



**ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΙΡΑΙΩΣ**

MSC Advanced Information Systems Advanced Software Development Technologies

“Εφαρμογή Διαμοιρασμού Ενέργειας Μπαταρίας Κινητών μέσω
Crowdsourcing”

By
Spyridon Desyllas

Advised by lecturer Efthimios Alepis

University of Piraeus



Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Εφαρμογή Διαμοιρασμού Ενέργειας Μπαταρίας Κινητών μέσω Crowdsourcing Mobile Phones Battery Energy Share through Crowdsourcing
Όνοματεπώνυμο Φοιτητή	Σπυρίδων Δεσύλλας
Πατρώνυμο	Στυλιανός
Αριθμός Μητρώου	ΜΠΣΠ/ 14021
Επιβλέπων	Αλέξης Ευθύμιος, Επίκουρος Καθηγητής

Ημερομηνία Παράδοσης **Φεβρουάριος 2018**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)

(υπογραφή)

(υπογραφή)

Αλέπης Ευθύμιος Επίκουρος
Καθηγητής

Βίρβου Μαρία
Καθηγητής

Πατσάκης Κων.
Επίκουρος Καθηγητής

1 Introduction	3
1 Wireless Charging	4
1.1 What is wireless charging	5
1.2 What technology is available today	5
1.3 Qi Wireless Standard	6
1.4 The future of wireless charging: Limitations no more	7
2 Crowdsourcing	8
2.1 Crowdsourcing methods	9
2.2 Crowdsourcing based on criteria points	13
2.2.1 Criteria based on user behaviour	15
2.2.2 Criteria based on geolocation	17
2.2.2 Criteria based on battery level	21
2.3 Summary of points that construct the algorithm	22
3 Technical Implementation	23
3.1 A Cross Platform Mobile Framework	25
3.2 Architecture Design	29
3.2.1 REST API	29
3.2.2 NoSQL Database Engine	32
3.2.3 Architecture and Deployment	35
3.3 C# with .Net Framework and Xamarin.	36
3.4 MongoDB JSON Documents	38
3.4.1 Javascript Object Notation (JSON)	38
3.4.2 Object Serialization	41
3.5 MongoDB geospatial Queries	44
3.5.1 Nearest Friends	44
3.5.2 Range Friends	45
3.5.3 Implementation in MongoDB	46
3.5.3.1 Nearest Friends Implementation	46
3.5.3.2 Range Friends Implementation.	50
3.6 Cloud technology for remote accessible data storage.	53
3.7 Source Code Repositories	57
4. Testing And Evaluation	58
4.1 Unit Testing framework and Test Driven Development.	59
5. Application Demonstration	64
5.1 Android Activities	64
5.2 Main Activity	64
5.3 Register Device Activity	66
5.4 Nearest Devices Activity	67

Εισαγωγή

Έχετε ποτέ βρεθεί στη θέση να μην έχετε μπαταρία στο smartphone σας;

Το να βρεθεί μια πρίζα ή ένας φορτιστής, ή ακόμα και ένα powerbank είναι συνήθως δύσκολο.

Σκεφτήκατε ποτέ ότι μπορείτε να συναντήσετε ένα περαστικό ή κάποιο άτομο που κάθεται κοντά σας το οποίο μπορεί να σας δώσει ένα ποσοστό της ενέργειας του smartphone του σε αντάλλαγμα να κάνετε το ίδιο και για κάποιον άλλο που μπορεί να βρίσκεται στη θέση σας στο μέλλον;

Αυτή η εργασία διατριβής εξαρτάται από το crowdsourcing και την εύρεση ενός επαρκούς αριθμού χρηστών για να συμβάλει σε ένα δίκτυο που βασίζεται στην ανταλλαγή εφεδρικής ενέργειας των κινητών τηλεφώνων. Πρόκειται κυρίως για ένα έργο κινητής εφαρμογής, ωστόσο η εφαρμογή εξαρτάται και από την διαθέσιμη τεχνολογία υλικού (μεταφορά ενσύρματης ή ασύρματης ενέργειας από μπαταρίες, κινητές συσκευές που υποστηρίζουν μια τέτοια επιλογή).

1 Introduction

Have you ever been in the position of having no battery in your latest technology smartphone? Both finding a power outlet and a charger, or even a powerbank is usually difficult. Have you ever considered the option that a passer-by or a person sitting near you might not mind giving you a percentage of his/her smartphone's energy in exchange of you doing the same for someone else who might be in your position in the future?

This thesis work depends on crowdsourcing and finding a sufficient number of peers to contribute to a network based on exchanging spare energy of mobile phones. This is mostly a mobile app project, however, the implementation depends also on the available hardware technology (wired or wireless battery energy transfer, mobile devices supporting such an option).

1 Wireless Charging

The concept of wireless charging has been under development for so many years as it can solve many problems in today's smart phone devices. The smart phone devices are becoming more powerful with horsepower equivalent of a desktop computer. This processing power demands tremendous levels of energy consumption and in the field of mobile technology the energy storage capacity is very limited. Over the years the processing and graphical power of these devices has been evolved but the battery technology to support this power is not par with the technological advancements of computer power. It is clear enough that using our mobile devices wherever and whenever we want to has been shackled by the limitations of energy. Every mobile device user should carry a cable charger with the hope of finding a power bank or a charging docking station in every public place. This is very difficult to do and is something that limits the benefits that we can get from the smart devices as these devices are meant to improve our lives instead of adding technical barriers. This is the reason why many scientists are focused on delivering wireless charging technology across mobile devices, not only this will eliminate the need of extra relation to power banks and accessories but also it will make the life of the daily users much easier knowing that their mobile device can be used on demand in under any situation.

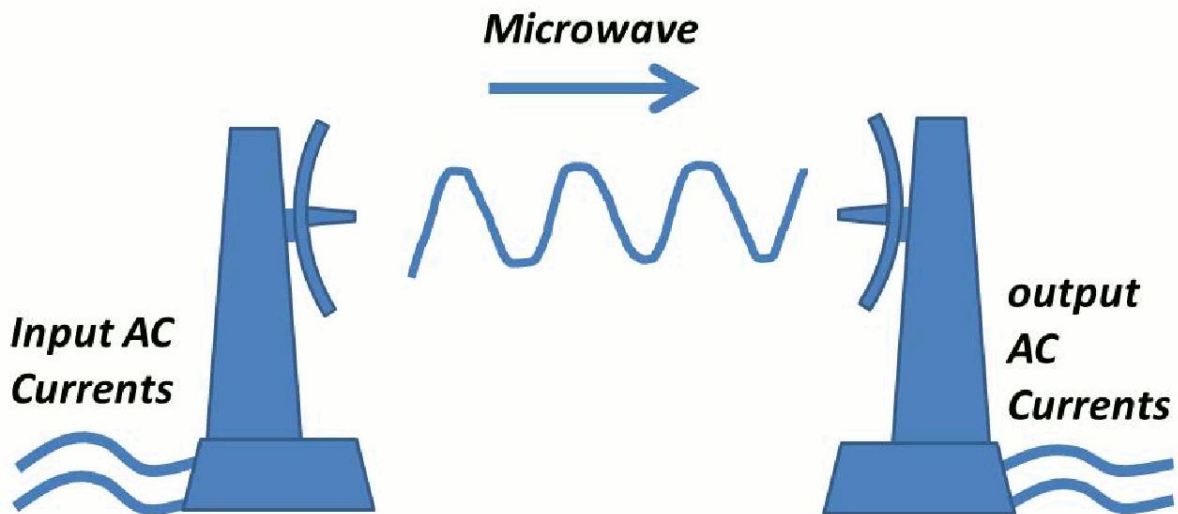


Fig 1.a , Representation of wireless energy transfer

The concept in theory is simple. We need a transmitter and a receiver of energy, the energy in our case is electricity, electrical currents exported by an AC transmitter and received by an AC receiver. In our case the transmitter and the receiver are both batteries residing in mobile smart devices.

1.1 What is wireless charging

Inductive charging (also known as wireless charging) uses an electromagnetic field to transfer energy between two objects through electromagnetic induction. This is usually done with a charging station. Energy is sent through an inductive coupling to an electrical device, which can then use that energy to charge batteries or run the device.

Induction chargers use an induction coil to create an alternating electromagnetic field from within a charging base, and a second induction coil in the portable device takes power from the electromagnetic field and converts it back into electric current to charge the battery. The two induction coils in proximity combine to form an electrical transformer.[1][2] Greater distances between sender and receiver coils can be achieved when the inductive charging system uses resonant inductive coupling. Recent improvements to this resonant system include using a movable transmission coil (i.e. mounted on an elevating platform or arm) and the use of other materials for the receiver coil made of silver plated copper or sometimes aluminium to minimize weight and decrease resistance.

1.2 What technology is available today

Wireless power for mobile phones has been a long time coming. Especially outside the US, where it got a head start thanks to telecoms operators backing it in the early days. Things have certainly picked up in the past year, with Samsung getting off the fence and including it as a standard, rather than an option.

Additionally, wireless products that will become available in the near future will be more powerful, and versatile, offering an even better user experience. We'll see even more charging stations at coffee shops, restaurants and bars. Keeping a smartphone charged will just be easier all round. There are various methods and different standards for wireless charging which we can see in the following table.

	WPC or Qi (Wireless Power Consortium)	PMA (Power Matters Alliance)	A4WP (Alliance for Wireless Power)
Established	2008, Qi was first wireless charging standard	2012, Procter & Gamble and Powermat	2012 by Samsung and Qualcomm
Technology	Inductive charging, 100–205kHz; coil distance 5mm;	Inductive charging, 277–357kHz; similar to Qi	Resonant charging, loosely coupled; serious emission issues remain.
Markets	Qi has widest global use; Over 500 products, more than 60 mobile phones	Tight competition with Qi, gaining ground, 100,000 Powermats at Starbucks,	A4WP and PWA merged, no product available
Members & companies	Samsung, LG, HTC, TI, Panasonic, Sony, Nokia, Motorola, Philips, Verizon, BMW, Audi, Daimler, VW Porsche, Toyota, Jeep	Powermat, Samsung, LG, TDK, TI, AT&T, Duracell, WiTricity, Starbucks Teavana, Huawei, FCC, Energy Star, Flextronics	Qualcomm, TediaTek, Intel, LG, HTC, Samsung, Deutsche Telecom. No commercial products

Table 1.2.a Recognized standards for wireless charging.

1.3 Qi Wireless Standard

Modern wireless charging follows a complex handshake to identify the device to be charged. When placing a device onto a charge mat, the change in capacitance or resonance senses its presence. The mat then transmits a burst signal; the qualified device awakens and responds by providing identification and signal strength status. The signal quality is often also used to improve the positioning of the receiver or enhance magnetic coupling between mat and receiver.

The charge mat only transmits power when a valid object is recognized, which occurs when the receiver fulfills the protocol as defined by one of the interoperability standards. During charging, the receiver sends control error signals to adjust the power level. Upon full charge or when removing the load, the mat switches to standby.

Transmit and receive coils are shielded to obtain good coupling and to reduce stray radiation. Some charge mats use a free moving transmit coil that seeks the object placed for best coupling, others systems feature multiple transmit coils and engage those in close proximity with the object.

WPC calls the transmitter the TX Controller, or Base Station, and the receiver on the mobile device the RX Controller, or Power Receiver. There is a resemblance to a transformer with a primary and secondary coil.

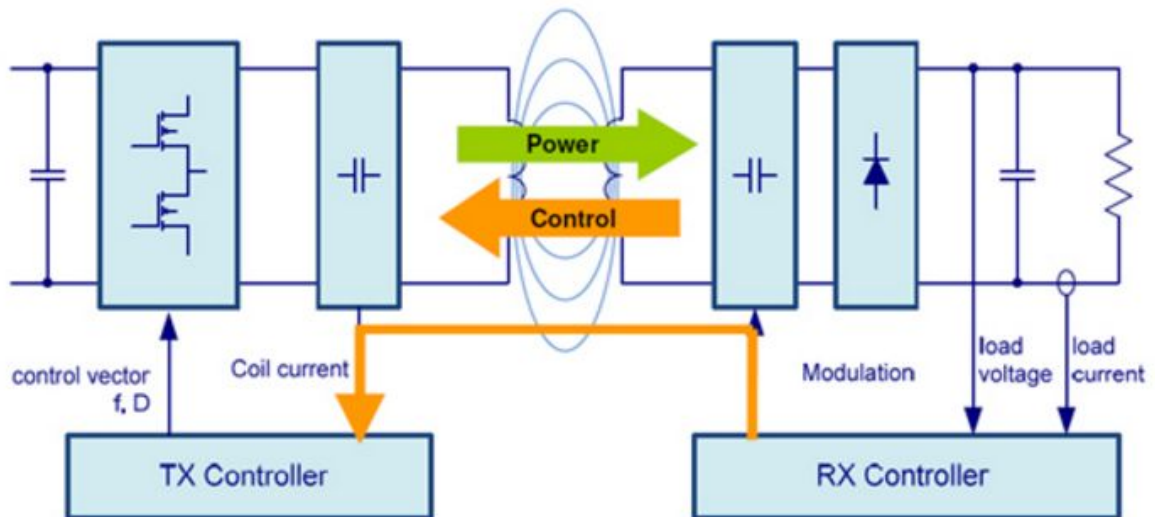


Figure 2: Overview of Qi wireless charging system.

Several systems are competing that may not be compatible. The three most common are Qi, PMA, A4WP.



Fig 1.3.a an overview of a Qi wireless charging system.

1.4 The future of wireless charging: Limitations no more

In the following years we will see the updated Qi standard in new products and as a standard in a plethora of mobile devices. The new 1.2 version of Qi mean the low power found in first generation chargers will soon triple to 15w. Meaning the charging time for a smartphone will drop from three hours to just 80 minutes.

The original Qi products will still be compatible, but will only operate at the original speed. The original products also required fairly precise placement of the phone for the charge to work, but multi-coil technology now allows some freedom of position.

Backwards compatibility is also a very important feature, as there are already lots of Qi enabled devices out there. That's why phone manufacturers are very likely to stick to the Qi charging standard with new devices, some may add AirFuel as well, but Qi is guaranteed.

As a long time proponent of wireless charging, the mobile phone consumers will be thrilled to see the technology so close to being universal. It's going to make people's lives just that little bit better. Running low on phone power will no longer cause panic, with so many charging options.

There will be less messing with cables which currently people don't think as a major issue. But resolving a small problem for millions of people is still a worthwhile endeavour. It is also the first step in creating a completely waterproof phone and a safe compartment where a phone can connect to your car and be charged, all while linking you to your phone safely.

2 Crowdsourcing

The wireless charging technology is the medium of achieving the goal of this thesis project. The vision of this project is to enable users of mobile devices to exchange battery levels with each other's phones based on user preference and previous charging history of a user. This is where we will need to combine wireless charging technology with crowdsourcing algorithms and software. Imaging the following scenario, user A is in a shopping mall and his/her mobile phone is very low on battery. User A needs to be able to search in a pool of mobile phone's batteries to drain some energy from a candidate mobile device. The pool of mobile devices is created by other users located in the same shopping mall or the around perimeter, the prerequisite for a user to be included in the pool is to use the MyCrowdCharger mobile application that registers the mobile geo location data along with user details. A mobile device is considered candidate based on various factors that we will explore in later chapters.

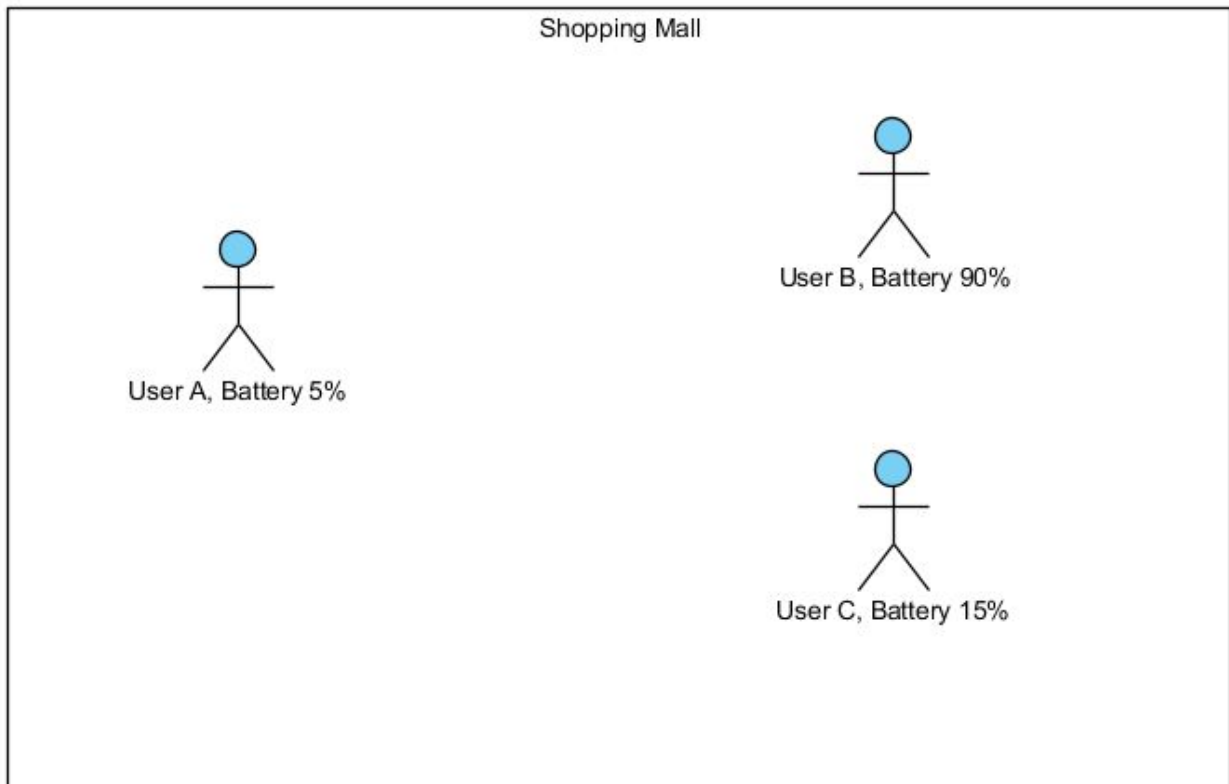


Fig 2.a User case of MyCrowdCharger

As you can see in the diagram our user pool consists of 3 users, all of them located in the shopping mall. User A is in search of some battery energy from the pool, the pool contains two available users, User B and User C with their respective battery levels. It is make sense that the most successful candidate to give some energy will be the user with the most full battery. In our case will be User B. It will not make sense for User C to give energy as User C will be in the same situation as User A and will be in search of a new user to refill his/her battery. User B on the other hand will be glad to give some battery and after the energy transfer he/she will have sustainable battery levels again.

2.1 Crowdsourcing methods

Crowdsourcing like the wireless charging technology is the other aspect that we need to achieve a community of user based charging. Much like the wireless charging it is a whole different scientific field that in combination with physics' wireless charging field will give us the tools to achieve our goal.

Crowdsourcing is a specific sourcing model in which individuals or organizations use contributions from Internet users to obtain needed services or ideas. Crowdsourcing was coined in 2005 as a portmanteau of crowd and outsourcing. This mode of sourcing to divide work between

participants to achieve a cumulative result was already successful before the digital age (i.e., "offline"). Crowdsourcing is distinguished from outsourcing in that the work can come from an undefined public (instead of being commissioned from a specific, named group) and in that crowdsourcing includes a mix of bottom-up and top-down processes. Advantages of using crowdsourcing may include improved costs, speed, quality, flexibility, scalability, or diversity. Crowdsourcing in the form of idea competitions or innovation contests provides a way for organizations to learn beyond what their "base of minds" of employees provides (e.g., LEGO Ideas). Crowdsourcing can also involve rather tedious "microtasks" that are performed in parallel by large, paid crowds (e.g., Amazon Mechanical Turk). Crowdsourcing has also been used for noncommercial work and to develop common goods (e.g., Wikipedia). Arguably the best-known example of crowdsourcing as of 2015 is crowdfunding, the collection of funds from the crowd (e.g., Kickstarter).

Crowdsourcing can either take an explicit or an implicit route. Explicit crowdsourcing lets users work together to evaluate, share, and build different specific tasks, while implicit crowdsourcing means that users solve a problem as a side effect of something else they are doing.

With explicit crowdsourcing, users can evaluate particular items like books or webpages, or share by posting products or items. Users can also build artifacts by providing information and editing other people's work.

Implicit crowdsourcing can take two forms: standalone and piggyback. Standalone allows people to solve problems as a side effect of the task they are actually doing, whereas piggyback takes users' information from a third-party website to gather information.

In his 2013 book, *Crowdsourcing*, Daren C. Brabham puts forth a problem-based typology of crowdsourcing approaches:

Knowledge discovery and management is used for information management problems where an organization mobilizes a crowd to find and assemble information. It is ideal for creating collective resources.

- Distributed human intelligence tasking is used for information management problems where an organization has a set of information in hand and mobilizes a crowd to process or analyze the information. It is ideal for processing large data sets that computers cannot easily do.
- Broadcast search is used for ideation problems where an organization mobilizes a crowd to come up with a solution to a problem that has an objective, provable right answer. It is ideal for scientific problem solving.
- Peer-vetted creative production is used for ideation problems where an organization mobilizes a crowd to come up with a solution to a problem which has an answer that is subjective or dependent on public support. It is ideal for design, aesthetic, or policy problems.

Crowdsourcing can be achieved in many different fields but in more modern days is widely used in internet and mobile.

A specific aspect of crowdsourcing that is very much interesting for our case is the mobile crowdsourcing.

Mobile crowdsourcing involves activities that take place on smartphones or mobile platforms, frequently characterized by GPS technology. This allows for real-time data gathering and gives projects greater reach and accessibility. However, mobile crowdsourcing can lead to an urban bias, as well as safety and privacy concerns.

Crowdsourcing includes three different actors:

- Crowd Sourcers
- Contributors
- Requesters

The crowd sources are the initiative of a crowd resource operation. It might be a business, the government, an organization etc. In our case the crowdsourcer is the actual mobile application that initiates a crowdsourcer in a pool of users to achieve its goal. The goal for the crowdsourcer is to find an eligible mobile phone to give mobile energy and will use crowdsourcing to find which is the best candidate for this purpose.

The contributors are the entities that take part in crowdsourcing, without contributors is impossible to achieve crowdsourcing as it is the main essence to apply it. The contributors are the “crowd” that are contributing data in order to be “sourced” by the crowdsourcer. In our case the contributors are the mobile devices and their users. They are submitting data like geolocation and other criteria to the crowdsourcer in order to be sourced.

The requesters are the entities that requests from the contributors. In our case a requester is a user with a depleted battery on his/her mobile device. The requester is asking the contributors through the crowdsourcer to find the best candidate from the pool of contributors (users).

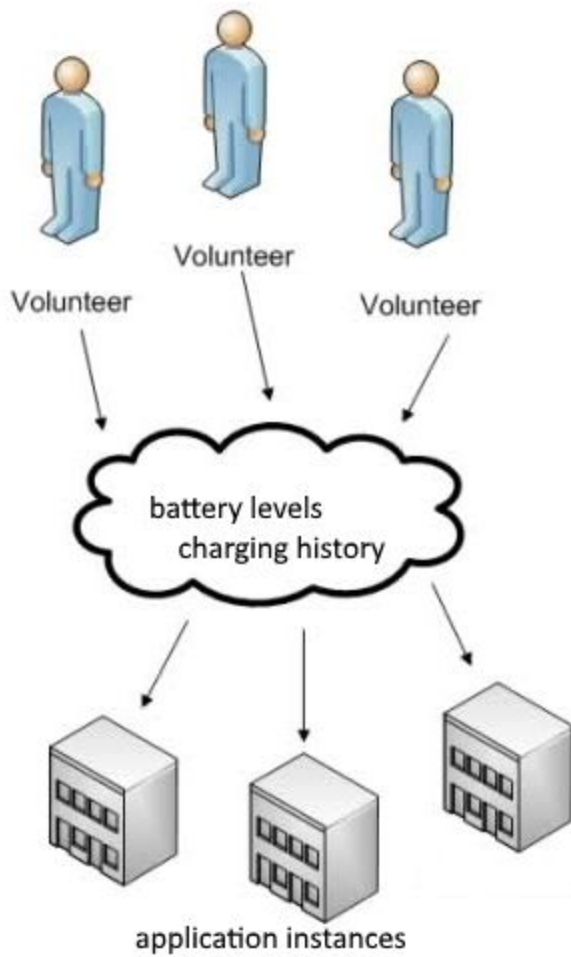


Fig 2.1.a Crowd sourcing in our application

2.2 Crowdsourcing based on criteria points

The crowdsourcing that will be performed to the pool of contributors will need to include a combination of criteria in order to be successful and more accurate. As we noticed before it is obvious that a more successful candidate will need to have plenty of battery to spare but there are actually more criteria that we will need to factor in. A new criteria that we can introduce is, with a scale of 1 to 10, how keen is the user to give energy. This is not easy to measure as we need to construct this criteria by user's history and user behaviour in general. Also to convince the user to give energy we will need to reward this user with the chance of getting energy when he/she needs in return. The basic idea of this project is to drain energy from other users so that we can re-charge our device but also give the users in the pool a reward. A reward can be no other than helping the users in need and "pay back" the energy that we've got from them so that they can benefit in a similar situation.

In the table below we will include all the basic criteria that will construct our level of successful candidate. From now on we will name this criteria "willingness" as it will show to use how eager is the candidate to give the requested amount of energy.

Willingness will be :

$$W = B * T$$

where *B* is Battery Level and *T* is the number of times gave energy in the past.

Willingness will be a point based criteria generated in the following table using the equation above.

Charging history Battery level	1 times	10 times	20 times	100 times
100%	1	10	20	100
90%	0.9	9	18	90
50%	0.6	6	12	60
30%	0.3	3	6	30
15%	0.15	1.5	3	15
5%	0.05	0.5	1	5

Table 2.2.a Willingness points based system

For example if a user has full battery 100% and had given energy 10 times in the past the level of willingness he/she achieves is 10.

$$W = B \cdot T = \frac{100}{100} \cdot 10 = 10$$

If a user with nearly depleted battery at 5% had given battery 100 times in the past the level of willingness he/she has is

$$W = B \cdot T = \frac{5}{100} \cdot 100 = 5$$

Respectively when the first user with full battery has given energy only 1 time in the past the willingness will be:

$$W = B \cdot T = \frac{100}{100} \cdot 1 = 1$$

This may seem odd at the first place but it makes sense, even though the first user has full battery he/she achieves a lower score than the user with 5% battery levels. The reason is that despite the user has full battery and could be a perfect candidate based on the user history he/she only gave battery just once in the past thus making him a not very ideal candidate. In fact

the user with nearly depleted battery has given energy many times in the past, even though he has depleted energy he/she may willing more to give energy.

But what is the point of the Willingness factor. In fact when a user asks for a candidate from the pool of user all candidates are sorted by their Willingness level in a descending manner, this way the user with the best willingness comes first in the list and makes the ideal choice for the requester. If this user for some reason (connection lost, application not responding etc..) is not available then the immediate user in the list becomes the ideal candidate to give energy to the requestor.

2.2.1 Criteria based on user behaviour

In order to make the use of the application appealing to the users, even though to the users that usually give battery instead of asking for energy we use the willingness level for a threshold of the requester as well. Of course the willingness level is used for the sorting of the candidates to make the ideal candidate for energy sharing but also it can be used as a threshold to the requester to limit the amount of energy that he/she can asks. In this way we can create an eco-system of contributors competing with each other to achieve the best score of willingness so that they can ask more energy levels the more they use the application and the more they contribute.

We will introduce a new value for the requester and we will name this value threshold.

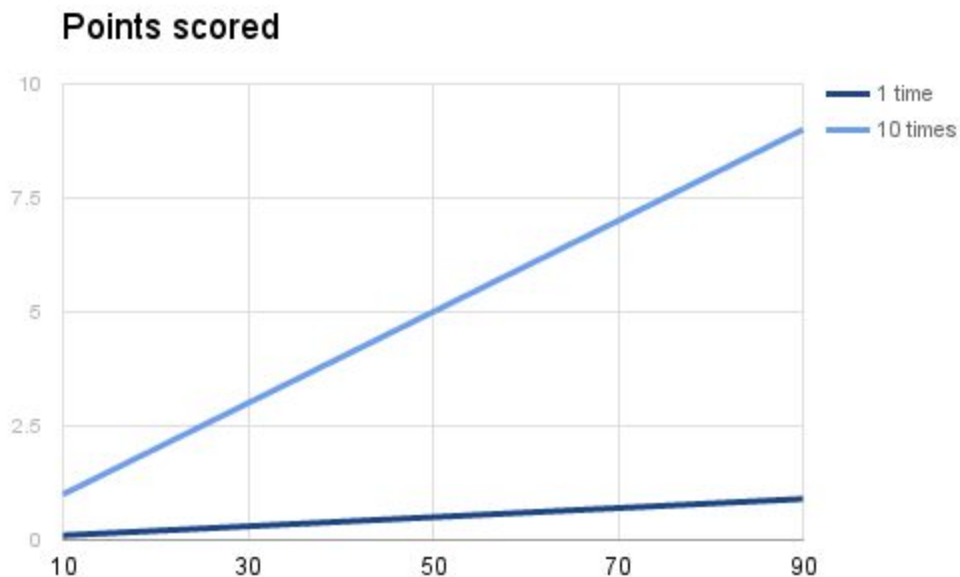
The Threshold categorizes the users into the following categories.

User Category	Energy to drain
New starter (0 - 10 times)	5%
Average (10 - 50 times)	10%
Good (50 - 100 times)	20%
Very Good (≥ 100 times)	30%

Table 2.2.1.a Application user categories

This way we can limit the appetite of each user and we can prevent the abuse of the candidate's energy consumption. A new started can only borrow 10% of energy from a candidate as a new user the application does not yet now if this user will be a valuable contributor in the future, thus we will have to limit the energy drain until there are sufficient information that this specific user not only asks for energy but also is willing to give energy and be a valuable peer that will help the life cycle and the network of our application.

We can now represent these two values in a Cartesian field to gain a better understanding.

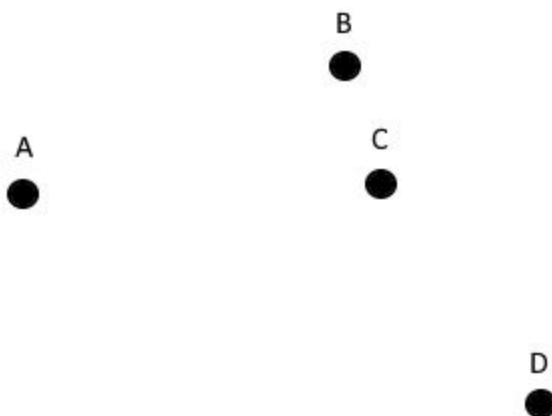


2.2b Graph of willingness, in the x axis is the battery level, in the y axis is the willingness for a user that has given energy 1 time in the past and for another user that has given energy 10 times in the past.

As we can see the user that contributed 10 times in the past will get many more points and will be a more valuable peer to contribute energy. The second user although has used the application only once so even though the battery levels are high he will score far less points in the sorting operation.

2.2.2 Criteria based on geolocation

We introduced the criteria points of the outsourcing operation in regards with User's Battery levels but also with User's behaviour in order to provide a better user experience and a healthier network of peers that consume but also contribute to the network ecosystem. At this point it would be wise to include another factor to our equation. Regardless of the technology that will be used for the wireless charging we will have to consider that the distance of the user's mobile device will always be a major factor in terms of the choice and the final selection of the energy giver and consumer. If the devices are far away to each other is more likely to have some issues during the charging process or maybe even slower charging and bad performance in general.



Picture 2.2.2a Geographical representation of peers

In the above example the User A needs battery levels from all the available users in a specific perimeter. Users B, C and D are holding mobile devices with sustainable levels of battery and are crowdsources extensively but the crowdsourcing mechanism of our application. In this case we need to answer another question to User A. Which is the closest user near him that will be able to transfer these levels of energy more secure and more reliable than the other users. We will need to apply a Geolocation query and more specifically a Nearest Neighbour query to find which users are more close by to user A.

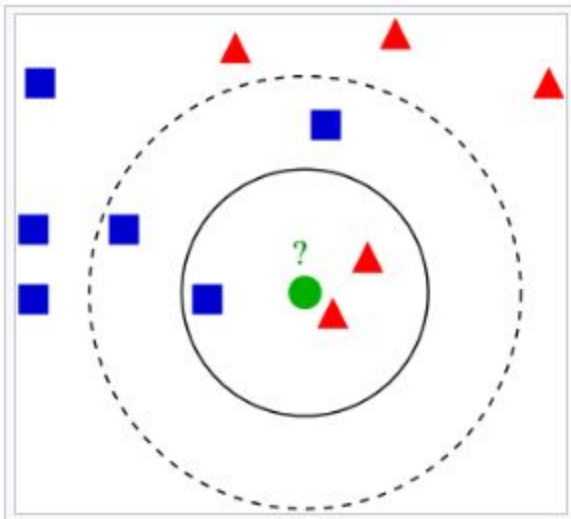
Nearest Neighbour algorithm

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression microarray data, for example, k -NN has also been employed with correlation coefficients such as Pearson and Spearman.[3] Often, the classification accuracy of k -NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number.[4] One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class (or value, in regression problems) of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. K -NN can then be applied to the SOM.



Picture 2.2.2b Example of k -NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ (solid line

circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

Parameter Selection

The best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification,[5] but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques (see hyperparameter optimization). The special case where the class is predicted to be the class of the closest training sample (i.e. when $k = 1$) is called the nearest neighbor algorithm.

The accuracy of the k -NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve classification. A particularly popular[citation needed] approach is the use of evolutionary algorithms to optimize feature scaling. Another popular approach is to scale features by the mutual information of the training data with the training classes. In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids tied votes. One popular way of choosing the empirically optimal k in this setting is via bootstrap method

The 1-nearest neighbour classifier

The most intuitive nearest neighbour type classifier is the one nearest neighbour classifier that assigns a point x to the class of its closest neighbour in the feature space, that is

$$C_n^{1nn}(x) = Y_{(1)}$$

As the size of training data set approaches infinity, the one nearest neighbour classifier guarantees an error rate of no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data).

The weighted nearest neighbour classifier

The k -nearest neighbour classifier can be viewed as assigning the k nearest neighbours a weight $1/k$ and all others 0 weight. This can be generalised to weighted nearest neighbour classifiers. That is, where the i th nearest neighbour is assigned a weight w_{ni} , with $\sum_{i=1}^n w_{ni} = 1$. An analogous result on the strong consistency of weighted nearest neighbour classifiers also holds.

Let C_n^{wmn} denote the weighted nearest classifier with weights $\{w_{ni}\}_{i=1}^n$. Subject to regularity conditions on to class distributions the excess risk has the following asymptotic expansion

$$\mathcal{R}_{\mathcal{R}}(C_n^{wmn}) - \mathcal{R}_{\mathcal{R}}(C^{Bayes}) = (B_1 s_n^2 + B_2 t_n^2) \{1 + o(1)\},$$

for constants B_1 and B_2 where $s_n^2 = \sum_{i=1}^n w_{ni}^2$ and $t_n = n^{-2/d} \sum_{i=1}^n w_{ni} \{i^{1+2/d} - (i-1)^{1+2/d}\}$.

The optimal weighting scheme $\{w_{ni}^*\}_{i=1}^n$, that balances the two terms in the display above, is given as follows: set $k^* = \lfloor Bn^{\frac{4}{d+4}} \rfloor$,

$$w_{ni}^* = \frac{1}{k^*} \left[1 + \frac{d}{2} - \frac{d}{2k^{*2/d}} \{i^{1+2/d} - (i-1)^{1+2/d}\} \right] \text{ for } i = 1, 2, \dots, k^* \text{ and}$$

$$w_{ni}^* = 0 \text{ for } i = k^* + 1, \dots, n.$$

Picture 2.2.2c The Weighted nearest neighbour equation

2.2.2 Criteria based on battery level

We investigated the criteria points based on the user scoring willingness to donate energy as well as the points scored by the nearest neighbour algorithm based on geolocation data. The third and final factor for our scoring points will be the battery level of the mobile devices of the candidates. It is logical to include this factor in the equation as it would be meaningless for a candidate to score high in the board and will to give energy if the user's device itself has a depleted battery.

2.3 Summary of points that construct the algorithm

In a nutshell the algorithm of sorting the candidates that can donate battery energy levels will be the following

```
candidates.OrderBy(x => x.Distance).  
    .ThenBy(x => x.Willingness)  
    .ThenBy(x => x.BatteryLevel);
```

2.3 Algorithm to sort candidates using C# lambda expressions

For our algorithm we are using C# language code in with lambda expressions.

Each code snippet and algorithm in this project will be represented in C# as this will be the language of our technical implementation in the mobile devices that we will talk about in the next chapter.

Lambda expression is an anonymous function that you can use to create delegates or expression tree types. By using lambda expressions, you can write local functions that can be passed as arguments or returned as the value of function calls. Lambda expressions are particularly helpful for writing LINQ query expressions.

To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator =>, and you put the expression or statement block on the other side. For example, the lambda expression `x => x * x` specifies a parameter that's named `x` and returns the value of `x` squared.

Lambda Expressions (C# Programming Guide) :

<https://msdn.microsoft.com/en-us/library/bb397687.aspx>

3 Technical Implementation

In order to develop this project numerous and different frameworks and tools can be considered. At the time this thesis project is written the following frameworks are available for the needs of our project.

Framework	Programming Language	Mobile OS	Native or Web
Android Studio	Java	Android	Native
XCode	Swift	iOS	Native
Ionic	HTML / CSS / Javascript	Cross Platform	Web
Xamarin	C#	Cross Platform	Native

Appendix 3.a Available mobile application development technologies

There are some other various frameworks to develop web or native application but we will not go in depth as it is out of scope of this project.

Although all frameworks can cover the needs of our application we will choose Xamarin, the reasons are :

- C# is a powerful language implementing all the modern programming features that we expect from a object oriented language
- Use one codebase but build the application both for Android iOS and Windows phone
- Use Visual Studio powerful IDE capabilities to build Cross Platform applications.
- Use .Net Frameworks features to build an application according to SOLID programming principles.

The aim is to use C# and Xamarin to create a robust application using SOLID principles. SOLID principles are five basic principles of object-oriented programming and design. The intention is that these principles, when applied together, will make it more likely that a programmer will create a system that is easy to maintain and extend over time. The principles of SOLID are guidelines that can be applied while working on software to remove code smells by providing a framework through which the programmer may refactor the software's source code until it is both legible and extensible. It is part of an overall strategy of agile and Adaptive Software Development.

Initial	Stand s for	Concept
S	SRP	<p>Single responsibility principle</p> <p>a class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)</p>
O	OCP	<p>Open/closed principle</p> <p>“software entities ... should be open for extension, but closed for modification.”</p>
L	LSP	<p>Liskov substitution principle</p> <p>“objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.” See also design by contract.</p>
I	ISP	<p>Interface segregation principle</p> <p>“many client-specific interfaces are better than one general-purpose interface.”^[8]</p>
D	DIP	<p>Dependency inversion principle</p> <p>one should “depend upon abstractions, [not] concretions.”^[8]</p>

Appendix 3.b SOLID Principles of object oriented programming.




At this point it would be wise to say that SOLID principles are not exclusive to C# and Xamarin as they are more rules and methodologies rather than technical features but C# programming language features enable developers to apply SOLID principles in a more guided,

intuitive and easier manner. This is a personal opinion of the writer of this thesis and does not derive from any research or evaluation of other technologies.

3.1 A Cross Platform Mobile Framework

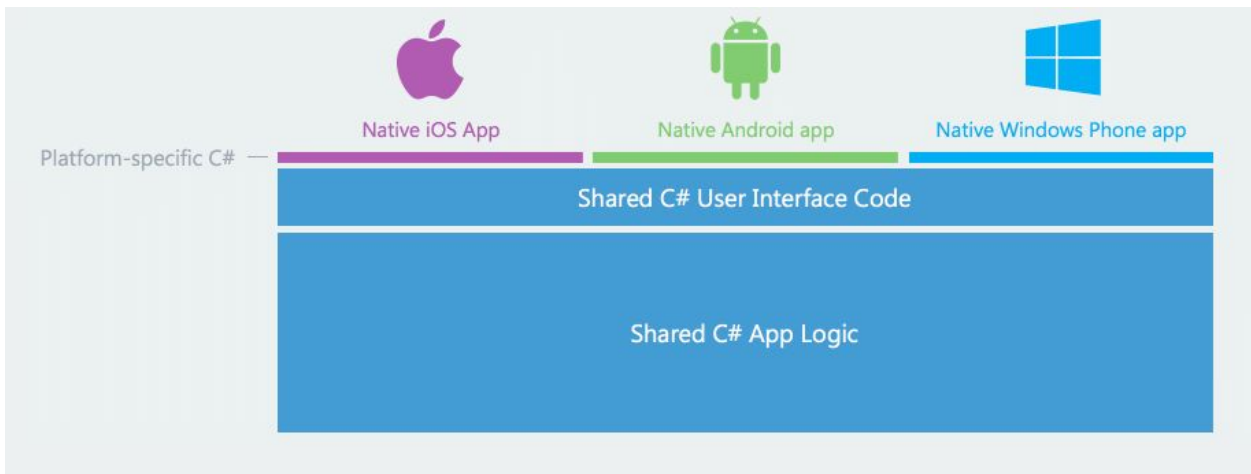
As we mentioned before our intention is to create an application using a single programming language and a single framework and use one and only one codebase for every available mobile OS (Cross Platform). Xamarin framework will provide all the tools that we will need in order to achieve the elasticity and rapid development of this project.

The application will look and feel native because it will benefit from Native User Interfaces, Native API Access and Native Performance.

Xamarin Features	Description
 Native User Interfaces	Xamarin apps are built with standard, native user interface controls. Apps not only look the way the end user expects, they behave that way too.
 Native API Access	Xamarin apps have access to the full spectrum of functionality exposed by the underlying platform and device, including platform-specific capabilities like iBeacons and Android Fragments.
 Native Performance	Xamarin apps leverage platform-specific hardware acceleration, and are compiled for native performance. This can't be achieved with solutions that interpret code at runtime.

Appendix 3.1.1 Xamarin Features

The application will use a Shared Logic layer written in C# that will be shared to a middleware layer build on .Net framework. The middleware framework is responsible to share the components of the application to the specific device OS providing a common interface code.

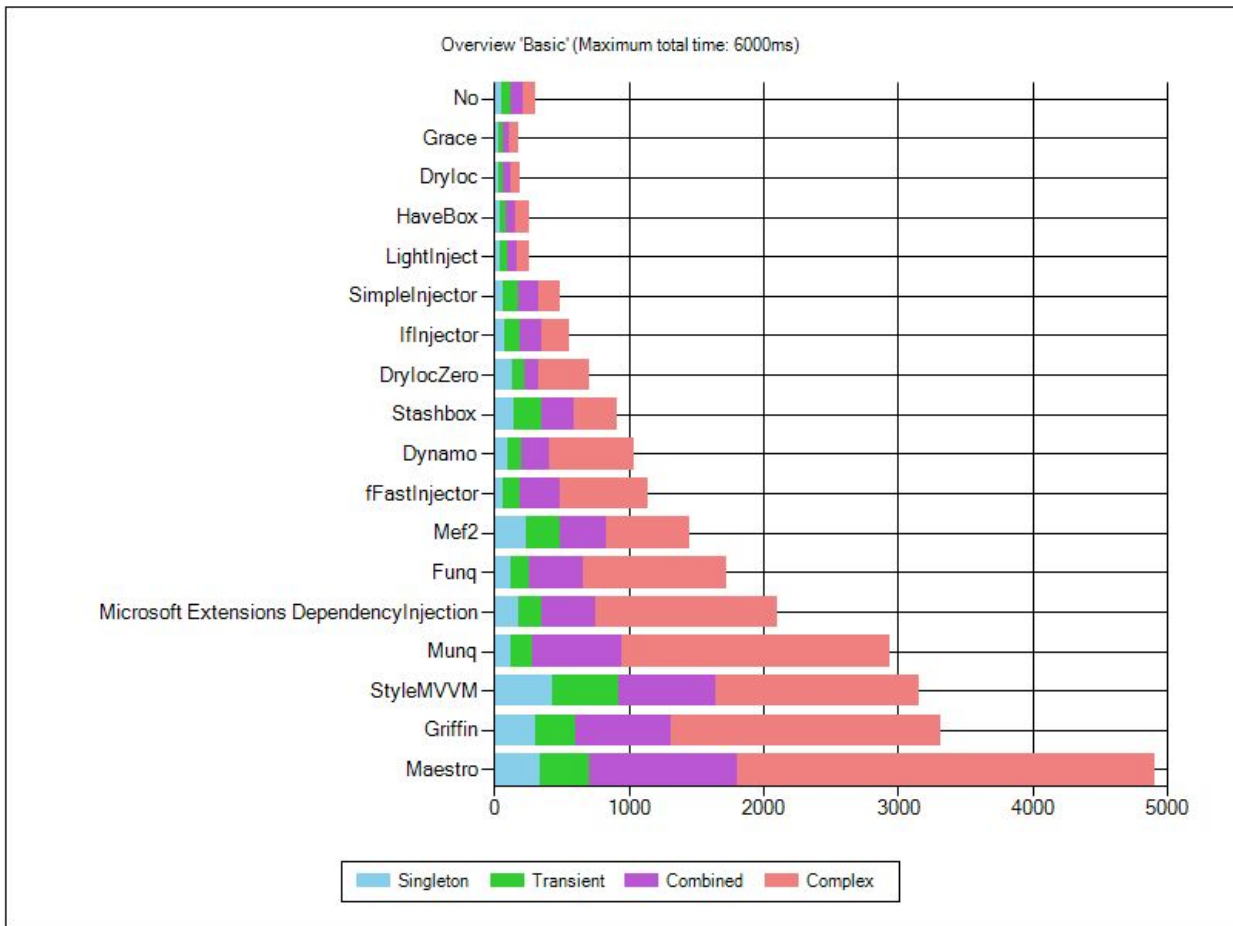


Pic 3.1.2 Architectural layers of our common interface approach

The shared C# Application logic consists of all the components that provide the logic to our application. All logic components are testable in isolation while mocking out all of their dependencies. For this reason all the components of the shared layer are written using all of the SOLID principles providing a separation of control of the dependencies of the classes by injecting them in the constructor using a dependency injection framework.

The Shared C# User Interface Code is provided by Xamarin and is acting as a middleware between the application logic that we will write and the specific code that is needed by each device operation system.

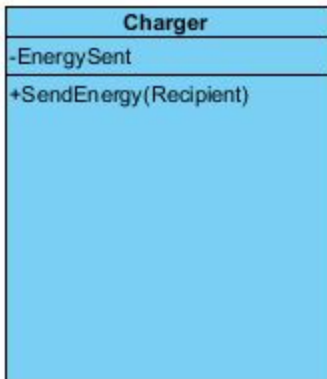
Previously we mention dependency injection briefly but it is time now to go in depth into this programming technique as it is essential for the success of this project. We want the application logic to be hardware agnostic in a way that the logic will exist and function the same way regardless of the medium that is being used for wireless charging all other hardware specific features. Using dependency inversion we can invert the dependencies of the classes of the application logic to a third party framework. The reasons are that it will be very easy to test our application logic while mocking the hardware dependencies, the dependency injection framework will be responsible to inject any implementation to the logic, fake or real the application will not be affected whatsoever. We will choose Simple Injector as our third party injector for various reasons but the main reason is the performance comparing to other similar frameworks.



Appendix 3.1.3 Comparison of Dependency Injection frameworks performance

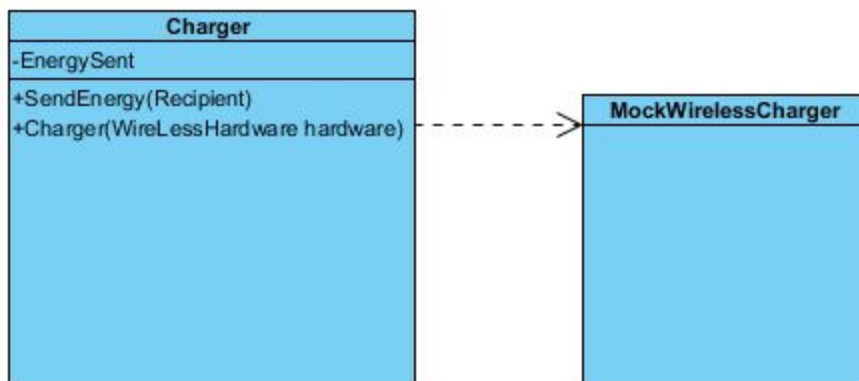
Simple Injector is responsible to inject the dependencies to the application according to a configuration storage that contains all the implementations for each logic component.

An an example imagine the component that will be used for the actual charging, lets name this component *MyCrowdCharger.Battery.Charger*, this component is a class with the following methods and properties.



Pic 3.1.3 Charger Class diagram

The Charger class has one method `SendEnergy(Recipient)` and one property `EnergySent`. The `SendEnergy` method simply sends energy levels to a recipient while the `EnergySent` property responds back the total energy that this charger has sent to any recipients. While this class will contain the logic for the energy sending mechanism we don't want to couple this logic with any actual implementation of a real hardware wireless charging. By decoupling the logic and the actual hardware implementation it will be easy for our application to adapt different charging mechanism in the future, or use a mocking charging mechanism for testing. When the application starts the Charger dependencies will be injected via the class constructor so that during the application runtime the Interface of the Charger will be using an actual implementation of a mock charger or a specific wireless charger that the mobile device provides.



Picture 3.1.4 Charger Injected with a mock wireless charger.

In the example above the Charger class is being injected my SimpleInjector framework with a mock implementation of the Wireless Charger instead of a real one in its constructor. By doing so it is helping us to test this component in isolation writing unit tests and also demonstrate the application in a controlled test environment without the dependency of the physical medium of the charging. In the time being we may use a specific wireless hardware technology but if a better technology is available in the future we want our application to be as adaptive as possible and integrate this new technology with no changes in the logic layer whatsoever.

3.2 Architecture Design

We described our Cross Platform approach so now it is time to dive into the architecture of our application and explain thoroughly each component.

The application is consisting of a native mobile application, a web service layer for data communication with the server and a NoSQL Database instance. The application will perform all the basic CRUD operations via the web services layer which will use noSQL Database engine for data storage in JSON file format. The web service will be implemented as a RESTfull api using simple JSON data object at the Data Transfer Objects (DTOs). Below we will explain the REST architecture as well as the NoSQL Database engines.

3.2.1 REST API

Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations. Other forms of Web service exist, which expose their own arbitrary sets of operations such as WSDL and SOAP. "Web resources" were first defined on the World Wide Web as documents or files identified by their URLs, but today they have a much more generic and abstract definition encompassing every thing or entity that can be identified, named, addressed or handled, in any way whatsoever, on the Web. In a RESTful Web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or some other defined format. The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources. Using HTTP, as is most common, the kind of operations available include those predefined by the HTTP verbs GET, POST, PUT, DELETE and so on. By making use of a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running.

The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI). The term is intended to evoke an image of how a well-designed Web application behaves: it is a network of Web resources (a virtual state-machine) where the user progresses through the application by selecting links, such as /user/tom, and operations such as GET or DELETE (state transitions), resulting in the next resource (representing the next state of the application) being transferred to the user for their use.

The architectural properties affected by the constraints of the REST architectural style are:

- Performance - component interactions can be the dominant factor in user-perceived performance and network efficiency

- **Scalability** to support large numbers of components and interactions among components. Roy Fielding, one of the principal authors of the HTTP specification, describes REST's effect on scalability as follows:

REST's client–server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—**proxies**, **gateways**, and **firewalls**—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate **cacheability**.

- Simplicity of a Uniform Interface
- Modifiability of components to meet changing needs (even while the application is running)
- Visibility of communication between components by service agents
- Portability of components by moving program code with the data
- **Reliability** is the resistance to failure at the system level in the presence of failures within components, connectors, or data

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
<p>Collection, such as http://api.example.com/resources /</p>	<p>List the URIs and perhaps other details of the collection's members.</p>	<p>Replace the entire collection with another collection.</p>	<p>Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.^[17]</p>	<p>Delete the entire collection.</p>
<p>Element, such as http://api.example.com/resources /item17</p>	<p>Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.</p>	<p>Replace the addressed member of the collection, or if it does not exist, create it.</p>	<p>Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it.^[17]</p>	<p>Delete the addressed member of the collection.</p>

Table 3.2.1.a The following shows how HTTP methods are typically used in a RESTful API

3.2.2 NoSQL Database Engine

A NoSQL (originally referring to "non SQL", "non relational" or "not only SQL") database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century, triggered by the needs of Web 2.0 companies such as Facebook, Google, and Amazon.com. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages.

Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability, partition tolerance, and speed. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages (instead of SQL, for instance the lack of ability to perform ad-hoc joins across tables), lack of standardized interfaces, and huge previous investments in existing relational databases. Most NoSQL stores lack true ACID transactions, although a few databases, such as MarkLogic, Aerospike, FairCom c-treeACE, Google Spanner (though technically a NewSQL database), Symas LMDB, and OrientDB have made them central to their designs. (See ACID and join support.)

Instead, most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes "eventually" (typically within milliseconds) so queries for data might not return updated data immediately or might result in reading data that is not accurate, a problem known as stale reads. Additionally, some NoSQL systems may exhibit lost writes and other forms of data loss. Fortunately, some NoSQL systems provide concepts such as write-ahead logging to avoid data loss. For distributed transaction processing across multiple databases, data consistency is an even bigger challenge that is difficult for both NoSQL and relational databases. Even current relational databases "do not allow referential integrity constraints to span databases. There are few systems that maintain both ACID transactions and X/Open XA standards for distributed transaction processing.

There have been various approaches to classify NoSQL databases, each with different categories and subcategories, some of which overlap. What follows is a basic classification by data model, with examples:

- **Column:** Accumulo, Cassandra, Druid, HBase, Vertica, SAP HANA
- **Document:** Apache CouchDB, ArangoDB, Clusterpoint, Couchbase, DocumentDB, HyperDex, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB

- **Key-value:** Aerospike, ArangoDB, Couchbase, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB
- **Graph:** AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso, Stardog
- **Multi-model:** Alchemy Database, ArangoDB, CortexDB, Couchbase, FoundationDB, InfinityDB, MarkLogic, OrientDB

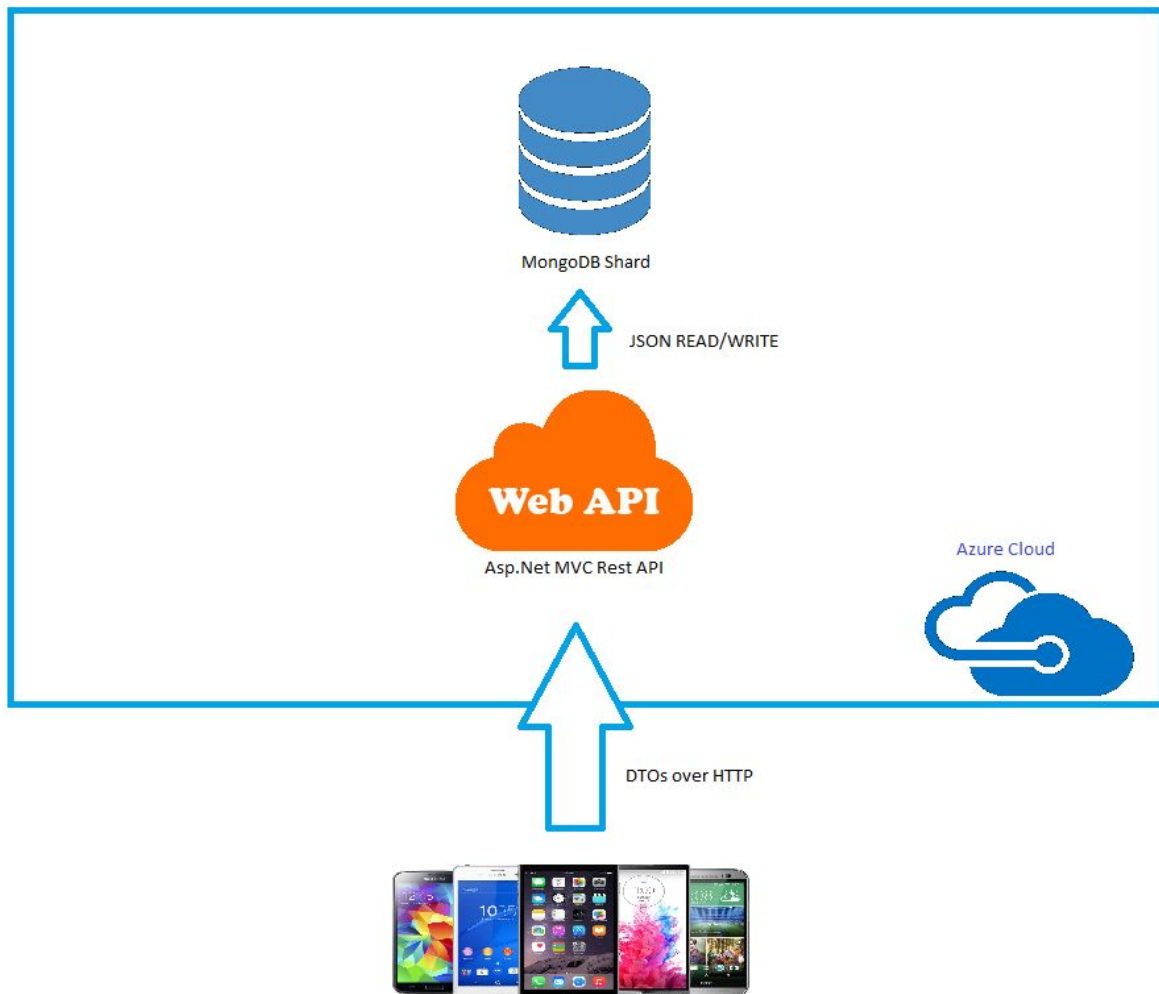
Type	Examples of this type
Key-Value Cache	Coherence, eXtreme Scale, GigaSpaces, GemFire, Hazelcast, Infinispan, JBoss Cache, Memcached, Repcached, Terracotta, Velocity
Key-Value Store	ArangoDB, Flare, Keyspace, RAMCloud, SchemaFree, Hyperdex, Aerospike
Key-Value Store (Eventually-Consistent)	DovetailDB, Oracle NoSQL Database, Dynamo, Riak, Dynamite, MotionDb, Voldemort, SubRecord
Key-Value Store (Ordered)	Actord, FoundationDB, InfinityDB, Lightcloud, LMDB, Luxio, MemcacheDB, NMDB, Scalaris, TokyoTyrant
Data-Structures Server	Redis
Tuple Store	Apache River, Coord, GigaSpaces
Object Database	DB4O, Objectivity/DB, Perst, Shoal, ZopeDB
Document Store	ArangoDB, Clusterpoint, Couchbase, CouchDB, DocumentDB, IBM Domino, MarkLogic, MongoDB, Qizx, RethinkDB, XML-databases
Wide Column Store	BigTable, Cassandra, Druid, HBase, Hypertable, KAI, KDI, OpenNeptune, Qbase

Table 3.2.2.a A more detailed classification of NoSQL databases

In our case we are going to choose a Document Database for our needs and more specifically we will use MongoDB which we will talk about in a later chapter. In MongoDB as well as all document NoSQL databases data is stored in a form of a JSON file that represents a document.

3.2.3 Architecture and Deployment

Our application web service layer and database storage will use Cloud infrastructure for hosting due to the elasticity, performance and reliability that a large scale mobile application needs. The application will use Microsoft's AZURE cloud for deployment and will use Web Application SaaS (Software as a service) for the RESTful web api and IaaS (Platform as a service) for the MongoDB shards.

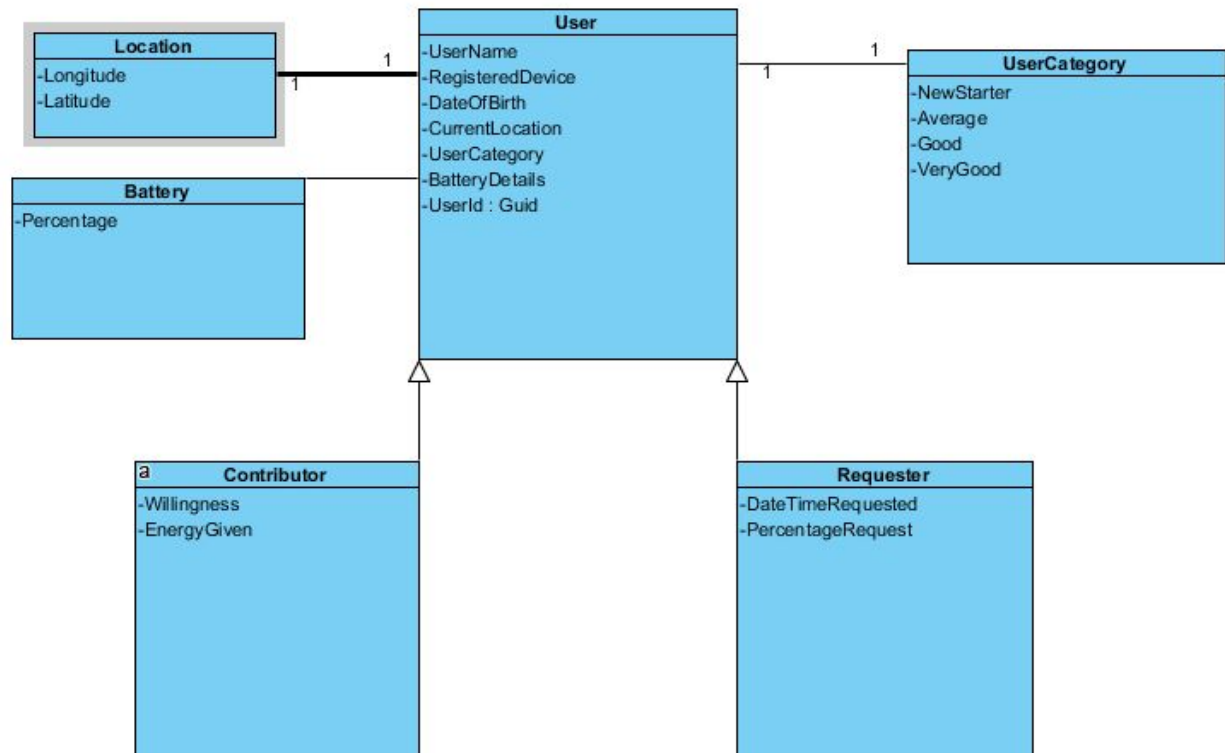


Picture 3.2.3a MyCrowdCharger Deployment Architecture

3.3 C# with .Net Framework and Xamarin.

As we mentioned in the previous chapters our application will be written in C# using .Net framework in the hood in order to be able to use Xamarin's¹ cross platform feature and interoperability between all layers.

In this chapter we will present all the basic entities of our application which will be represented as C# classes.



Picture 3.3a Class diagram of the user entities

¹ "Xamarin." <https://www.xamarin.com/>. Accessed 29 May. 2017.

3.4 MongoDB JSON Documents

For all our storage needs we will use MongoDB's document storage capabilities and utilise JSON file format for our objects. Before transferring the C# class objects from our Xamarin layer we will need to serialize these objects into JSON files and then pass them to the web api layer and towards the MongoDB storage.

3.4.1 Javascript Object Notation (JSON)

In **computing**, **JSON** (canonically pronounced [/ˈdʒeɪsən/ jay-sən](#)² sometimes **JavaScript Object Notation**) is an **open-standard format** that uses **human-readable** text to transmit data objects consisting of **attribute–value pairs**. It is the most common data format used for **asynchronous** browser/server communication, largely replacing **XML**, and is used by **AJAX**.

JSON is a **language-independent** data format. It derives from **JavaScript**, but as of 2017 many **programming languages** include code to generate and **parse** JSON-format data. The official Internet **media type** for JSON is `application/json`. JSON filenames use the extension `.json`.

JSON's basic data types are:

- **Number**: a signed decimal number that may contain a fractional part and may use exponential **E notation**, but cannot include non-numbers like **NaN**. The format makes no distinction between integer and floating-point. JavaScript uses a **double-precision floating-point format** for all its numeric values, but other languages implementing JSON may encode numbers differently.
- **String**: a sequence of zero or more **Unicode** characters. Strings are delimited with double-quotation marks and support a backslash **escaping** syntax.
- **Boolean**: either of the values `true` or `false`
- **Array**: an **ordered list** of zero or more values, each of which may be of any type. Arrays use **square bracket** notation with elements being comma-separated.
- **Object**: an unordered collection of name/value pairs where the names (also called keys) are strings. Since objects are intended to represent **associative arrays**,^[12] it is recommended, though not required,^[13] that each key is unique within an object. Objects are delimited with **curly brackets** and use commas to separate each pair, while within each pair the colon `:` character separates the key or name from its value.
- **null**: An empty value, using the word `null`

Limited **whitespace** is allowed and ignored around or between syntactic elements (values and punctuation, but not within a string value). Only four specific characters are considered whitespace for this purpose: space, horizontal tab, line feed, and carriage return. In particular, the **byte order mark** must not be generated by a conforming implementation (though it may be accepted when parsing JSON). JSON does not provide any syntax for comments.

² "JSON." <http://www.json.org/>. Accessed 29 May. 2017.

Early versions of JSON (such as specified by [RFC 4627](#)) required that a valid JSON "document" must consist of only an object or an array type, which could contain other types within them. This restriction was removed starting with [RFC 7158](#), so that a JSON document may consist entirely of any possible JSON typed value.

The following example shows a possible JSON representation describing a person.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Picture 4.3.1 A Basic JSON example to represent a person's information

3.4.2 Object Serialization

As we mentioned the application's Xamarin layer will serialize all the C# DTO object to JSON before passing them to the web service. Doing that the application decouples all the complexity of the objects while maintaining minimal size for efficient transfer.

For our serialization purposes we are going to use a JSON serialize/deserialize library for .Net framework. After evaluation of different libraries the choice is to use Newtonsoft's JSON.Net to perform the object serializations to JSON and deserialization from JSON back to C# objects and classes.

```
Product product = new Product();
product.Name = "Apple";
product.Expiry = new DateTime(2008, 12, 28);
product.Sizes = new string[] { "Small" };

string json = JsonConvert.SerializeObject(product);
// {
//   "Name": "Apple",
//   "Expiry": "2008-12-28T00:00:00",
//   "Sizes": [
//     "Small"
//   ]
// }
```

Serialize JSON

Picture 3.4.2.a JSON.Net serialization Example

```
string json = @"{
  'Name': 'Bad Boys',
  'ReleaseDate': '1995-4-7T00:00:00',
  'Genres': [
    'Action',
    'Comedy'
  ]
}";

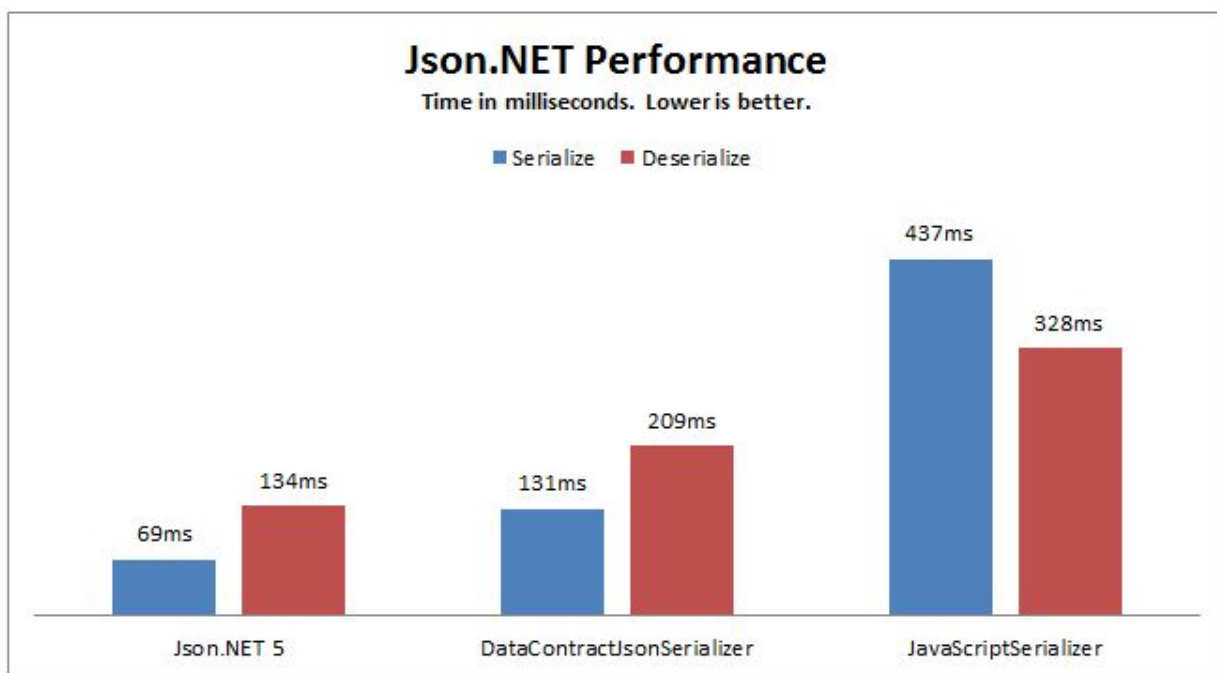
Movie m = JsonConvert.DeserializeObject<Movie>(json);

string name = m.Name;
// Bad Boys
```

Deserialize JSON

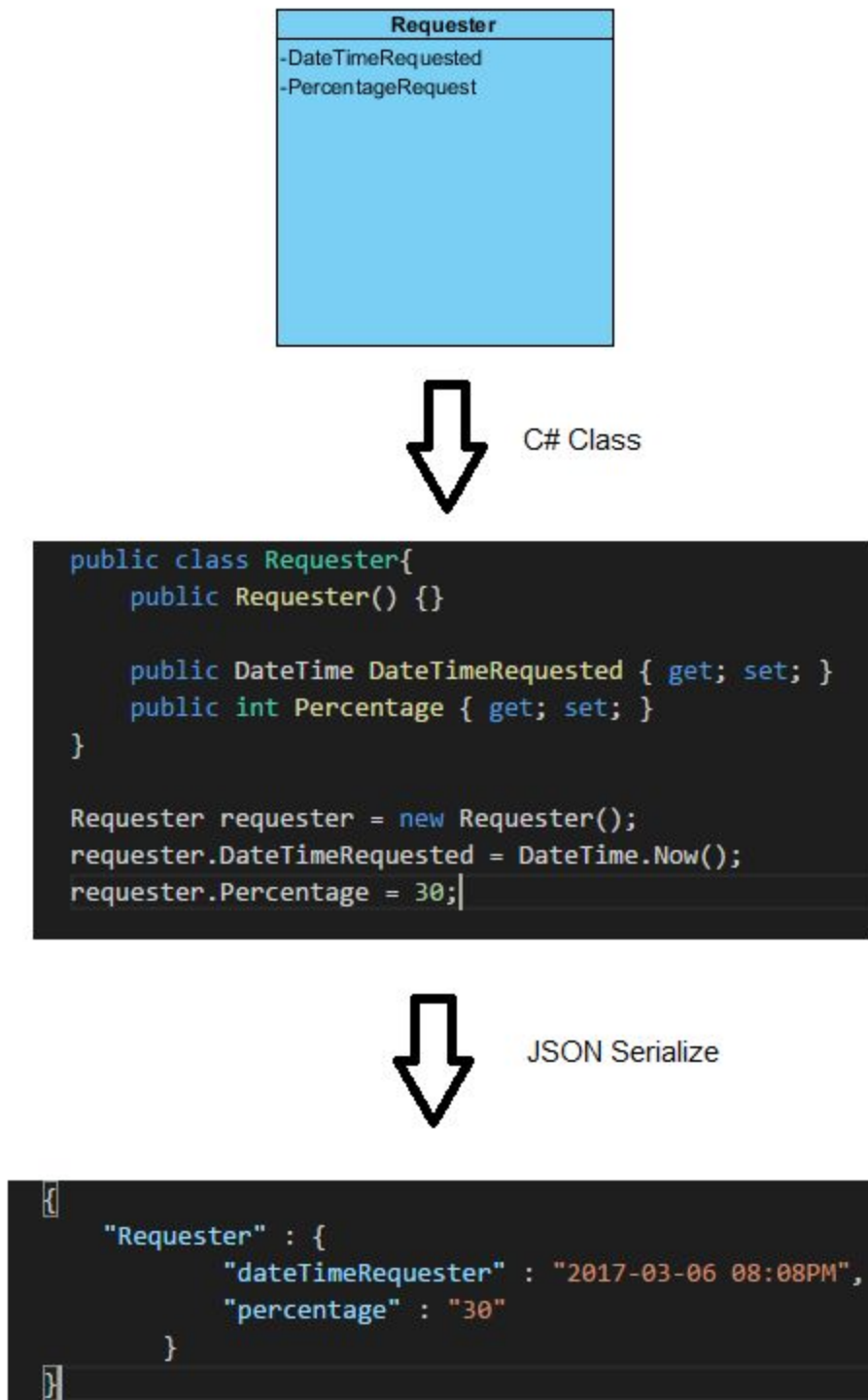
Picture 3.4.2b JSON.net deserialization example

The evaluation and experimentation of different libraries including .Net out of the box json serializer and Javascript serializer is presented in the following diagram:



Picture 3.4.2c Json.Net Performance vs DataContract.JsonSerialize vs JavaScriptSerializer

Json.NET performed significantly better than other known converted thus justified our choice for using it instead of the Microsoft .Net out of the box library.



Picture 3.4.2d Representation of Object, C# class and serialized JSON object.

3.5 MongoDB geospatial Queries

The backend of our application needs to support basic Geospatial queries in order to support the Geosocial aspect that our business logic needs, that means that the framework should be able to respond to queries such as NearestNeighbour, WithinRadius and various other geo location queries.

In our research of selection of such a database technology we needed to reassure that we will be able to support the following geo-location primitive queries³.

- Nearest Friends (NF)
- Range Friends (RF)
- Nearest Star Group (NSG)

The theory of these queries and the algorithms behind them are very well covered by the work of Nikos Papadias et al A General Framework for Geo-Social Query Processing. We will need to implement these queries in our backend system or at least of portion of them by using MongoDB's geospatial capabilities and apply the social aspect on top of them. The friends in the case of our project will be the "friend" users that we have exchange battery levels in the past and are well known users to us.

3.5.1 Nearest Friends

NF returns the k friends of user u that are closest to location q in ascending distance.

There are three different variations that we can implement to achieve the Nearest Friends geosocial query which we will explore in the following figure:

³ "A General Framework for Geo-Social Query Processing."
<http://www.vldb.org/pvldb/vol6/p913-papadopoulos.pdf>. Accessed 29 May. 2017.

Input: User u , location q , positive integer k

Output: Result set R

/ Algorithm 1 (NF₁) */*

1. $F = \text{GetFriends}(u)$, $R = \emptyset$
2. **For** each user $u_i \in F$, compute $\text{GetUserLocation}(u_i)$
3. Sort F in ascending order of $\|q, u_i\|$
4. Insert the first k entries of F into R
5. **Return** R

/ Algorithm 2 (NF₂) */*

1. $F = \text{GetFriends}(u)$, $R = \emptyset$
2. **While** $|R| < k$
3. $u_i = \text{NextNearestUser}(q)$
4. **If** $u_i \in F$, add u_i into R
5. **Return** R

/ Algorithm 3 (NF₃) */*

1. $R = \emptyset$
 2. **While** $|R| < k$
 3. $u_i = \text{NextNearestUser}(q)$
 4. **If** $\text{AreFriends}(u, u_i)$, add u_i into R
 5. **Return** R
-

Figure 3.5.1a Nearest Friends variations

3.5.2 Range Friends

Simply stated, RF returns the friends of user u that are within distance r to a location q

For range friends query respectively :

Input: User u , location q , radius r
Output: Result set R

/ Algorithm 1 (RF₁) */*

1. $F = \text{GetFriends}(u)$, $R = \emptyset$
2. **For** each user $u_i \in F$
3. $\text{GetUserLocation}(u_i)$
4. **If** $\|q, u_i\| \leq r$, add u_i into R
5. **Return** R

/ Algorithm 2 (RF₂) */*

1. **Return** $R = \text{GetFriends}(u) \cap \text{RangeUsers}(q, r)$

/ Algorithm 3 (RF₃) */*

1. $U = \text{RangeUsers}(q, r)$, $R = \emptyset$
2. **For** each user $u_i \in U$
3. **If** $\text{AreFriends}(u, u_i)$, add u_i into R
4. **Return** R

Figure 3.5.2a Range Friends variations

3.5.3 Implementation in MongoDB

Implementation of the primitive geo-location queries in MongoDB. We need to implement the primitive queries in MongoDB and apply the social functionality in top of if using the application logic. But for the primitive geolocation queries we will use MongoDB's geospatial indexes and functionality.

3.5.3.1 Nearest Friends Implementation

Let's start with the Nearest Friends query. For this query we will use the Social engine of the framework to fetch the friend users first with the following query in MongoDB, please note that this query does not use any geosocial indexes as it's only a query to get the users that have a high level of Willingness and high battery levels with the following method calls to our framework

This is the full query that we will want to achieve in a nutshell:

```

candidates.OrderBy(x => x.Distance).
    .ThenBy(x => x.Willingness)
    .ThenBy(x => x.BatteryLevel);

```

We have the willingness and battery level already in the system, the battery level is a direct feedback from each user's device while the Willingness is a variable we use to show the willingness of the user to give battery and as we saw in a previous chapter is calculated by the following form.

$$W = B * T$$

where B is Battery Level and T is the number of times gave energy in the past.

So only variable that our method will need to rely on MongoDB's geospatial indexes is the Distance. By distance we mean the actual distance (km, m and miles, yards) from the user's device to all other users devices in a radius.

To achieve this query in MongoDB we will need to create a geospatial index for the distance field in the database, we can use 2d indexes or 2dsphere indexes. 2d indexes are more simple and are easier for testing purposes as it calculates the distance in the cartesian field while the 2dsphere indexes fit better in a real world scenario because it calculates distance in an earth-like surface. 2d indexes return data in a x,y pair format for long and lat while 2dsphere stores data in GeoJson format⁴.

A 2d index would return the actual point of an object in the format of [40, 5] while the 2dsphere would return a GeoJson object in the format of:

```
{ type: "Point", coordinates: [ 40, 5 ] }
```

The 2d index supports queries that calculate distances on a Euclidean plane (flat surface). The index also supports the following query operators and command that calculate distances using spherical geometry.

These three queries use radians for distance. Other query types do not.

For spherical query operators to function properly, you must convert distances to radians, and convert from radians to the distances units used by your application.

To convert:

- *distance to radians*: divide the distance by the radius of the sphere (e.g. the Earth) in the same units as the distance measurement.
- *radians to distance*: multiply the radian measure by the radius of the sphere (e.g. the Earth) in the units system that you want to convert the distance to.

The equatorial radius of the Earth is approximately 3,963.2 miles or 6,378.1 kilometers.

Consider a collection `places` that has a 2dsphere index.

⁴ "GeoJSON." <http://geojson.org/>. Accessed 29 May. 2017.

The following query would return documents from the places collection within the circle described by the center [-74, 40.74] with a radius of 100 miles:

```
db.users.find( { loc: { $geoWithin: { $centerSphere: [ [ -74, 40.74 ],  
100 / 3963.2 ] } } } )
```

The following spherical query, returns all documents in the collection places within 100 miles from the point [-74, 40.74].

```
db.runCommand( { geoNear: "users",  
near: [ -74, 40.74 ],  
spherical: true  
} )
```


The output of the above command would be:

```
{
  // [ ... ]
  "results" : [
    {
      "dis" : 0.01853688938212826,
      "obj" : {
        "_id" : ObjectId( ... )
        "loc" : [
          -73,
          40
        ]
      }
    }
  ],
  "stats" : {
    // [ ... ]
    "avgDistance" : 0.01853688938212826,
    "maxDistance" : 0.01853714811400047
  },
  "ok" : 1
}
```

3.5.3.2 Range Friends Implementation.

Respectively for the range friends we will rely on MongoDB 2dsphere indexes.

We will use \$near query in mongoDB to get the nearest devices of the user's location:

\$near Specifies a point for which a geospatial query returns the documents from nearest to farthest.

The \$near operator can specify either a GeoJSON point or legacy coordinate point.

\$near requires a geospatial index:

- 2dsphere index if specifying a GeoJSON point,
- 2d index if specifying a point using legacy coordinates.

To specify a GeoJSON point, \$near operator requires a 2dsphere index and has the following syntax:

```
{
  <location field>: {
    $near: {
      $geometry: {
        type: "Point" ,
        coordinates: [ <longitude> , <latitude> ]
      },
      $maxDistance: <distance in meters> ,
      $minDistance: <distance in meters>
    }
  }
}
```

When specifying a GeoJSON point, you can use the optional `$minDistance` and `$maxDistance` specifications to limit the `$near` results by distance in meters:

- `$minDistance` limits the results to those documents that are at least the specified distance from the center point. `$minDistance` is only available for use with 2dsphere index.
- `$maxDistance` limits the results to those documents that are at most the specified distance from the center point.

To specify a point using legacy coordinates, `$near` requires a 2d index and has the following syntax:

```
{  
  $near: [ <x>, <y> ],  
  $maxDistance: <distance in radians>  
}
```

Consider a collection places that has a 2dsphere index.

The following example returns documents that are at least 3meters from and at most 5000 meters from the specified GeoJSON point, sorted from nearest to farthest:

```
db.users.find(  
  {  
    location:  
      { $near :  
        {  
          $geometry: { type: "Point", coordinates: [ -73.9667, 40.78 ] },  
          $minDistance: 3,  
          $maxDistance: 5000  
        }  
      }  
    }  
  )
```

3.6 Cloud technology for remote accessible data storage.

To achieve a backend that is truly independent of location boundaries and be agile and adaptive to the different devices whenever they are on the planet we will need to utilize Cloud technology.

Utilizing cloud we achieve geo replication of data to literally put data nearest to the user for better performance and reliability, we can also scale out the application depending on the traffic and the processing needs to accumulate all the different scenarios of our framework.

In previous chapters we designed the architecture of the application backbone and we decided that we have 3 different entities that form our solution.

First we have the mobile application in user's device itself, this is the client of our framework and will be written in C# using Xamarin cross-platform mobile framework. Then we have the RESTful web services that will accept all the calls from the client (mobile devices) and respond back data to the client. The RESTful services will be written in C# using ASP.NET MVC and web api technology.

Finally we have the backend database which is able to respond to geosocial queries with complex geolocation data structure. MongoDB will be used due to its geo-location indexes, also MongoDB's NoSQL technology will enable us to achieve maximum performance and fulfil thousands of reads and writes without affecting the performance of the application. MongoDB will be deployed in a cloud environment together or separately with the web services.

In a production environment Azure will be used for the web service deployment, in more detail the following services in Azure will be utilized for the web service deployment.

- Azure App Services for web service hosting with scale up, scale out capabilities
- Azure Application Insights for web service monitoring and logging

For MongoDB we will use:

- MongoDB shards in Azure
- Or Mongo Atlas cloud provided by Mongo enterprises.

For testing purposes and during development in the lab we will deploy MongoDB in a Raspberry Pi ⁵ 3 mini computer. Raspberry Pi is the chosen solution for us as is a tiny and affordable micro-computer but powerful enough to be an application server for testing purposes.

⁵ "Raspberry Pi." <https://www.raspberrypi.org/>. Accessed 29 May. 2017.

The framework architecture on Azure will fully cover our needs of scalability and geo-replication serving data from a data center closer to the user's device location for maximum performance and less latency.

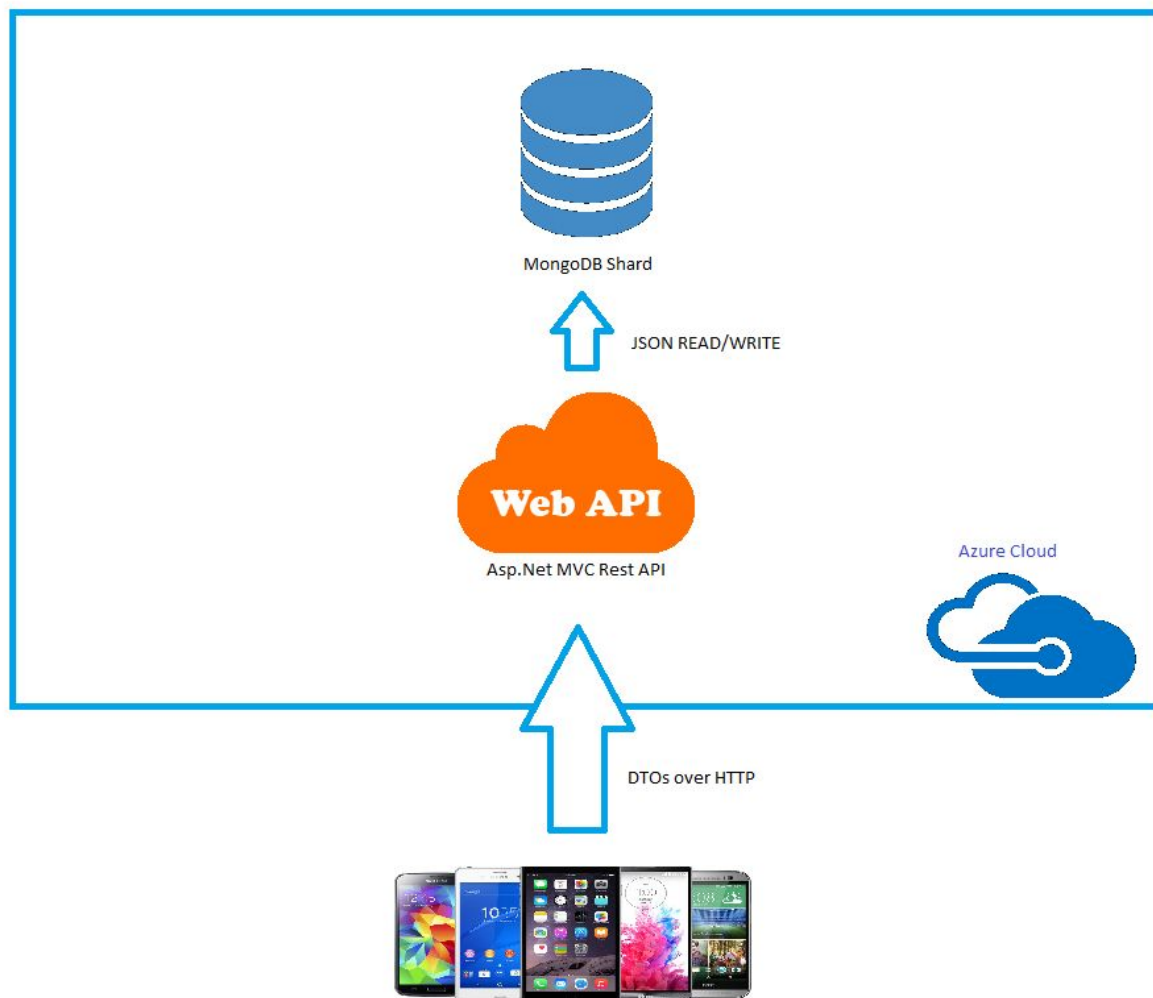


Figure 3.6.b Deployment in production environment using Azure Cloud.

For testing purposes a Raspberry Pi will be used as in the following diagram:

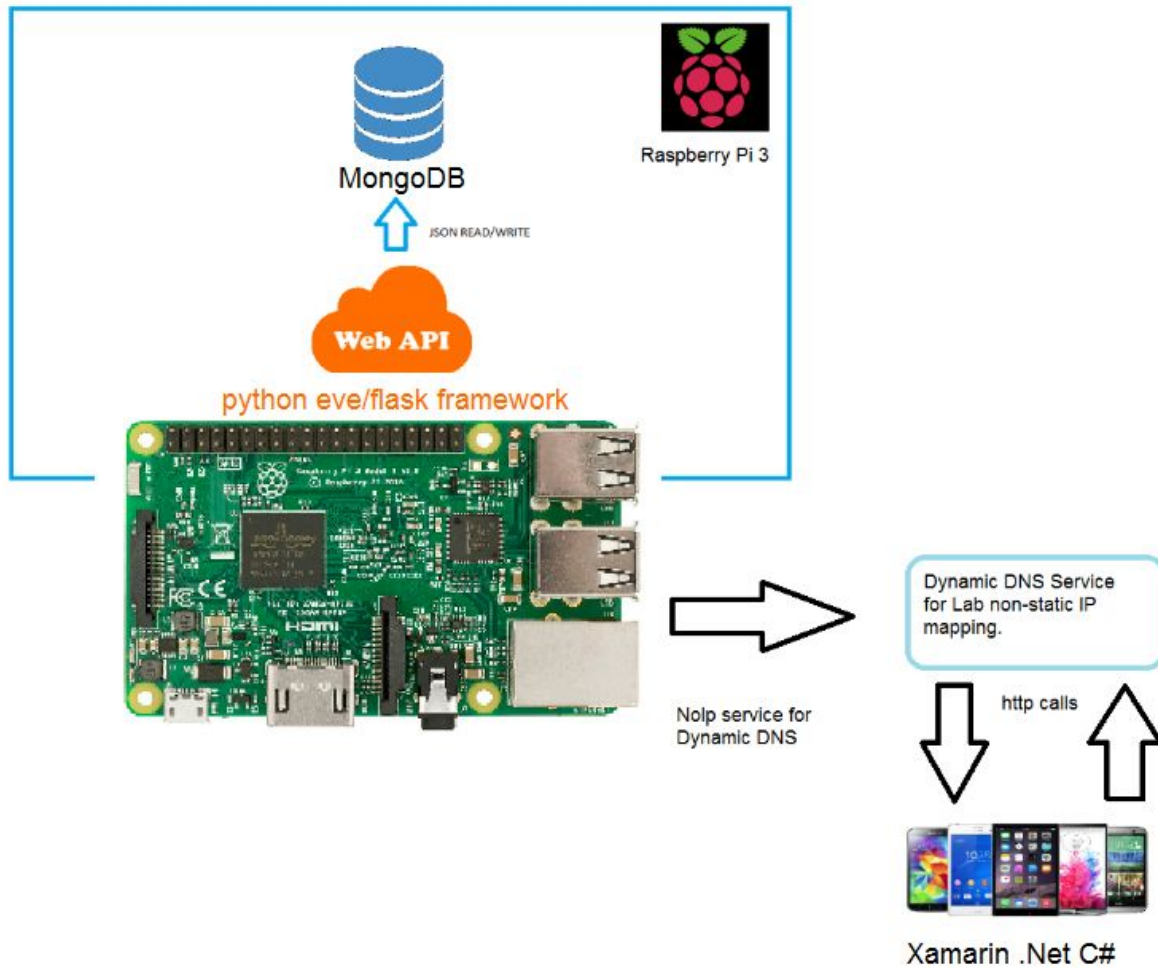


Figure 3.6.b Deployment in the Lab using Raspberry Pi 3

For the Lab environment we will use a different set of hardware and technologies to be more adaptive in testing and evaluation purposes while maintaining a minimum cost for both hardware and software. The mobile-clients part will use the same software and tools as production and will use .Net framework's Xamarin cross-platform development SDK. However for the server part we will not use any cloud architecture as this would increase the cost of this project without any actual need of scalability and performance in the lab environment. For this reason we will choose commonly cheap hardware with minimal energy footprint and low energy consumption. For the software part of the server side we will use open source frameworks and lightweight web servers with which require minimum effort of development as well as they are high compatible and easy to configure in the chosen hardware.

In more detail for the hardware part that will act as a Server with 24/7 uptime we will use:

- Raspberry Pi 3⁶ Minicomputer with Raspbian debian OS⁷

For the software part of the services we will use:

- MongoDB Community edition⁸ NoSQL Database engine
- Python⁹ programming language for the logic of the REST services
- Flask web framework¹⁰ as web server
- Cerberus¹¹ Data Validation framework for Python DTO objects
- Python-Eve¹² REST Api framework to glue everything together

⁶ "Raspberry Pi 3 Model B - Raspberry Pi."

<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Accessed 31 May. 2017.

⁷ "Download Raspbian for Raspberry Pi." <https://www.raspberrypi.org/downloads/raspbian/>. Accessed 31 May. 2017.

⁸ "Install MongoDB Community Edition — MongoDB Manual 3.4." <http://docs.mongodb.com/manual/administration/install-community>. Accessed 31 May. 2017.

⁹ "Python.org." <https://www.python.org/>. Accessed 31 May. 2017.

¹⁰ "Welcome | Flask (A Python Microframework)." <http://flask.pocoo.org/>. Accessed 31 May. 2017.

¹¹ "Cerberus." <http://docs.python-cerberus.org/>. Accessed 31 May. 2017.

¹² "Python-Eve." <http://python-eve.org/>. Accessed 31 May. 2017.

3.7 Source Code Repositories

All source code will be hosted in a Git Repository using Github service under a free academic license.

Our project will consist of two parts, the server side and the client side. The server side is a python codebase for communicating with MongoDB and responding back to the mobile devices and will act as a middleware between the MongoDB database and the clients' mobile devices. The client side code will be a C# codebase solution built with Visual Studio 2017 and Xamarin.

Both parts of this project are hosted on GitHub¹³ in the following URLs:

- mycrowdcharger-mobileclient <https://github.com/sdesyllas/mycrowdcharger-mobileclient.git>
- mycrowdcharger-api
<https://github.com/sdesyllas/mycrowdcharger-api.git>

¹³ "GitHub." <https://github.com/>. Accessed 31 May. 2017.

4. Testing And Evaluation

Last but not least I would like to talk about testing, by testing some times we refer to actual testing but QA teams and testers that they do manual testing with human interaction, while this is necessary for the evaluation of our application most of the times is not enough. By testing in this chapter we will talk about unit testing of the components of our application, all the classes in the code, the interfaces, the service and all the different entities have to be testable in isolation to ensure that every component work as intended like a good polished cog, and then if you put all the cogs together we need to ensure that the machine is still working, this is the integration testing. While developing each component of the application we will write a unit test for each and everyone of them ensuring that they work good in isolation, we expect that every part of the application has a dependency to another part, for example we will write a service to get results from our MongoDB database and log the results into a logger, this service has two dependencies, the MongoDB Logger and a logger service.

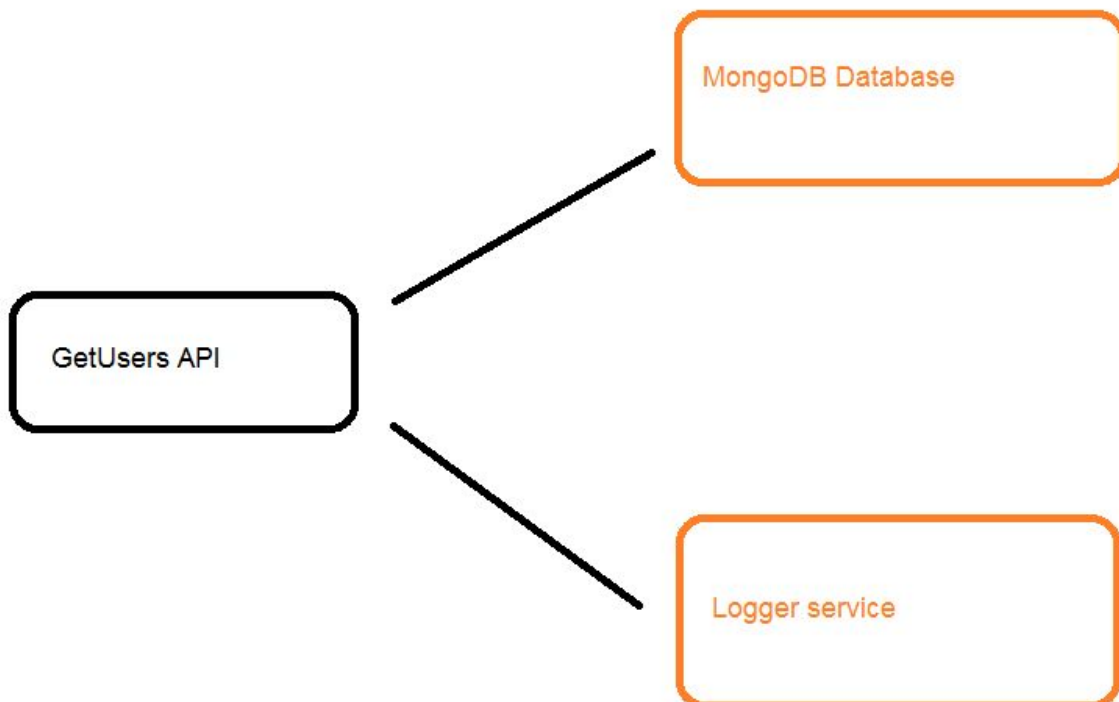


Figure 4.a GetUsers API with two dependencies.

In this example we have a GetUsers API that is an Interface and a class that implements this interface, This class has two other dependencies respectively two different interfaces that the need to be injected through the constructor of the GetUsers class. In testing though we are only

interested to test the GetUsers Api without actually testing each dependency because all the dependencies are expected to have their own unit tests and can be fully tested in isolation. We will see in the next chapter how we can achieve this technique which complies with the five principles of SOLID¹⁴ modern software design.

4.1 Unit Testing framework and Test Driven Development.

To achieve this technique we need to mock all the dependencies and test GetUser service in isolation without much worry about the other two dependencies (MongoDB and Logger Service). Please note that when we say with no worry about the dependencies it does not mean the we don't want to test them, but the unit tests for them will be written separately and in isolation. For example the logger service will have it's own unit tests as well as MongoDB dependency, but each component will be testable in isolation, when the time comes to use these components as dependencies to the service API then we can mock them and mock their behaviour of them so that our unit tests can focus on the GetUser Service only.

It doesn't mean that we don't care about the dependencies completely, we do care actually but our care is limited to their expected usage throughout the GetUser Api and not about their actual implementation. For example the GetUser Api will log a message before returning all the data, we will test if the logger logged something and this something it might be any sequence of characters, we have to remember that we don't focus on the logger at this point, our focus is completely targeted into the Api component, but the Api component unit tests need to test if the logger was called, if the logger is called with whatever results then we can be satisfied about our isolated unit test. This technique usually referred to Unit testing with Mocks¹⁵ is widely used and commonly found when developers take the Test Driven Approach on software development.

We will now write a unit test for the GetUsers service that will have two dependencies of MongoDB and Logger to get a better understanding of this approach.

First we will use the logger service which we can see in the following code snippe together with its interface:

¹⁴ "SOLID (object-oriented design) - Wikipedia."

[https://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)). Accessed 29 May. 2017.

¹⁵ "Unit Testing with Mock Objects - MSDN - Microsoft."

<https://msdn.microsoft.com/en-us/library/ff650441.aspx>. Accessed 29 May. 2017.

```
public class LogService : ILogService
{
    public void Log(string s)
    {
        Console.WriteLine(s);
    }
}

public interface ILogService
{
    void Log(string s);
}
```

Figure 4.1.1 LogService

The log service is a simple service implementing the ILogInterface with a simple Log method. Note the actual LogService implementation that writes the log message to the console, the goal for our unit test is to not use this actual implementation but to mock the behaviour generating a mock that implements the ILogInterface.

Now we can see the DataContext interface with the MongoDB implementation, again this will be mocked at our unit test.

```
public interface IDataContext
{
    User GetUserFromMongoDB(string userId);
}
```

Figure 4.1.2 IDataContext interface, this interface is responsible to fetch data from MongoDB. You don't see the actual implementation here as we will again mock the behaviour in it.

Now let's have a look at the System under test which in our case is the UserService.

```
public class UserService : IUserService
{
    private readonly ILogService _logService;
    private readonly IDataContext _dataContext;

    public UserService(ILogService logService, IDataContext dataContext)
    {
        _logService = logService;
        _dataContext = dataContext;
    }

    public string GetUser(int input)
    {
        _logService.Log($"Ready to get user with id : {input}");
        var user = _dataContext.GetUserFromMongoDB(input);
        _logService.Log($"result : {result}");
        return result;
    }
}
```

Figure 4.1.3 UserService, our system under test

The UserService implementing a GetUser method with only one input parameter which is the user id. When we call this method with the given user id we expect it to fetch the user for us and return the User Object in the return statement. At this point let's have a look at the highlighted code and think about what else we expect this method to do.

We expect this method to do two other things expect fetching the user.

- We expect it to call the Log method of the logger service twice, one before getting the user and one after the user is fetched.
- We expect it do call the GetUserFromMongoDB method and pass the parameter to MongoDB to fetch the user from the actual data context.

The highlighted code underlines the dependencies of this class, we have the following dependencies

- _logService of type ILogService
- _dataContext of type IDataContext

Each dependency is injected through the GetUser class constructor. This way we achieve a separation of concern as an external entity is responsible to inject the dependencies to the class' constructor. An IoC container¹⁶ will be used to inject the dependencies and resolve them during

¹⁶ "Inversion of control - Wikipedia." https://en.wikipedia.org/wiki/Inversion_of_control. Accessed 29 May. 2017.

runtime. With an IoC container we achieve inversion of control as the control of injecting the dependencies is passed into an external provider such as Simple Injector¹⁷.

Now that we saw the implementation of the GetUser Class let's proceed to the actual unit test and find out how the dependencies are mocked.

```
[TestFixture]
public class GetUsersTests
{
    [TestCase(1, ExpectedResult = "User1")]
    [TestCase(2, ExpectedResult = "User2")]
    public string GetUsers_Can_Return_UsersFromDataStorage(int input)
    {
        // Arrange
        var mockLogService = new Mock<ILogService>();
        var mockdatacontext = new Mock<IDataContext>();
        mockLogService.Setup(x => x.Log(It.IsAny<string>())).Callback(
            () => Console.WriteLine("mock logger called")).Verifiable();
        datacontext.Setup(x => x.GetUserFromMongoDB(It.IsAny<string>)).Verifiable();

        var userService = new UserService(mockLogService.Object, datacontext.Object);

        // Act
        var result = userService.GetUser(input);

        // Assert
        mockLogService.Verify(x => x.Log(It.IsAny<string>()), Times.Exactly(2));
        mockdatacontext.Verify(x => x.GetUserFromMongoDB(It.IsAny<string>()), Times.Exactly(1));

        return result.UserName;
    }
}
```

Figure 4.1.4 GetUsers unit test

Let's go through the code and explore line by line what is happening.

1. We write the test with the given user id and the expected result, as you can see for user id 1 the expected result is "User1" and for user id 2 the expected result is "User2", the expected result is the UserName as you can see in the highlighted section in the return statement of the unit test method.
2. We follow the triple AAA pattern and we start with Arranging our system under test first
3. We create two mocks for ILogService and for IDataContext using a Mocking .Net library Moq¹⁸.

¹⁷ "Simple Injector." <https://simpleinjector.org/>. Accessed 29 May. 2017.

¹⁸ "GitHub - moq/moq4: Repo for managing Moq 4.x." <https://github.com/moq/moq4>. Accessed 29 May. 2017.

4. We mock up the behaviour for our mock dependencies and we declare that whenever we call their methods with the given type of any parameter they will be verifiable at the end of the test.
5. We create a new instance of the UserService, we pass the mock Object which is a fake instance of our mock objects created by Moq library. This way we inject a mocked object that fulfills the expectations of the UserService constructor, in our case the mock objects implement the respectively interfaces.
6. Now we act against the system under test, we call the GetUser Method of the newly created instance of our system under test and we set the return in a variable.
7. Finally before returning we asserting the values. For the Log service we expect it to be called twice from within the GetService method and for the DataContext we expect it to be called once. Remember in the implementation of the GetUser method the log is being called twice and this is the reason why we expect to be called twice in the test assertion.

This is the technique of unit testing with mocks and in isolation, we will write similar tests for each component of our application using the following tools and frameworks:

- NUnit ¹⁹. Library for unit testing for .Net framework
- Moq ²⁰. The most popular and friendly mocking framework for .Net framework
- AAA pattern²¹. A pattern for arranging and formatting code in Unit Test methods.

¹⁹ "NUnit - Home." <https://www.nunit.org/>. Accessed 29 May. 2017.

²⁰ "GitHub - moq/moq4: Repo for managing Moq 4.x." <https://github.com/moq/moq4>. Accessed 29 May. 2017.

²¹ "Arrange Act Assert - C2 Wiki." <http://wiki.c2.com/?ArrangeActAssert>. Accessed 29 May. 2017.

5. Application Demonstration

5.1 Android Activities

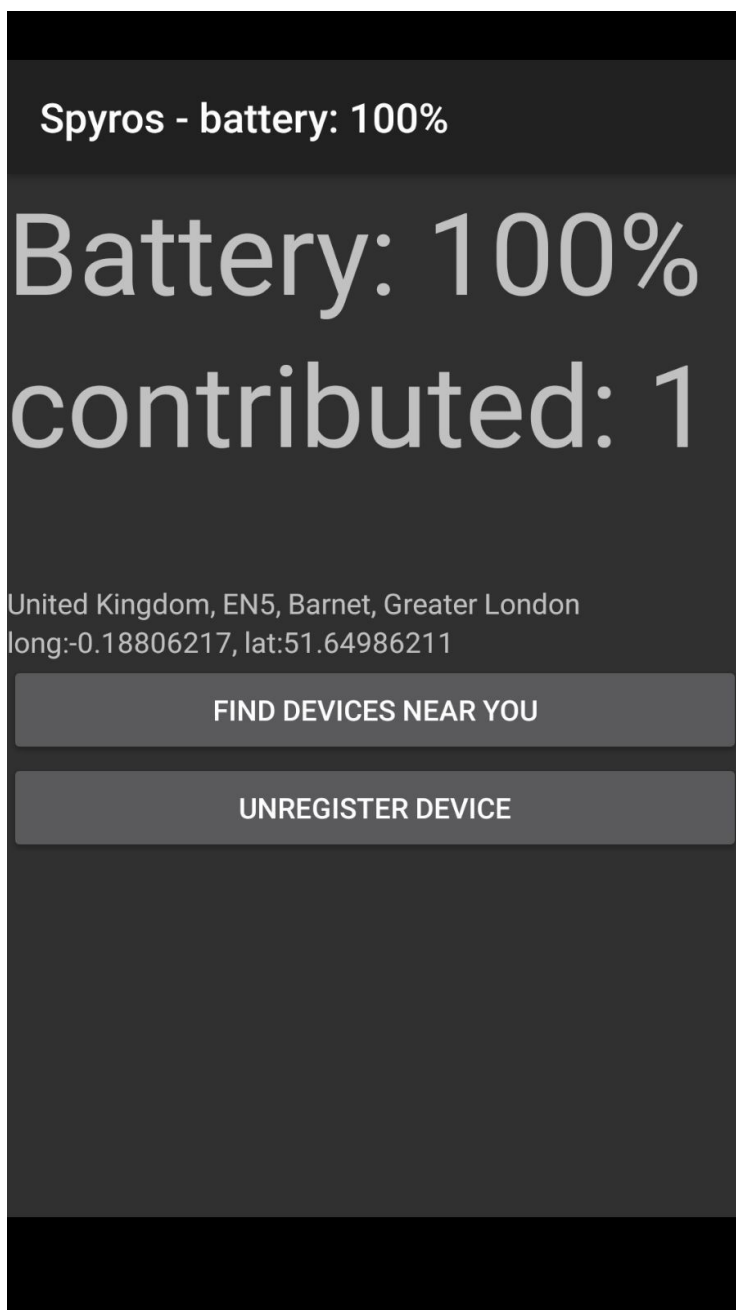
Our application consists of 3 main activities

- Main Activity
- Register Device Activity
- Nearest Devices Activity

Each activity is responsible for every different functionality of our application making use of the crowd charger Api REST client for communication with the NoSql database.

5.2 Main Activity

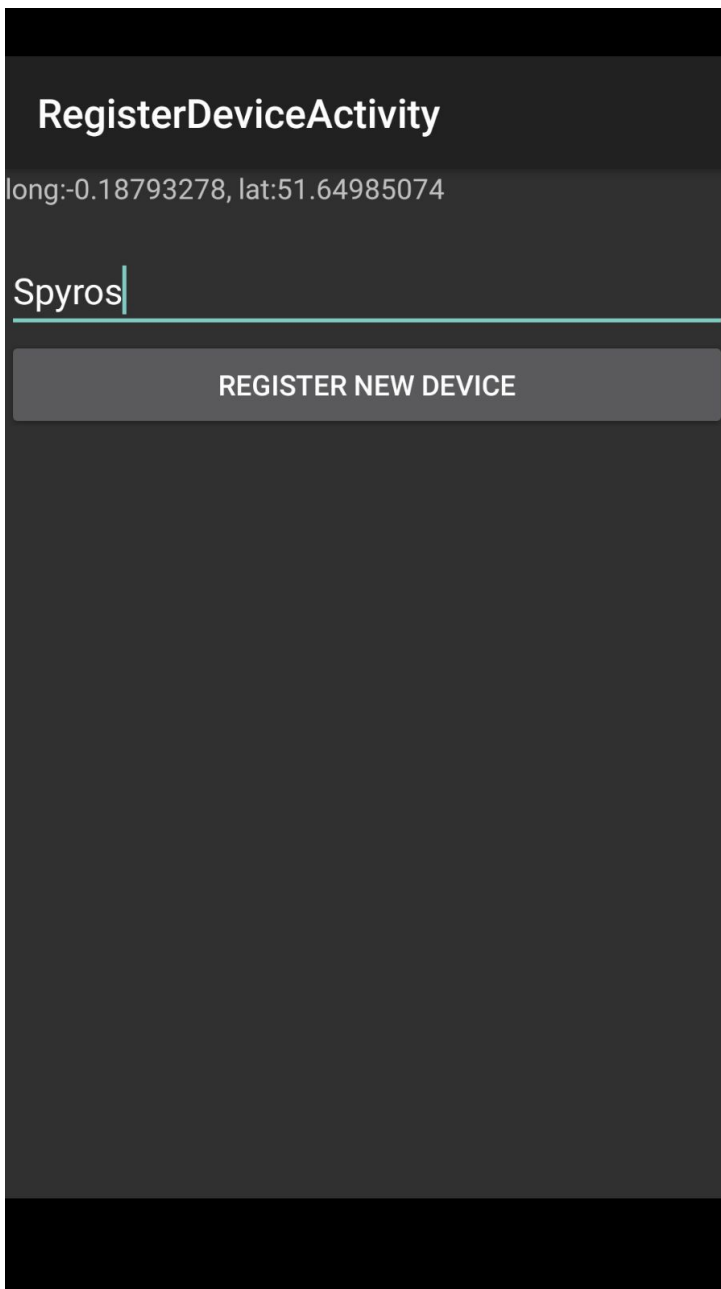
The main activity is the entry point of our application, here the user can watch information as the current battery level, the user info and the current location of the device. If it's the first time that the user opens the application the pipeline will be transferred to the Register device activity for in order for the user to provide a nickname for the device that is using the app. In the main activity there are two buttons, a button to search for the nearest devices and a button to unregister the device, the unregister device button will call the DELETE method in the REST Api to remove the device id from the NoSQL database, this functionality will be performed within the same activity, while the Get nearest devices activity will transfer the application context to the Nearest Devices Activity.



Picture 5.2.a - Main Activity

5.3 Register Device Activity

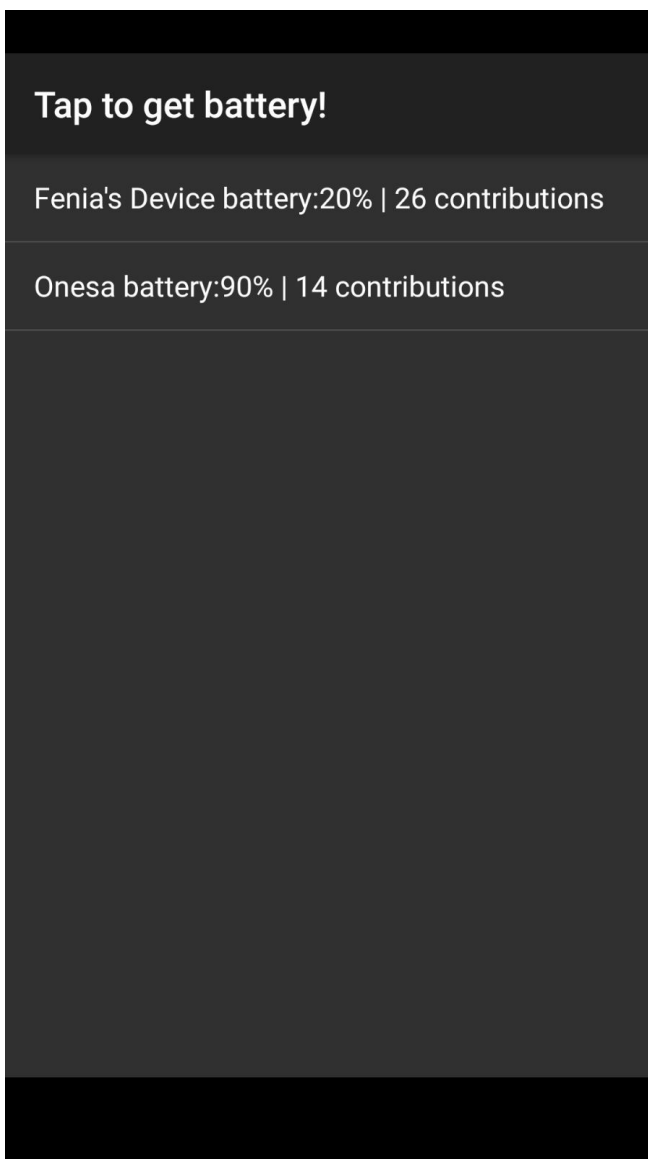
The register Device activity is where the user can register the device id with a nickname to make it more user friendly, generally the device name is an ugly alphanumeric sequence, thus we use the nickname to represent this device to other users.



Pic 5.3a - Register Device Activity

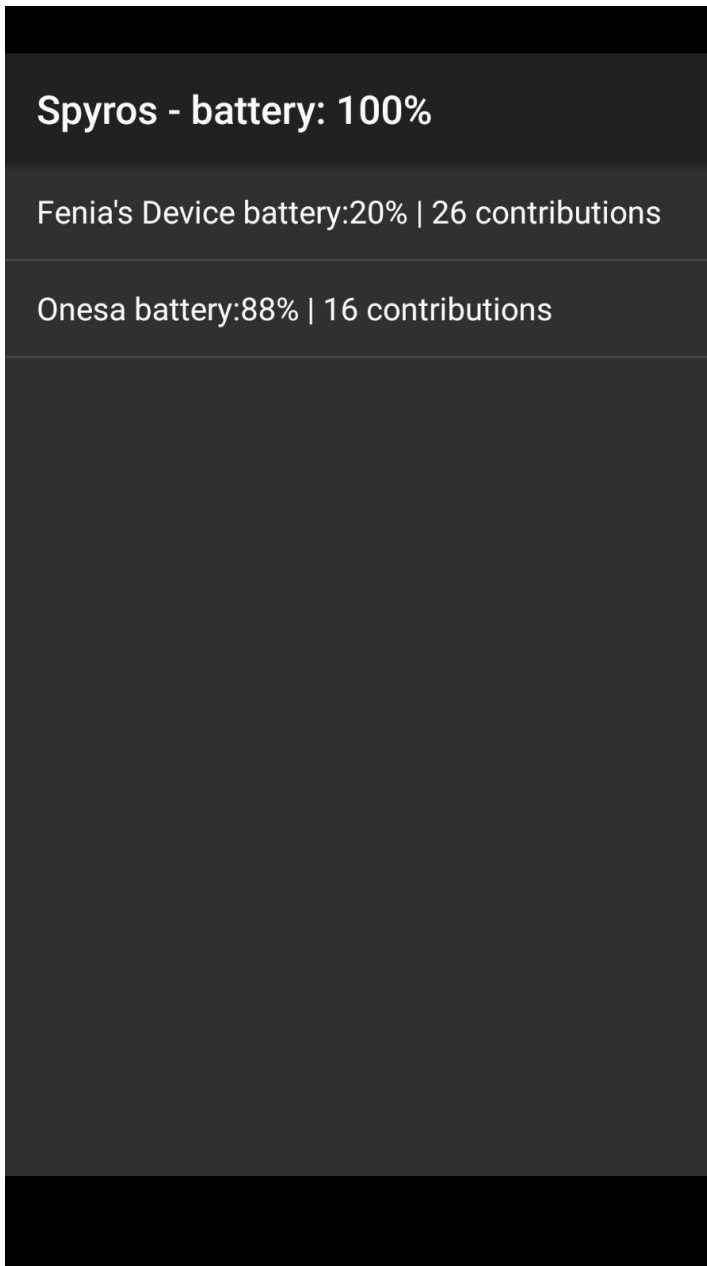
5.4 Nearest Devices Activity

In the nearest device activity we can get all the nearby devices, sorted by distance ascending within a radius of 1 mile.



Pic 5.4a - Nearest Devices, Tap to get battery

The the user is able to tap on any user from the list to get a percentage from his/her device battery.



Pic 5.4a - Nearest Devices, Got battery from a user

5.5 Conclusion

For our prototype we used a mock battery transmission framework which can be easily replaced with a real energy transmission framework depending on the technology and the hardware that is available at that time. The main aspects of this project are the geo location querying and the factor of contributions among the users creating a community of energy contributors. There a couple of things we can consider for implementation in the future such as replacing the mock battery transmitter and receiver with a real electrical energy implementation, also we can consider the usage of messaging queue so that a message broker engine can raise energy consumption tickets to the sender's device and only if they accept the ticket the energy transfer can happen. Another field of improvement will be the implementation and design of more sophisticated crowdsourcing algorithms as for the sake of this project a lot of effort was put on the geospatial queries and geolocation searching algorithms of the devices.

Bibliography

- Xamarin mobile framework <https://www.xamarin.com>
- MongoDB NoSQL database engine <https://www.mongodb.com/>
- Flask Web development python framework <http://flask.pocoo.org/>
- Raspberry Pi micro computer <https://www.raspberrypi.org/>
- A general framework for geo-social query processing Nikos Armenatzoglou Et al.
- Above the Clouds : A Berkeley View of Cloud Computing Michael Armbrust Et al.
- JavaScript Object Notation Data interchange format <https://www.json.org/>
- Microsoft .Net Framework https://en.wikipedia.org/wiki/.NET_Framework
- Dependency injection https://en.wikipedia.org/wiki/Dependency_injection
- Simple Injector <https://simpleinjector.org/index.html>
- Adaptive Code via C# Gary McLean Hall, Microsoft Press Redmond, USA 2014