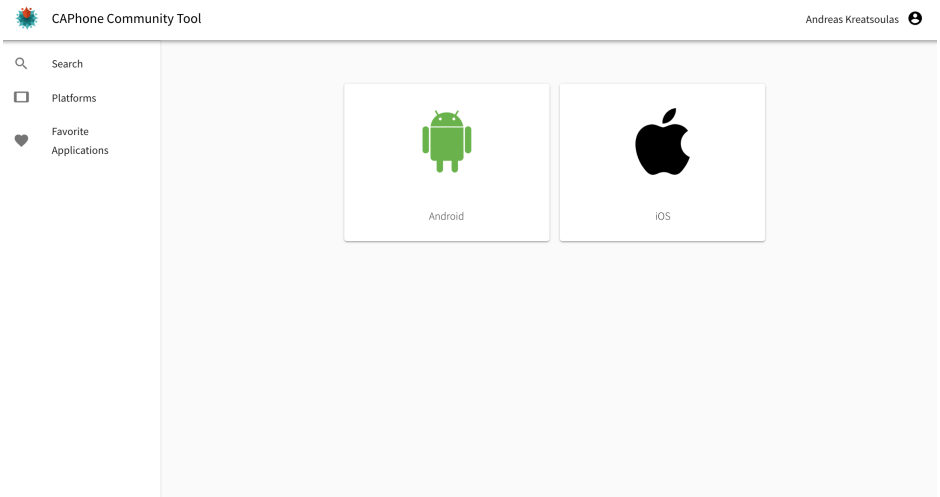

Figure 5.15: Available platforms in smartphone applications

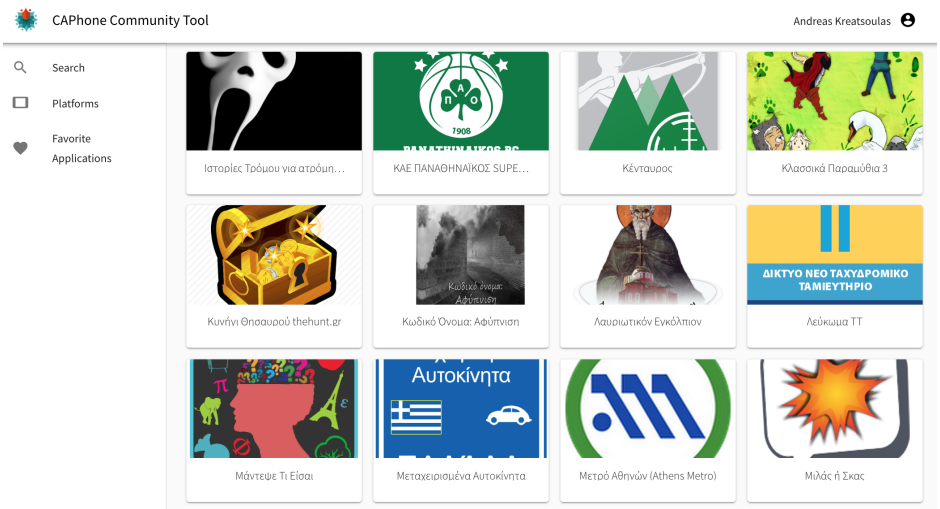

Figure 5.16: A sample of Android applications

Afterwards the User, will search for the application "CAPhone - Demo" by selecting the *"Search"* on the left side menu (Requirement 8) and by specifying the name, as we can see in Figure 5.17. The results appear at the bottom of the page. He will select the application, so he can see more details about it. In Figure 5.18 we can see the main page of the application and the details provided.

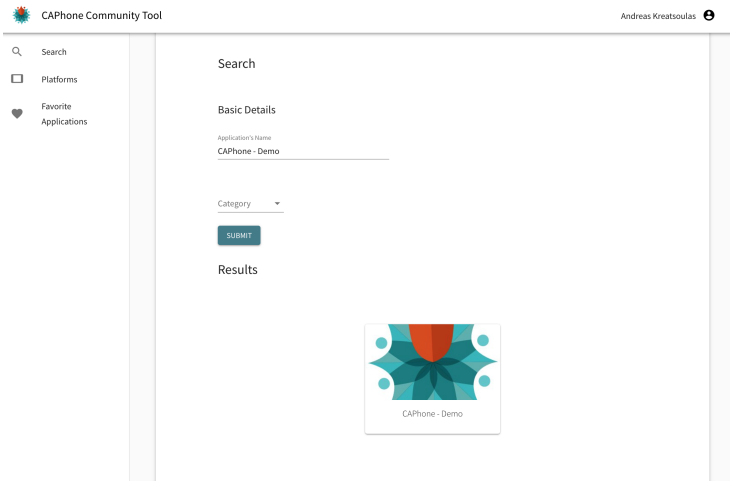

Figure 5.17: Search for the application CAPhone - Demo

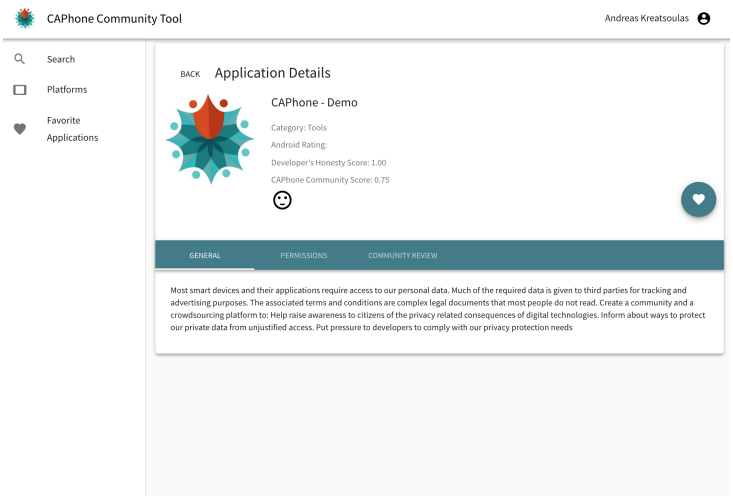

Figure 5.18: The Main page of application CAPhone - Demo

The main page of the application "CAPhone - Demo" is divided into two parts. In the top part we have the application's details, such as the name, category, Android rating (in our case this rating is not specified), Developer's Honesty Score (Requirement 17), which at the beginning is 1 and the CAPhone Community Score (Requirement 13), which at the beginning is 0.75, as we have explained in section 4.3, cause there are not any evaluations from the community yet. Also there is a heart button on the right side, with which the User can add the application to his Favorite Applications (on the left side menu). In the bottom part, there are three main tabs. In **General**, there is the description of the application (Figure 5.18), in **Permissions** there are the permission groups with some details that the developer wants access to (Figure 5.19) and at last we have the **Community Review**.

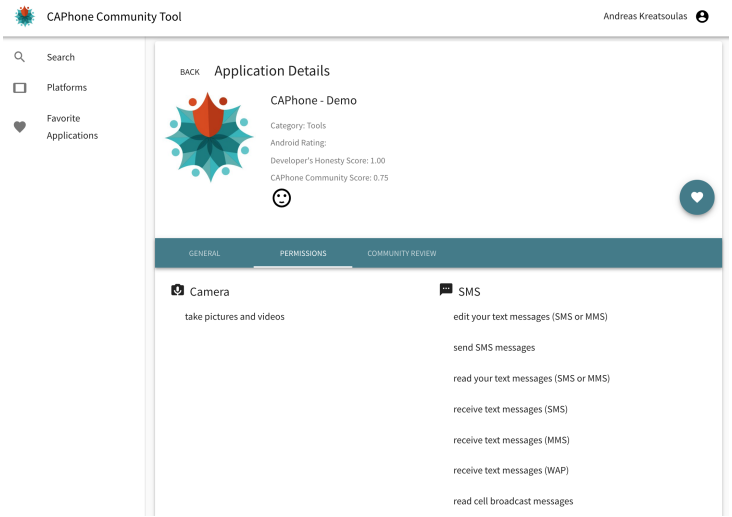

Figure 5.19: The permissions of application CAPhone - Demo

In that section, as we can see in Figure 5.20, users can express their expectations for the permission groups that developer wants access to (Requirement 12). When they click on a permission group, the row expands and the users can see the justifications of the developer for the current permission group, if there is any evidence for them and some general statistics about the *Total evaluators* that have expressed their opinion and based on them, *The percentage of Users who agree* and *The percentage of Users who disagree* (Figure 5.21). Moreover, users can submit probable mockery evidence that proves that the application uses a permission group without informing the customers or does something different from the statements of the developer, by selecting the **Add Evidence** (Requirement 14). All the evidence that the community has found, appear in the table at the bottom of the page and the users can confirm or report them (Requirement 15). At that point all of the above do not have any information because the application was just added and there are neither users' expectations or evidence.

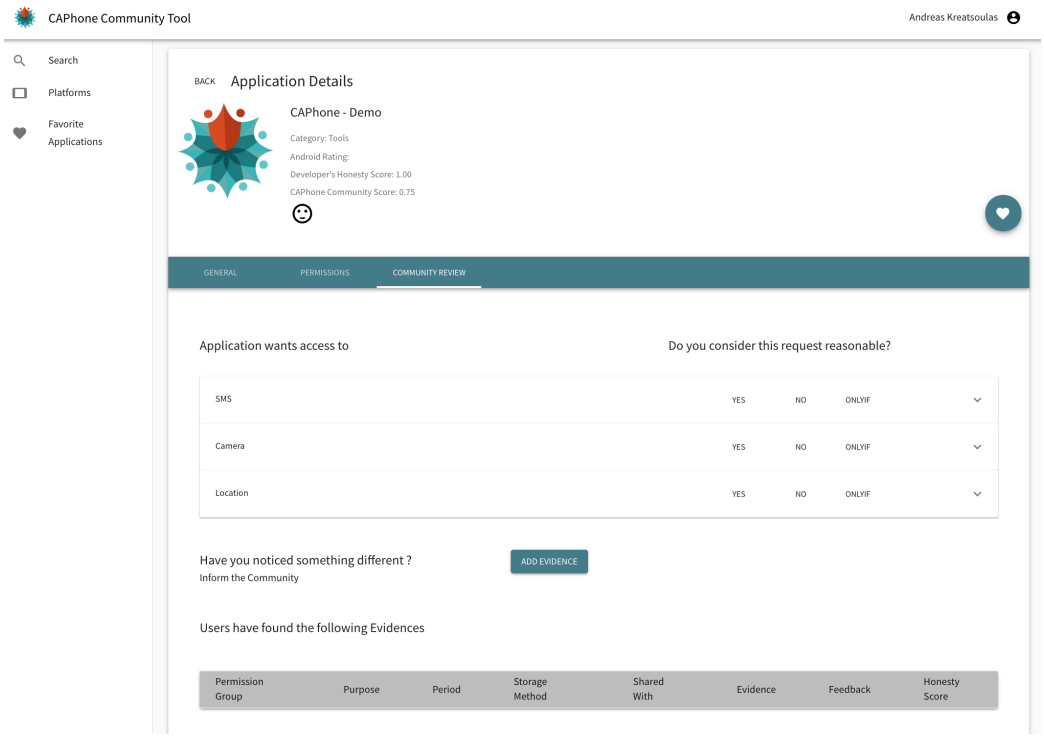

Figure 5.20: The Community Review section of application CAPhone - Demo

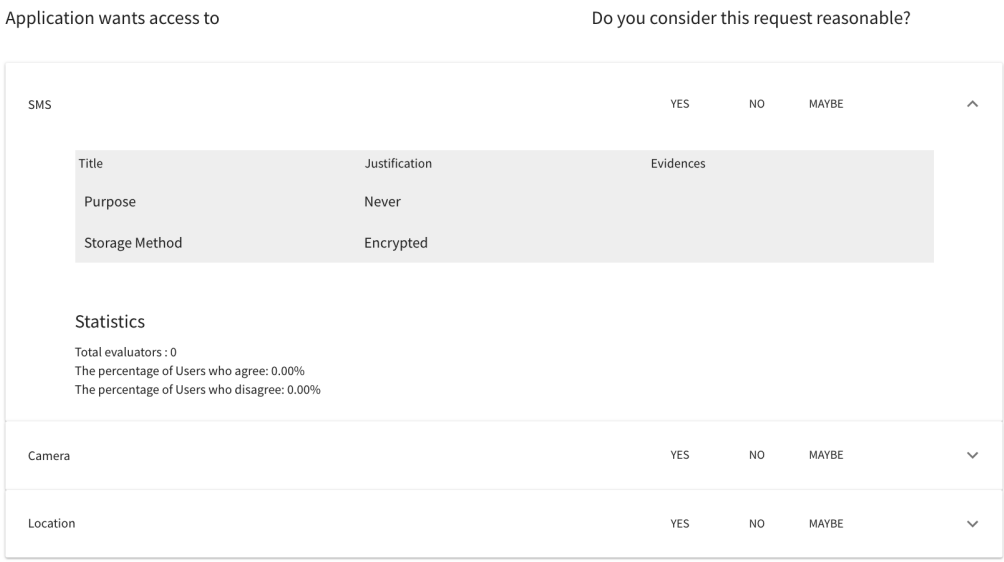

Figure 5.21: The Permission Groups appeared in Community Review section, with their Justifications and some statistics

In this Use Case, the User finds it reasonable for the application "CAPhone - Demo" to want access to permission group SMS for purpose "Never" and storage method "Encrypted". For that reason he permits that access by selecting "Yes", which is filled in with green color (Figure 5.22). Moreover, he is not sure about the other two permission groups and he does not select any value. As we can see, when the User permits the permission group SMS, the CAPhone Community Score changed from 0.75 to 0.84, based on the Privacy Function (subsection 4.3.4). Also, the statistics of this permission group changed accordingly. There is only one evaluator, and the percentage of Users who agree is 100%.

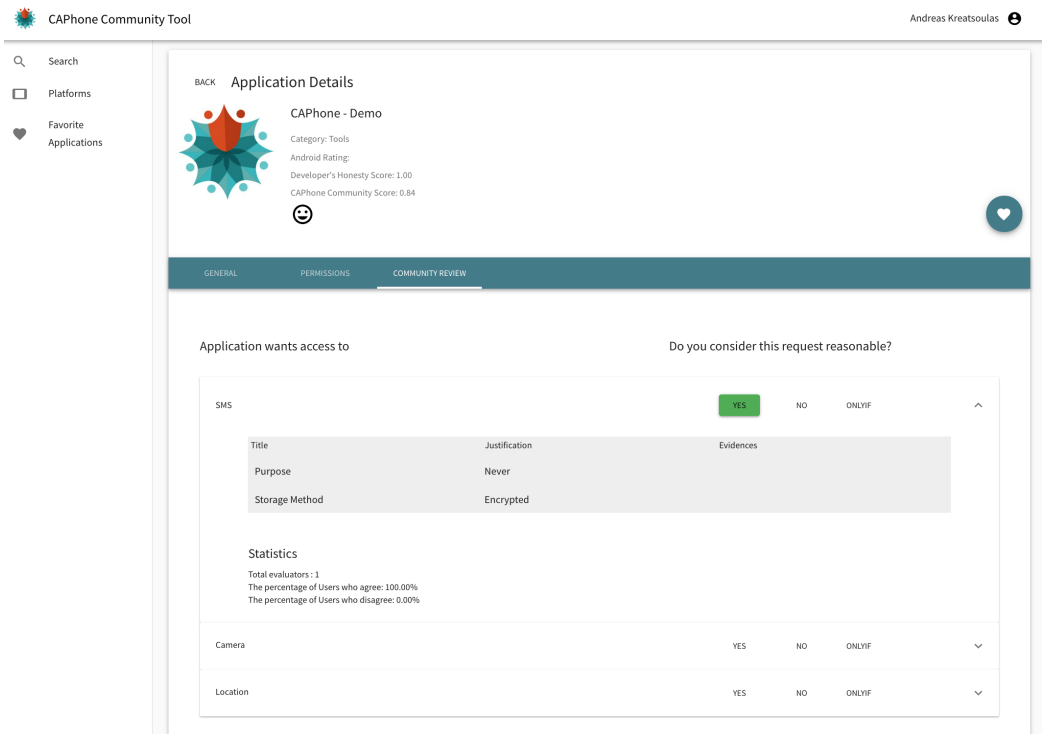

Figure 5.22: The screen after the User submitted his expectations

5.3.3 A Use Case for a more experienced User

A more experienced User wants to express his concerns and expectations about the application "CAPhone - Demo". He has found a mockery evidence that proves that the application has access to the permission group Microphone and wants to submit it, so the other members of the community get informed. Moreover, he wants to keep track of this application, in order to check if the rest of the community find the evidence reasonable. Thus, he wants to add the application in a separate list.

After the user, registers and logs in to the platform he searches for the application "CAPhone - Demo". In the Community Review section, he does not find reasonable the developer's need, to access the permission group SMS and Camera. For that reason he decides to refuse access to them, by selecting the "No" option. He wants to express his concerns about the permission group Location. He disagrees with the justification that Location is shared with "Third Parties". Thus, he decides to select the "Maybe" option. By selecting it, a dialog appears, where he defines that he would have given access to the permission group Location, if it was shared with "Nobody". As we can see, after the User submits his concerns and expectations, the CAPhone Community Score changed from 0.84 to 0.82 (Figure 5.24).

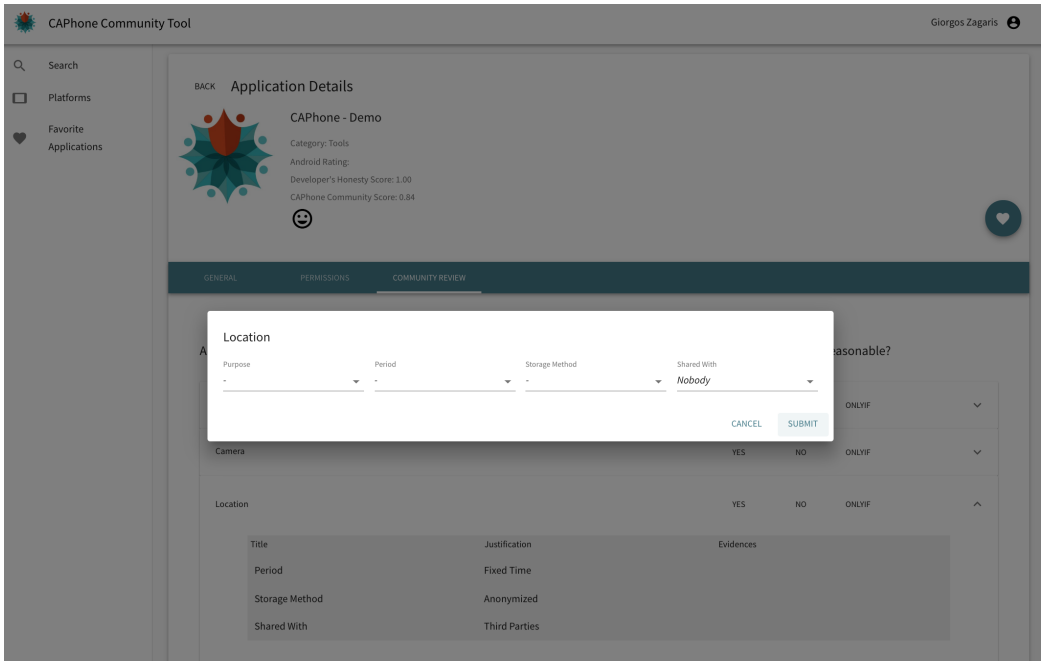

Figure 5.23: The dialog for adding an expectation

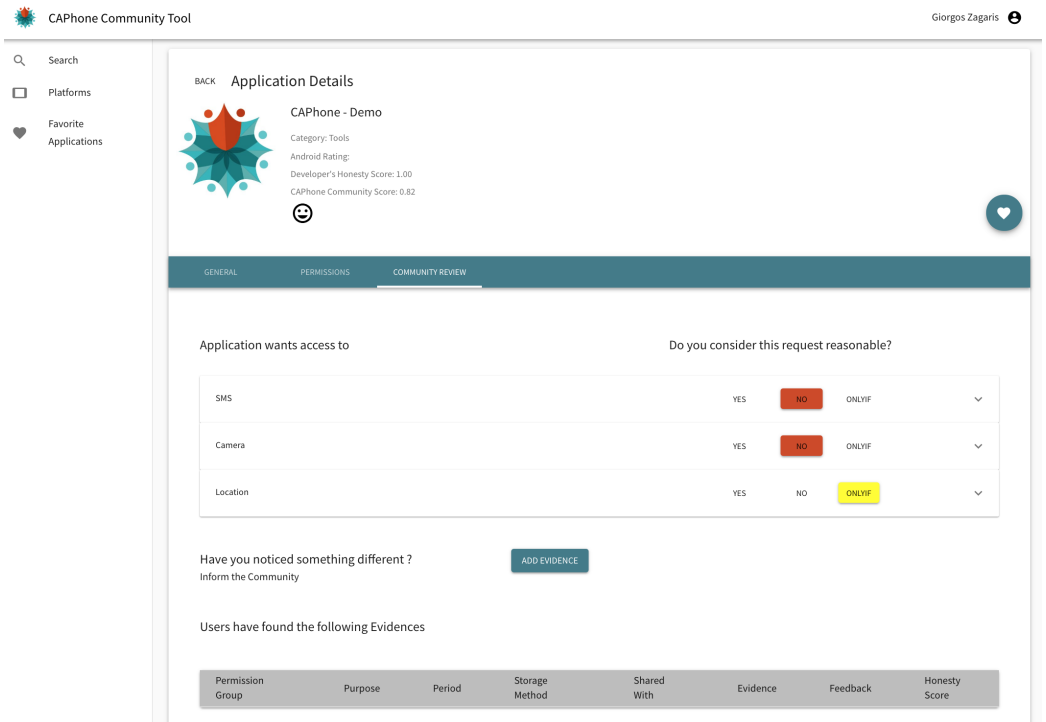

Figure 5.24: The screen after the User submitted his concerns and expectations

Moreover, the User wants to submit the mockery evidence he has found, that proves that the application has also access to the permission group Microphone. In order to submit the evidence, he selects the *"Add Evidence"* button. Then a dialog appears, where the User has to specify the permission group of the evidence, a justification for it and an evidence link. In our case, the User found that the application has access to the Microphone, with purpose "Quality Services". The link that proves that, is "www.demoMic.com" (Figure 5.25).

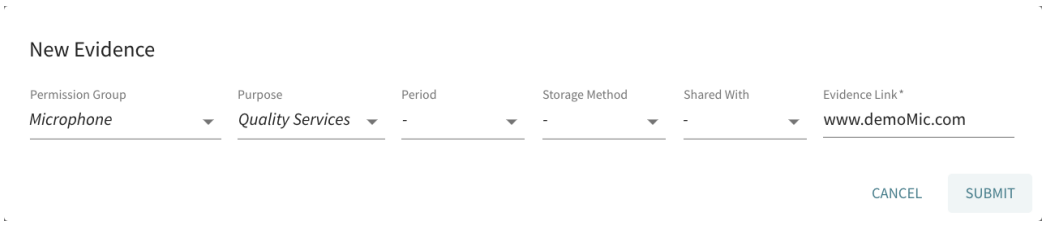

Figure 5.25: The dialog for adding new evidence

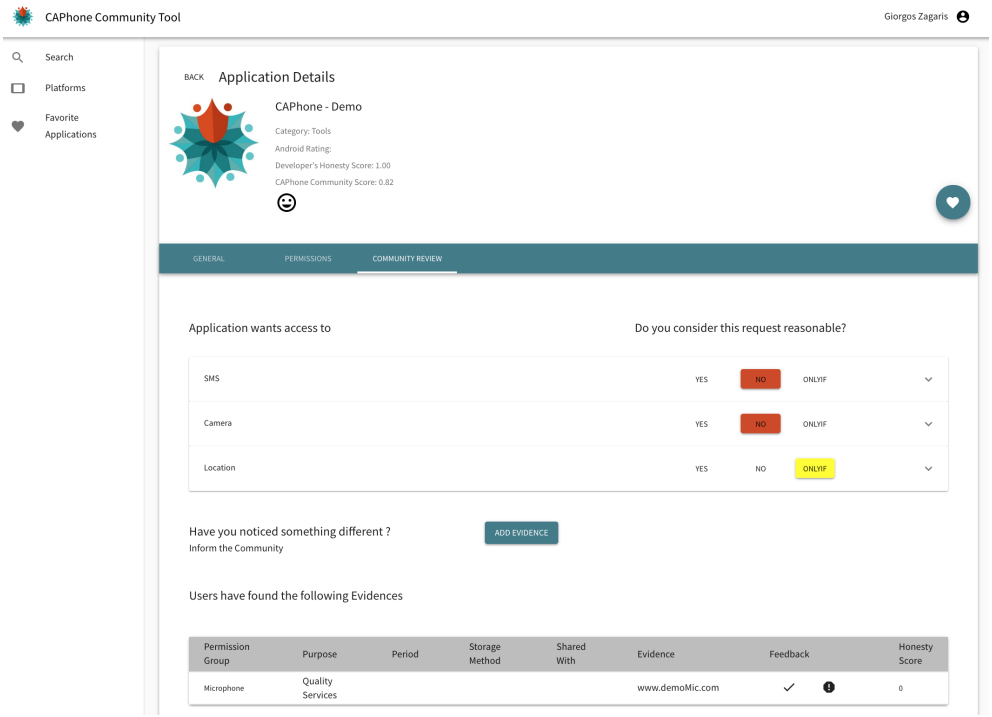

Figure 5.26: The screen after the User submitted his evidence

In Figure 5.26 we can see the evidence that the experienced User submitted to the platform, in order to inform the other members of the community. As we can see, at the bottom of the page, there is a table with all the evidence the community has found for the application CAPhone - Demo. The first column is about the permission group and the next four columns about the justifications. The sixth column provides the link for the evidence and next to it, there is a column named *"Feedback"*, where the users can confirm or report the evidence (Requirement 16). The last column is about the Honesty Score, which represents how valid an evidence is, based on the Honesty Function (subsection 4.3.4).

Finally, the User, in order to keep track of the application CAPhone - Demo, he wants to add it in a separate list. So, he has to select the "Heart" button on the top right of the screen. The application is going to appear in the section *"Favorite Applications"* on the left side menu (Requirement 6), as we can see in Figure 5.27

The Basic User that we referred to in the previous section, saw the mockery evidence about the CAPhone - Demo and he thinks that is valid. Thus, he decided to confirm it. As we can see in Figure 5.28, the honesty score of the evidence changed to 0.6 and also the Developer's Honesty Score at the top of the page, dropped from 1 to 0.4 (Requirement 17). That means that the Developer is not really honest with the consumers, because the community has found that he is also using the Microphone for Quality of Services, without informing them.

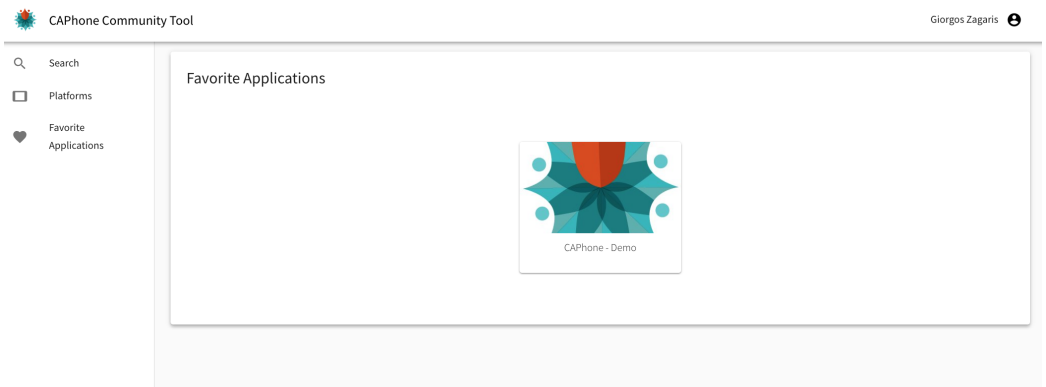

Figure 5.27: The favorite applications of the User

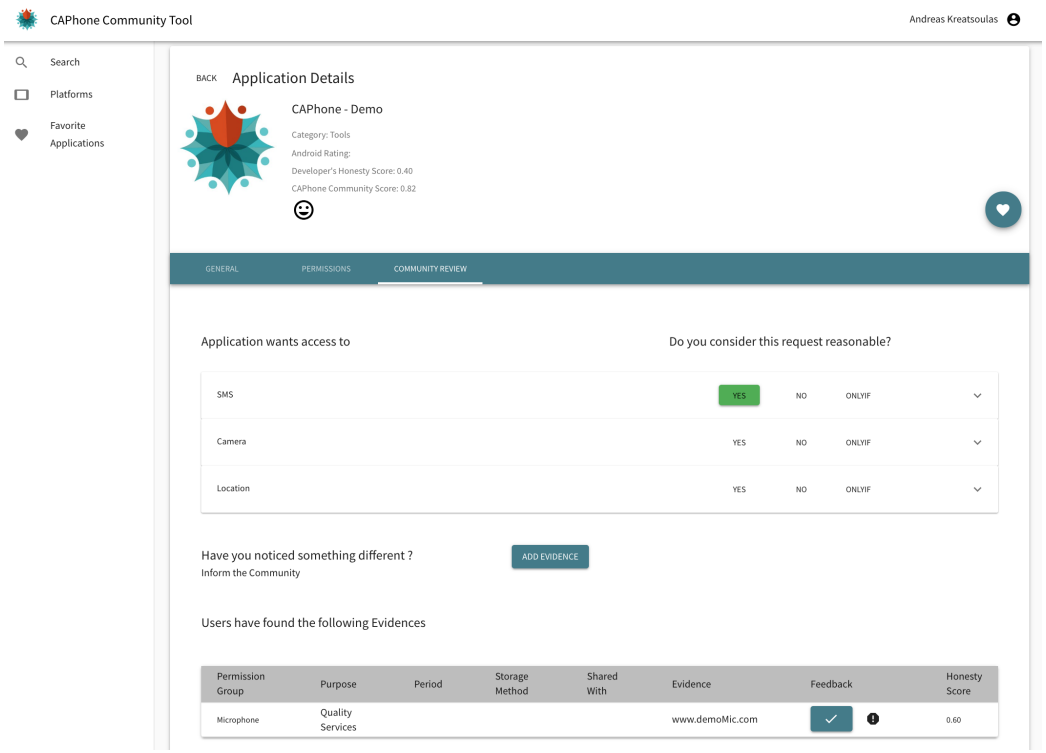

Figure 5.28: The changes in Developers' Honesty Score

Chapter 6

Conclusions and Future Work

After presenting our work and the corresponding implementation, we currently revisit the main outcome and technical contributions from a more abstract standpoint. Also we will remark prominent future work that we would like to implement, in order to improve the functionality of our platform, as long as the user experience.

At first we saw that a major problem nowadays, is that people generate vast amounts of data by using their digital smart devices, but do not really know who can collect it and what they can do with it. A suggested solution to that is collective awareness platforms, where users interact with each other as members of a community and they can share their knowledge, make better informed decisions as consumers, nudge collective environmentally savvy behavioural changes, and set up more participatory democratic processes.

This is the goal that we are trying to achieve through our collective awareness platform CAPhone, which will deliver a semantic repository of consumer and developer generated content about the privacy behaviour of digital products. Also, it will help consumers understand the Terms of Service and their implications via crowdsourced approaches, visual cues, and structured, goal oriented discussions. More specifically, with this platform, consumers can have an overall image of how privacy friendly an application is, how honest the developers are with the customers, and how the customers will think and act as members of a community.

For the implementation part of the platform, we have used Semantic Web to store and manipulate our data and for the construction of the platform we used React, Redux and Material UI. Moreover, in order to calculate the privacy friendliness and honesty score for each application, we made our own score functions with the use of Wilson Score Interval and Mean Squared Error.

This is the first release of our platform in which our main goal was to model how honest the developers are with the customers, and how the customers will think and act as members of a community. The functionality that we provide has this aim and as future work we would like to add some more features. Our goal is to create a privacy aware market, where the effort and time spent by the CAPhone Community members is considered a value mining process that creates rewards.

Each value mining process will be rewarded with a specific amount of points that will capture its importance and effort required. Further, each member will be rewarded CAPhone badges after mining a specific number of points, rewarding the members' dedication.

Rewarding points are important aspects of gamification, making interactions more fun and appealing, and creating a feeling of achievement. Badges help members identify themselves to their peers as persons who have reached a certain level of activity, which often serves as a powerful incentive, according to research [2]. After all, gamification and social rewards are vital elements of online community success for a number of reasons [1]. First, they promote user recognition: badges, ribbons and other visual indicators of prestige within a community platform are appreciated and sought after by many participants. Moreover, they build long-term engagement: participation in an online community has an element of habit to it, which becomes more difficult to abandon as the member obtains considerable rewards and recognition [2]. But apart from recognizing members that are already engaging, our objective for using scoring, gamification and badging is also to help signal to other members what types of engagement earn recognition, which we consider highly important for the quick involvement of new members in CAPhone.

Another addition that we want to have, is to give the ability to users to synchronize their digital devices with the platform. For instance, in this version, users have to find the Android applications that they have downloaded in their smartphones and afterwards to mark them as favorites. In the upcoming version we would like to offer a synchronization mechanism, where the users have the option to insert to the platform all of their applications based on their Android or iOS account. The same will apply for all the smart devices.

Last but not least, we would like to expand our Knowledge Base with new product types and to upgrade the design of our interface, in order to make it more user friendly. We are convinced that this platform will help people become more aware of how to control their personal data and will give them an extra motivation to interact with each other, which will also benefit their well being both, on a personal level and as a total.

[1] http://www.socialmediatoday.com/content/how-social-rewards-programs-are-changing-online-marketing

[2] http://www.gamification.co/2013/07/18/kaplan-gamification-system-shows-more-studentsengagement, http://www.pewinternet.org/2012/05/18/the-future-of-gamification

Bibliography

[1] Lorrie Faith Cranor Aleecia M. McDonald. The cost of reading privacy policies. *I/S: A Journal of Law and Policy for the Information Society*, 2008.

[2] A. Anderson, D. P. Huttenlocher, J. M. Kleinberg, and J. Leskovec. Steering user behavior with badges. *In 22nd International World Wide Web Conference*, page 95–106, 2013.

[3] Andy Seaborne Eric Prud'hommeaux. Sparql query language for rdf. https://www.w3.org/TR/rdf-sparql-query/, 15 January 2008. Last accessed January 2019.

[4] Flaviocopes. Introduction to redux saga. https://redux-saga.js.org/, February 21, 2018. Last accessed January 2019.

[5] Giorgos Flouris, Theodore Patkos, Ioannis Chrysakis, Ioulia Konstantinou, Nikolay Nikolov, Panagiotis Papadakos, Jeremy Pitt, Dumitru Roman, Alexandru Stan, and Chrysostomos Zeginis. Towards a collective awareness platform for privacy concerns and expectations. *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", At Valletta, Malta,*, pages 135–152, October 2018.

[6] Enrico Franconi Grigoris Antoniou and Frank van Harmelen. Introduction to semantic web ontology languages. *Principles and Practice of Semantic Web Reasoning*, pages 1 – 21, 2005.

[7] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, pages 199–220, 1993.

[8] Heng Xu Haiyan Jia. Measuring individuals' concerns over collective privacy on social networking sites. *Cyberpsychology: Journal of Psychosocial Research on Cyberspace*, 2016.

[9] Pete Hunt. Txjs. https://youtu.be/A0Kj49z6WdM, 2015. Last accessed January 2019.

[10] Norman Sadeh Jialiu Lin, Bin Liu and Jason I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. *Symposium On Usable Privacy and Security (SOUPS)*, pages 199–212, 2016.

[11] Anirban Dasgupta Lawrence D. Brown, T. Tony Cai. Interval estimation for a binomial proportion. *Statistical Science*, pages 101–133, 1999.

[12] Pierre Simon marquis de Laplace. Theorie analytique des probabilites. page 283, 1812.

[13] Eric Miller. An introduction to the resource description framework. *Bulletin of the American Society for Information Science and Technology*, pages 15 – 19, 31 January 2005.

[14] Robert G. Newcombe. Two-sided confidence intervals for the single proportion: comparison of seven methods. *Statistics in Medicine Journal*, pages 857–872, 1998.

[15] Tim Berners-Lee Nigel Shadbolt, Wendy Hall. The semantic web revisited. *IEEE Computer Society*, pages 96 – 101, MAY/JUNE 2006.

[16] A. Passaut O. Sacco. A privacy preference manager for the social semantic web. *Second International Conference on Semantic Personalized Information Management: Retrieval and Recommendation - Vol 781*, pages 42–53, 2011.

[17] Elias P. Papadopoulos, Michalis Diamantaris, Panagiotis Papadopoulos, Thanasis Petsas, Sotiris Ioannidis, and Evangelos P. Markatos. The long-standing privacy debate: Mobile websites vs mobile apps. 2017.

[18] Jeremy Rifkin. *The Zero Marginal Cost Society: The Internet of Things, the Collaborative Commons, and the Eclipse of Capitalism*. St. Martin's Griffin, July 7, 2015.

[19] Cambridge Semantics. Introduction to the semantic web. https://www.cambridgesemantics.com. Last accessed January 2019.

[20] FABRIZIO SESTINI. Collective awareness platforms: Engines for sustainability and ethics. *Technology and Society Magazine, IEEE*, pages 54–62, 2012.

[21] Lisa Sullivan. Confidence interval. *Boston University School of Public Health*, 27-10-2017.

[22] Blazegraph Team. Sparql query language for rdf. https://wiki.blazegraph.com/wiki/index.php/About_Blazegraph, 27 January 2016. Last accessed January 2019.

[23] Techopedia. Semantic web. https://www.techopedia.com/definition/27961/semantic-web. Last accessed January 2019.

[24] Ora Lassila Tim Berners-Lee, James Hendler. *The Semantic Web*. Scientific American, 2001.

[25] W3Techs. Owl web ontology language overview. https://www.w3.org/TR/2004/REC-owl-features-20040210/, 2010. Last accessed January 2019.

[26] Sean Wallis. Binomial confidence intervals and contingency tests: mathematical fundamentals and the evaluation of alternative methods. *Journal of Quantitative Linguistics*, pages 178–208, 2013.

[27] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, pages 209–212, 1927.