Abstract

Blablabla, Tralala

Table of Contents

Contents of CD

1. Introduction

We live in exciting times. Throughout the last decade, the internet has found its way into homes and pockets all over the world. Latest statistics estimate that over a third of the world population has access to the internet – in developed countries percentages range from 53% in Greece to up to 97% in Norway, averaging about 73% [Internet World Stats, 2013]. In addition to this, there are more than 1 billion smartphones in use worldwide, according to a report released by Strategy Analytics in October 2012 [Worldwide Smartphone Population Tops 1 Billion in Q3 2012 , 2012].

At the same time, modern societies are right in the middle of a massive shift in consumer behaviour. Pressing unresolved environmental concerns and a global recession that has fundamentally shaken people's confidence in existing economic structures have brought us to a point at which we started to rethink our consumption behaviours. Online marketplaces that encourage and enable sharing, redistributing and swapping goods offer people a sustainable alternative to the hyper-consumerism that dominated the 20th century. "Collaborative consumption" has become one of the new big buzzwords in the start-up scene, referring to a social phenomenon that shows itself in the shape of countless new enterprises and platforms with many different concepts but one shared goal: redefining the way we interact, cooperate and consume. Whether it's peer-to-peer travel platforms such as Airbnb and Couchsurfing or redistribution platforms for used goods like eBay, Craigslist or Kleiderkreisel – technology has opened people's minds towards a more sustainable lifestyle and given them the possibility to connect and organize in order to establish new consumption behaviours.
This kind of development is particularly interesting in the area of transportation. Traditionally, the only alternative to car ownership was using public transportation, particularly for longer distances that can't be travelled by biking or walking. With more and more people gaining access to the internet, a quickly growing community for ride sharing evolved that connected people who own a car and plan

to drive from one place to another with other people who don't have a car but want to travel the same route and fill up the empty seats in the driver's car in exchange for a contribution to the gas expenses – a win-win situation. In the last couple of years, even more new concepts of shared transporation appeared: from car sharing enterprises such as Zipcar, Flinkster or car2go that offer their members a fleet of vehicles to be available on demand to peer-to-peer car sharing models like Getaround or Nachbarschaftsauto where car owners can list their cars to be rented by others at cheap hourly or daily rates – traditional car ownership is becoming less desirable, particularly in urban areas, and people are starting to embrace the idea of sharing.

There is another dimension to the modern age that many consider problematic: While the internet has enabled us to connect with practically anyone in the world and we may now be avidly swapping our used clothes, books and DVDs with strangers all over the country, many people don't even know their next door neighbor. Especially in large cities, individuals easily become isolated and lonely. Big apartment buildings often completely lack a sense of neighborship and mutual support. In a time where everyone is virtually connected to the entire online community on a regular basis, many people feel a strong need and desire to reconnect with real people, with the actual world that surrounds them.

A lot of the new enterprises that appeared within the wave of the collaborative consuption claim it their mission to revive people's engagement in their own community. However, in the very most cases, the primary goal of a business is, naturally, to generate revenue. An honest and effective attempt in renewing the belief in the importance of community should remain a non-profit venture. Neither the people engaging in the process, nor the organization behind it should be motivated by the prospect of accumulating wealth. Nevertheless, in a profit-oriented society like ours, where time is known to be money, it would be naive to assume that a huge mass of people would be willing to sacrifice their time and resources to help others out of sheer altruism.

This thesis is an attempt to to conceptualize an online platform in which people can connect to exchange services and favors within their community without the use of money. Instead of doing this on a one-to-one basis where one favor is traded directly for another ("I do someting for you and in return, you do something for me"), the community as a whole creates a network in which liabilities and gainings resulting from a deed can be abstracted to credit and debt towards the community itself. After two people engaged in a transaction of service, the person providing the service can claim something back from the community, while the person utilizing the service owes something to the community – in other words: "I do something for you and therefore I can ask somebody else to do something for me". A virtual currency or point system is used to account for what members have contributed vs. what they have taken, thereby encouraging everyone to give and take in a balanced manner. The services offered can be anything from simple errands like grocery shopping or transportation of bulky objects, jobs like dog walking or baby sitting to rather skill-based tasks like assembling furniture or fixing a car.

Systems like this have existed for a long time, as the following chapters will outline in further detail. However, most of these communities still operate "offline" and have failed to take the leap into the digital age and avail themselves of the possibilities that modern technologies offer to facilitate such a system, particularly in simplifying processes and attracting new members.
This project is not attempting to reinvent the wheel, but merely to suggest an approach of taking a proven concept, a wheel that appears to be stuck in the past, and redesign it into a dynamic and appealing shape that has the potential of making the old wheel roll again in a new era.
The working title for this platform will be "Skillz". It is short and catchy enough to stick in people's minds, but also carries an obvious reference to the subject matter of the project itself, considering people's individual skill sets as the backbone of such a platform.

A community based on this idea can only work properly if enough people participate to generate a large pool of offers and requests. The larger the amount and variety of people participating, the more flourishing the trade will be and the more likely it is that people will see and appreciate the value in the system – if they find their wants sufficiently met by what skills and services other members have to offer, they will have a motivation to apply themselves within the community to earn the credit they need for employing somebody's services.
Therefore, the main and foremost concern in building the platform has to be the effort of attracting as many members as possible. There are several different aspects to this that will serve as the basis for this project's fundamental criteria:

Low entry barrier

Getting people to participate in the first place must be as uncomplicated as possible. This includes communicating the idea in a concise and appealing way so people know what they sign up for and will be excited to do so.

Usability

The platform has to be easy and fun to use for people with all kinds of different levels of computer and internet expertise. Digital natives have to find their way around just as well as retirees that maybe just recently started their journey into the world wide web and might not yet be familiar with with all the concepts and methaphors that others are taking for granted.

Visual appeal

This may be highly subjective, but it is also closely related to the aspect of usability. A clean and well-structured visual design will undoubtedly enhance the level of usability. Especially for users that consider the internet their second home, this will also be a key factor in the initial attraction of

attention and interest – with all those hip and sleek-looking platforms out there, none of the "cool kids" will want to sign up for something that looks too dull, too noisy or just simply outmoded.

Security and trust

Protection of personal data and securing the system against malicious attacks must be a primary concern for any platform that deals with user's private data. Especially older people are typically more hestitant to give out their personal information, so reassuring promises that their data is in good hands must be adequately backed by high security standards.
Furthermore, appropriate measures to establish and support user's trust in the system as a whole and especially in each other must be taken – afterall, it will be a common scenario for somebody to let a complete stranger into their own home, for example to paint their wall.

Mobile availablity

With more and more people accessing the internet from portable devices, an appropriate mobile solution in addition to the conventional web version is a key factor in the success of any online platform. Especially a system like this can benefit it many different ways from the features that modern smartphones provide. Concepts to make productive use of device features such as location services and push notifications will have to be incorporated into the system in ways that makes sense.

Provided that the abovementioned criteria are fulfilled, the platform will have the potential of drawing enough participants to get a brisk econmy started. Considering what a highly dynamic environment the internet is and how the snowball principle can make interesting things go viral in a matter of days, it is hard to predict how the amount of users and traffic will develop. In the best case scenario, the idea will catch on and spread quickly. This means that during peak growth periods, the number of users and server requests could increase nearly exponentially. A system collapsing from server

overload due to high traffic could damage people's and enthusiasm and trust, likely resulting in loss of a considerable amount of users. Therefore, another important criteria for this project is:

Scalability

The system must be be designed stable enough to handle such an unexpected success and flexible enough so that the adjustments required to serve a fast-growing userbase are unproblematic and easy to perform without major modifications to its basic architecture.

While all of the above-mentioned criteria are essential to the platform's success, the key focus of this thesis will be set on usability engineering. An immensely important aspect of this project is the fact that it is intended to be for everyone. There is no specific target group to cater to – in fact, the diversity of the platform's user base in regard to age, interests, capabilities and talents will be directly reflected by the varierty of offers and requests. The broader and more diverse the pool of participants is, the larger the benefit for everyone. This is especially true because participants will have to be both givers and receivers. If Adam is great with computers but doesn't know how to drill a hole, Beth is versatile with using tools but hates gardening and Carl is an avid gardener but helpless when it comes to removing a virus from his computer, the system will only make sense if there are enough Adams, Beths and Carls signed up who are able to fulfill someone's particular needs, but also have needs of their own.
Therefore, the highest priority has to be designing the platform in a way that makes it fun and easy to use by all kinds of user classes, regardless of their technical expertise. An intuitive, well-structured user interface and a consistent interaction model are the basis for achieving this goal. The overall challenge is to optimize the user experience for a broad audience but at the same time "sprinkle some sugar" that will make specific user classes happy without distracting or confusing others.

The following chapters will give insight into the history of moneyless exchange systems and provide three case studies of currently operating platforms that are thematically related. Subsequently, relevant technological solutions and standards will be examined and evaluated regarding their suitability for being applied within the platform. On that basis, a concept of the technical implementation will outline the overall approach, concretely determine the chosen technologies and give reasons for their selection. Following that, the platform's range of functionality and its underlying economy will be specified in further detail. This part will focus on external processes, describing how the platform presents itself to the user and how the user interacts with the platform. It will also constitute the rules and restrictions that have to be applied to the system in order to ensure fairness and establish trust between users and in the community as a whole.

In addition to to the conceptual part, this thesis will include a prototype mobile application as well as some wireframes and screen designs for the desktop web platform. These practical parts will be serving as a suggested solution to the criteria established above, with particular focus on the challenge of catering to a highly diverse target audience. A chapter accompanying the practical implementation will outline the basic code structure and explain fundamental design decisions regarding visual layout, user flow and code architecture in greater detail.

Finally, a closing chapter will evaluate whether the provided solution meets the established criteria and what aspects of the prototype can be applied to an actual production version. It will sum up all the previous findings, draw a final conclusion regarding the project's feasibility and discuss the future prospects for a possible real-world implementation.

2. Theoretical Foundation

2.1. History and theory of moneyless exchange systems

2.1.1. Barter

In the early ages of mankind, long before the concept of money existed, people relied on direct trades of items such as food, clothing, tools and weapons. Such direct exchanges of goods or services are called barter. Bartering has one obvious disadvantage: It requires a double coincidence of wants [Moffatt, 2013]. If one person owns a hammer that they want to trade for a bag of rice, they need to find somebody who not only wants to obtain a hammer, but also has a bag of rice to offer in return. Otherwise a trade can not take place. Another problematic aspect of bartering is the lack of common measures [Upadhyaya, 2012]. Without money or another standardized form of measurement, it is often difficult to determine and negotiate the value of goods or services, which might result in a disadvantage for one of the involved parties.

These commonly recognized drawbacks of direct bartering systems eventually led to the introduction of metal as a standardized representation of value. Around 2500 BC, copper rings were a common form of payment in ancient Egypt. [Stout, n.d.] The first recorded evidence of money as an accepted means of payment in exchange of goods dates back to around 600 BC, when the first stamped coins were minted in Lydia, an ancient nation located in an area that is now a part of Turkey [Stout, n.d.].

Throughout the course of history, money evolved into an increasingly abstract concept, establishing a convenient and widely accepted means of payment and thus providing the basis for rapid economic growth of modern societies, while simultaneously raising new severe issues: The possibility to accumulate wealth without any maximum limits and the general concept of interest along with banks growing into powerful and insufficiently supervised institutions are only some of the problems that

have created a grave crisis for the monetary system, the effects of which are perceptible more than ever in current times.

Throughout the evolution of monetary societies, barter never ceased to exist, often gaining immense importance during times of financial crisis such as the Great Depression that dominated the USA in the 1930s [History of barter, 2011].

Over time, the obstacles that direct forms of bartering are facing were met with a more organized system of exchange. Instead of trading goods or services directly and one-to-one, parties would pay and be paid with virtual value units, which in return could be spent on purchases from a different party. A centralized institution within the barter exchange keeps track of each member's account balance and processes all transactions. This opened up a new horizon of possibilities, making it more feasible for businesses to engage in barter exchange while at the same time forming the basis for new networks to emerge.

2.1.2. Local exchange trading systems

The term "local exchange trading system" (abbreviated to LETSystem or LETS) was coined in by Michael Linton in 1983. Linton, originally from the UK, migrated to Courtenay, Canada in the 1970s to practice and teach the Alexander Technique [Trahair, 1999]. During the early 1980s, the small town was hit by the recession and local purchasing power suffered immensely. Linton, motivated to continue treating his clients who couldn't afford to pay him anymore, developed and implemented a local system of community exchange that required no use of cash. He established five fundamental criteria that must be fulfilled by a system in order to be considered a LETSystem [Linton, 1994]:

Cost of service

The system should be administered from within the community in a professional and sustainable way. Individuals who run the accounts of the system should be appropriately rewarded for their efforts in the currency that is used within the system. However, any attempt to generate profits from the system, for example by taking commisions on transactions, is illegitimate.

Consent

The system is based on consent given freely by all participants. There is no obligation for any member to participate in a trade. This also means that every new participant joining the network will start with an account balance of zero.

Disclosure

In order for the system to be trust-worthy, it must be transparent to all participants. This can only be achieved by keeping information about a member's account balance and trading volume available to all other members. This will also facilitate participants in collectively regulate the system.

Equivalence to the national currency

The LETSystem currency uses national currency as a means of measure. Without the possibility of putting the system's currency in relation to something of real-life value, it would be virtually impossible for participants to determine the value of their efforts or decide whether another participant's offer is reasonably priced.

No interest

In order for the system to operate without profit, no interest in charged on negative balances or paid on positive balances. This should also discourage participants in storing up large amounts of credits.

Over the past 30 years LETSystems have gained immensely in popularity all over the world, particularly during times of recession. Australia and New Zealand where amongst the most avid promoters of LETSystems. In 1989, the Australian government allocated a budget of $50,000 to faciliate the development of LETSystems all over the country [Orweni, 1993]. By the mid-nineties, 250 different systems existed in Australia with the largest one having around 2,000 members. In 2001, Germany counted over 35,000 members across 350 LETSystems [Hoffmann, 2001]. While systems are inherently independent, there are many examples of local systems cooperating with each other to span greater networks.

With technology evolving, many LETSystems have made use of the new possibilites in one way or the other. While most of them at least have a website providing some basic information, only few have fully adapted to the new era by using software for adminestering account balances and offering a digital directory to participant's offers and requests.

2.1.3. Time banking

Similarly to LETSystems, but with a very different approach on the aspect of measuring value, time banks use units of time as their currency. The first recorded evidence of the use time units as a means of payment dates back to 1825, the year in which Robert Owen, a wealthy industrialist and social reformer from Wales, emigrated to America, purchased a small town in Indiana and called it "New Harmony". His goal was to realize his vision of a *"New Moral World, a world of enlightenment and prosperity leading to human happiness defined as mental, physical and moral health enjoyed in a rational way of life".* [Pitzer, 1989]

The experiment failed, but the idea of a time-based was adapted soon afterwards by Josiah Warren, an American individualist anarchist who was amongst the initial participants of Owen's society. In 1927, Warren opened up the Cincinatti Time Store, the first store that accepted "labor notes" which represented a promise to perform labor in the future, as a form of payment to purchase goods [About time banking, n.d.]. The standard used to determine the value of one hour of labor was 12 pounds of corn, based on the calculation that this is the amount of corn that would be produced by one hour of labor. Warren, who was a strict follower of the labor theory of value, believed that all work should be valued equally and it would therefore be unethical to charge more for a product then the amount of labor it takes to produce it. The store enjoyed great popularity until it was closed down in 1930, when Warren deemed his experiment successful and set out to start new colonies based on his ideas.
The first succesful modern time bank was founded in 1991 by Paul Clover in Ithaca, New York [About time banking, n.d.]. After a recently established LETSystem in Ithaca had failed to attract enough members, Clover developed the system of "Ithaca HOURS". He initially managed to convince 90 participants, amongst them a massage therapist and a toy store owner, to accept HOURS as an exchange for services or goods. The system became an immense success and is still flourishing today, serving as an inspiration for many time banks to appear all over the world throughout the last decade.

2.2. Case studies

2.2.1. "Wi daun wat" - Tauschring Rostock

Established in 1996, "Wie daun wat" [Tauschring Rostock, n.d.] is a local trading exchange circle in Rostock, Germany. The currency used by participants to exchange services or goods is called Knoten (German for "knots"). The value of one Knoten is freely negotiable and not tied to a specific monetary

or time-based measure, though as a guidance it is suggested to roughly set a value of 10 Knoten for one hour of work. During its peak time in 2003, the community counted about 200 members. During the last 10 years, the level of participation decreased significantly. Only few people still actively take part in the exchange and the community has failed to attract enough new members.

One reason for this development might be the lack of appeal for younger generations due to the community's relatively old-fashioned mode of operation. The "Warnowknoten", a newspaper containing general information about meetings and organizational matters as well as a full directory of all offers and requests, is issued quarterly by the administering team. The community also has a website that offers various information, a PDF version of the newspaper's current issue and printable forms necessary for signing up as a new member, advertising new requests or offers and submitting a transaction to be processed on the participating party's account.

The process of engaging in an exchange requires several steps: If a member finds an advertisement in the paper that they are interested in, they can look up the contact information of the person behind the offer in a member directory through a 4-digit identification number. The offers and requests have no initial prices assigned to them – members have to negotiate the amount of Knoten they are willing to accept as a reasonable compensation.

After the service has been provided or exchange of items has taken place, both parties have to fill out and sign a form regarding the transaction

Wohnunghüten
Ich leere Deinen Briefkasten, versorge Deine Pflanzen usw. bei Krankheit oder Urlaub. (1244)
Telefonservice
Übernehme Telefonservice, z. B. Benachrichtigungen, Weckaufträge, auch überregional (1021)
Hausordnurg
Übernehme Hausordnung und Einkäufe bei Krankheit oder Abwesenheit (kein Kfz) (1021)
Vollwert-Büffet
vegetarisch, für besondere Anlässe: Salate, Suppen, Desserts, Brote, Brötchen, Brotaufstriche (1144)
"Urlaubsvertretung"
Gieße Blumen und leere den Briefkasten, wenn Sie unterwegs sind. (1349)
für Ältere
Biete Unterhaltung, Begleitung, Spaziergänge und Einkaufshilfe für ältere Menschen. (1349)
Rundumhilfe Haushalt
Näharbeiten, Reinigung, Fensterputzen, Gartenarbeit, Botengänge, Betreuung...ich helfe! (1226)

3 Bauen und Renovieren

Angebot

Renovieren
Biete Hilfe beim Renovieren - Handlangerarbeiten beim Tapezieren und Streichen. (1454)
Möbelaufbau
Egal ob Tisch, Bett, Schrank oder Regal, ich helfe Dir, die Möbel aufzubaun (1452)
Handwerkern in Haus und Hof
....am liebsten mit Holz, Lösungen für optimale Platzausnutzung, alle möglichen Reparaturen, für jedes Problem gibt es mindestens eine Lösungsidee! (1412)

Nachfrage

Renovieren
Wer kann mir einen Raum im Gartenhaus verputzen (2 m x 4 m) ? (1413)
Heimwerkern
Wer kann helfen: Zuschnitt einer Küchenplatte für das Bad unter dem Waschbecken. (1385)

4 Kleidung

Angebot

Pullover
Du brauchst einen Pullover? Ich stricke ihn mit Hand oder Strickmaschine. (1453)
Näharbeiten
Biete Änderungen - z. B. an Hosen, Reißverschlüsse einnähen, Mützen stricken bzw. nähen. (1447)

Fig. 2-1: Example of offer and request listings [Warnowknoten - Marktzeitung, 2013]

details and send it to the accounting team, which will update both member's account balances according to the value documented in the transaction form.
While the community has obviously made an attempt to adapt to the demands of modern times by providing a website and making content available in a digital form, the necessary steps to truly make the network more accessible and convenient have not been taken. The website is lacking any functionality that would simplify the process of participation, such as the option for members to log into the platform to view their account balance, initiate contact with other members and digitally submit their transaction claims.

2.2.2. Community Exchange System

The system, orginally known as the "Cape Town Talent Exchange" was founded 10 years ago in Cape Town, South Africa by political activist Tim Jenkin [Community Exchange System, n.d.]. It started off as a regular LETSystem, introducing the "Talent" as the unit of currency. The community grew fast and started to attract attention from all over the country. In the same year, six more exchange groups were established in different parts of South Africa. Initially, they operated independently from each other, but as demands to interact with particpants from different groups started to rise, the initiators and administrators of the communities started to think about ways to connect their systems into one big network. The initial problem with this idea was the fact that all communities are inherently closed off systems within which all transactions have to balance out to zero. If one person would be allowed to simply spend their credit within a different system, both affected systems would suffer from an imbalanced economy. But since all of the newly established communities in South Africa already made use computer technology to manage accounts and directories, an internet-based solution was quickly found:

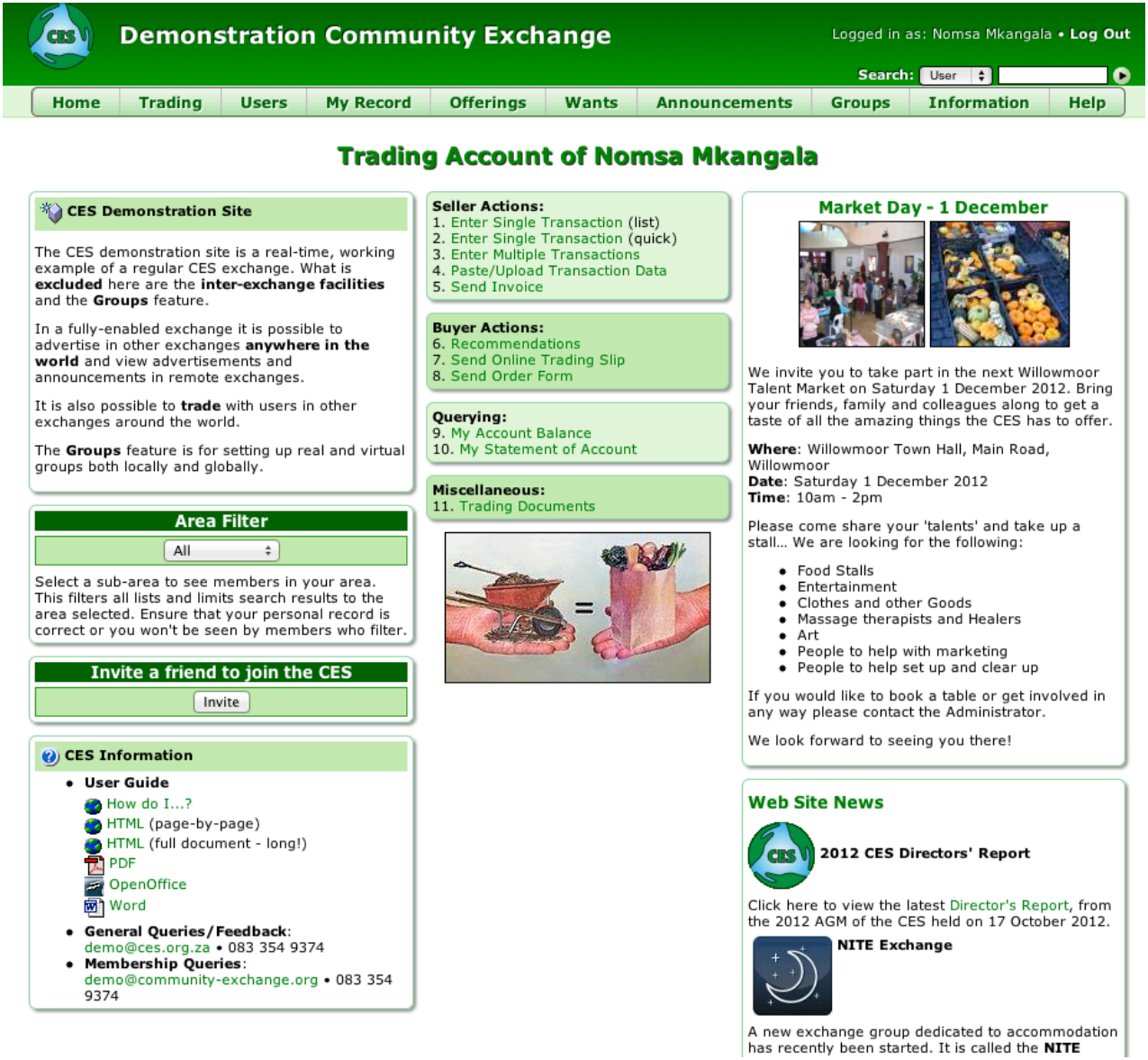


Fig. 2-2: Screenshot of the demonstration account on the CES website [CES Demonstration Site, n.d.]

"The solution was to use "virtual users", who are like real users with accounts in all other groups. Thus if seller A in group A wants to sell something to buyer B in group B, the account would record that seller A actually sold to virtual user A in group A. Virtual user A, who has accounts in all other groups, could "step across" to group B (and be called virtual user B) and sell on to final buyer B in group B. The account in group A would balance to zero as the credits from the sale by seller A would cancel out with the debits of the purchase by virtual user A; and the account in group B would also balance to zero as the credits from the sale by virtual user B would cancel out the debits of the purchase by user B. The balance of trade between the groups could then be measured by the figures recorded for the virtual users." [Jenkin, 2011]

The newly created network to connect local exchange groups was called "Community Exchange Service" (CES). It quickly grew in popularity with more and more groups joining from all over the world to take advantage of the new possibilites of cooperation amongst different groups. By the beginning of 2013, the CES website counted 499 separate community exchanges from 52 different countries.

Unfortunately, not all of the groups listed appear to be still active. During the research for this thesis, two different attempts in joining both a group in San Francisco and in Berlin through the "sign up" section on the CES website were made. After the basic registration process, one was asked to wait for an e-mail of approval from the respective group administrator, but those e-mails never came. Without the approval, the registration process couldn't be completed, hence it was not possible to log into the CES system to browse offers and requests or interact with other members. However, the CES website offers the use of a demonstration profile to become familiar with the platform and its user interface.

2.2.3. TaskRabbit

TaskRabbit [TaskRabbit, 2013] is an online marketplace where people can outsource personal tasks and find people in their neighborhood to do these tasks for them. As opposed to priorly examined platforms, this one does not operate based on a virtual currency but instead involves actual money, allowing a more clear distinction between the people paying to receive services and people being paid to perform services. While the imbalance that could possibly (but not necessarily) result from this business model is somewhat contradictory to the idea of the system developed in this thesis, the platform is still highly interesting to examine because of its thematic similarity and the way it leverages modern technologies to simplify processes.
TaskRabbit was founded in 2008 in Boston, Massachusetts and has since spread to eight more big cities in the U.S., including Los Angeles, New York City and the San Francisco Bay Area. The company has drawn a lot of attention in the start-up scene and is backed by major investors.

The principle is fairly simple: Someone posts a task on the platform, specifies the location, a time frame and and what they're willing to pay somebody for getting the task done. "TaskRabbits" can then bid on the task – this can be higher or lower than specified by the user who posted the task. Task posters can decide whether to review the bidders of the task and assign it to a bidder of their choice or have the system automatically accept the first bidder who is willing to accept the specified price.
Common tasks range from pick-up and delivery of random things to grocery shopping, house cleaning and moving help with the most popular task being IKEA furniture assembly. The platform offers guidance for task posters on how to estimate appropriate task prices. A rating system as well as a thorough vetting process including a video interview and a federal background check for people who sign up as task runners ensures that task posters can count on their tasks being handled by someone reliable and trustworthy.

The variety of people serving as task runners is vast: College students, stay-at-home moms who are running their errands anyway and don't mind putting one or two more tasks on their to-do list, recent retirees who have a lot of free time and are happy to help others out while supplementing their pension – all these are very active demographic groups on TaskRabbit. There are actually quite a few people whose participation in TaskRabbit serves as their primary or even single source of income. According to Leah Busque, founder and CEO of TaskRabbit, the network's most active runners earn up to $5,000 a month, making task running a feasible alternative or supplement to regular or low-paid employment. [TEDxSoMa - Leah Busque - From Social Networking to Service Networking, 2011]

TaskRabbit makes money by adding a 20% service fee to the price that a bidder offers for performing a task. The company makes use of cutting-edge technologies puts a strong emphasis on mobile to make the process of connecting task posters and task runners as easy and hassle-free possible. Through dynamic location services and mobile alerting systems, potential task "TaskRabbits" can be alerted when a task pops up in their direct surrounding, creating possible scenarios like this: Someone posts a task that involves picking up a few items from IKEA. A task runner who is at the IKEA store at the very same moment of the task being posted gets notified. The task runner bids on the task and lets the task poster know that he or she is already at the store. The task poster gets notified about the incoming bid if it is accepted, the task runner is able to collect the items right way. According to Busque, the average time between a task being posted and the first bids rolling in is less than 10 minutes.

With thousands of verified TaskRabbits, the platform has grown immensely in the last few years, branching out to more and more cities across the U.S. and eventually planning to launch internationally.

Fig. 2-3: Screenshot of the TaskRabbit start page after login [TaskRabbit, 2013]

2.3. Available Technical Solutions

2.3.1. Front end technologies

2.3.1.1. Web technologies

HTML, CSS & JavaScript

HTML (Hyper Text Markup Language) was developed by Tim Berners-Lee around 1991 and is still actively evolving through the work of of the World Wide Web Consortion (W3C). HTML is the markup language that is used client-side for building web pages. It describes the content and structure of a page through a set of markup tags. Web browsers are designed to read HTML documents, interpret their content and display them to the viewer.

While HTML can also contain style rules, it is highly encouraged by the W3C to strictly seperate markup and style definitions and use CSS documents for everything concerning the visual design of a web page [The web standards model, 2011]. CSS (Cascading Style Sheets) is a style sheet language that defines the rules of how HTML elements are displayed and formatted.

HTML5, the latest version of HTML, added several new features such as tags for video, audio and canvas elements as well as support for 3D graphics. In combination with CSS3 and JavaScript, it serves as the foundation of a new dimension of rich, interactive user experience in the world wide web. Supplementary JavaScript libraries such as jQuery offer additional features such as animationed effects and support of AJAX techniques for asynchronous server communication.

Flash

Flash is a technology released by Macromedia in 1995. After Adobe acquired Marcomedia in 2005, it was developed and distributed as Adobe Flash. With the use of ActionScript, a fully object-oriented

programming language for developing Flash applications, it is well suitable for creating complex, feature-rich internet applications such as interactive, animated websites.
For a long time, Flash was the standard technology for multimedia-driven websites. However, with the emergence of HTML5 and its large spectrum of new features, the use of Flash in websites is seeing a steady decline . This may also partly be accounted for by the fact that Apple's mobile devices iPad and iPhone, which have become increasingly popular over the past few years, do not support Flash. Today, Flash is mostly used for video playback and the development of online games. Adobe has addressed the shift in technologies for interactive web development by announcing to set its focus on specific market segments. In its "Roadmap for the Flash runtimes", a white paper initially published in February 2012, Adobe draws a clear outline for the future of Flash:

"With the growth of competition in the browser market, browser vendors are increasingly innovating and providing functionality that makes it possible to deploy rich motion graphics directly via browser technologies, a role once served primarily by Flash Player. Increasingly, rich motion graphics are being deployed directly via the browser using HTML5, CSS3, JavaScript and other modern web technologies. Adobe expects that this trend will continue and accelerate, and Adobe will continue to play an active role in this space.
Adobe believes that the Flash runtimes are particularly and uniquely suited for two primary use cases: creating and deploying rich, expressive games with console-quality graphics and deploying premium video.
This focus does not mean that existing content will no longer run, or that Flash cannot be used for content other than gaming and premium video. However, it does mean that when prioritizing work, gaming and premium video use cases will take priority." [Roadmap for the Flash runtimes, 2012]

2.3.1.2. Mobile technologies

Native apps

The two big players in the world of mobile devices are Google and Apple. In the 4th quarter of 2012, a ranking published by the Intenational Data Corporation (IDC) amounts the total worldwide market share of both platforms combined at over 90% [Android and iOS Combine for 91.1% of the Worldwide Smartphone OS Market in 4Q12 and 87.6% for the Year, According to IDC, 2013], with Google's Android OS being the clear leader, running on 70.1% of all smartphones shipped in late 2012, while Apple's iPhone makes up 21% of sold devices in the same time frame.
Android applications are natively written in Java using the Android SDK and can be developed on any platform including Linux, Windows and Mac OS. Development tools are available for free and there is no special licence required to create and publish apps.
iOS apps are written in Objective-C using the iOS SDK. To create native apps for iOS platforms that can be published in the Apple App Store, developers must compile their app on the official iOS SDK running on Mac OS X. While development tools such as the IDE Xcode are available for free, a paid licence (currently $99/year) is required to test apps on real devices and publish them for App Store release.

Native apps, as opposed to mobile web apps, have the advantage of having full access to the specific device's capabilities such as the camera or accelerometer. They are downloaded and installed directly onto the device, therefore they can be used without necessarily requiring the user to be connected to the internet, provided the app itself does not need to access remote servers to work properly.
The obvious disadvantage of a native app is that it will only work on the respective platform that it was developed for. This means that if a business wants to release an app and has decided for a native approach, it will either have to target just one specific platform, or develop the same app in several

different languages for all desired target platforms. An alternative to this is the use of a hybrid approach.

Mobile web apps

Mobile web apps are apps that run in the web browser of a mobile device, instead of being downloaded and installed directly onto the device. They therefore require the device to be connected to the internet. Web apps have the clear advantage of being platform-independent, since all they need to be accessed is a mobile web browser, a core component of all web-enabled smartphones.

Since web apps are not published in app stores or marketplaces, developers can distribute and update them on their own terms without having to go through complicated approval processes that can sometimes hold back the release of an app for a considerable amount of time. On the other hand, not being listed in official stores may have the disadvantage of being difficult to discover – publishers of mobile web apps will have to find other ways of promoting their app to new users.

The new features of HTML5, CSS and JavaScript, along with additional frameworks if required, offer means for a fairly good imitation of platform-specific UI elements that can create an almost native look and feel of a well-designed web app. However, current mobile web browsers are only able to access a limited set of a device's specific capabilities. While properties such as orientation and geolocation may be available, a device's camera or accelerometer are currently not enabled in mobile web browsers.

Hybrid apps

Hybrid apps combine the advantages of both native apps and web apps. A hybrid app typically uses a web view control as a native wrapper for a web-based application. Frameworks such as PhoneGap, Appcelerator Titanium (both JavaScript-based) and Adobe AIR (ActionScript-based) offer developers an abstraction layer that exposes native APIs (and therefore full access to the device's capabilities) to

the app. Hybrid apps can also access the device's file system for caching data, making it possible to design the app in a way that it can be used when the device is offline.
Since hybrid apps are deployed natively for each target platform, they can be published to app stores and marketplaces just like any fully native app. This solves the problem of difficult discovery of mobile web apps.
Hybrid apps are a viable solution for cross-platform development and will likely see a sharp increase in popularity in the future. In February 2013, analyst and research house Gartner predicted that more than half of mobile apps deployed by enterprise by 2016 will be hybrid. [Gartner Says by 2016, More Than 50 Percent of Mobile Apps Deployed Will be Hybrid, 2013]

2.3.2. Back end technologies

2.3.2.1. The LAMP stack and its variations

LAMP is the acronym for a solution stack based on the following components:
Linux – the operating system
Apache – the web server
MySQL – the database management system
PHP, Perl or Python – the scripting language

These four layers combined form the basic architecture for a general purpose web server serving as the underlying back end for many dynamic web sites and applications. LAMP has become increasingly popular throughout the last decade, with many large and traffic-intensive platforms such as Flickr, Facebook and Wikipedia using it as their base architecture, though particular components of the stack may be exchanged with other technologies to fit indvidual requirements.

The LAMP stack offers several significant benefits: The components are open-source and available for free, provided they are used in other open-source projects. Required hardware components and commercial licences for database systems and application servers are very inexpensive compared to other popular solutions such as the ASP.NET or J2EE frameworks. LAMP is considered reasonably secure, very flexible and, if configured correctly, can by highly performant and scalable.
Disadvantages of the LAMP approach are seen less commonly in the bundle as a whole, but rather in known weaknesses of its components in particular scenarios. The following discusses the individual layers in further detail and suggests viable alternatives for each component.

Operating system solutions

As stated above, the L in LAMP refers to the operating system that the server is running on. Traditionally, this would be one of the many freely available Linux Versions: CentOS, Debian, RedHat Enterprise Linux, SuSE Linux or Ubuntu. However, the bottom layer is generally exchangeable with any arbitary OS – there are WAMP and MAMP configurations that run Windows or Mac OS as the underlying system respectively. Furthermore, there are a handful of UNIX-based operating systems that are less popular in commercial use but hold significant advantages over the traditional approach in back end architecture.
Solaris, developed by Sun Microsystems, offers high scalability and a set of powerful features such as ZFS, a file system combined with volume management capabilites that is designed to be highly robust against data corruption through use of data snapshots, multiple copies, and data checksums. Due to its excellent support of multithreading, Solaris also shows better performance than Linux on most hardware platforms, especially on larger installations with multiple CPUs and cores. Therefore, Solaris distributions are certainly a viable alternative if maximum performance is a crucial business requirement. [Chisnall, 2006]

Another solution that gained a lot of popularity, especially in environments in which security considerations are of high priority, is OpenBSD. It is designed to be outstandigly robust against attacks, for example by prohibiting simultaneous writing and execution processes on the same chunk of memory. A major disadvantage of OpenBSD is the systems relatively poor support of symmetric multiprocessing, making it significantly slower than most other operating systems. OpenBSD is a good solution for projects in which security is considered more important than performance. [Chisnall, 2006]

Web server solutions

The layer that contains the actual web server in the LAMP stack is usually a version of Apache. Apache was developed in 1995 as is still the most popular web server with a market share of over 60% on April 29, 2013, according to daily updated usage statistics on the W3Tech website [Usage of web servers for websites, 2013]. Apache is already installed and pre-configured in common Linux distributions and therefore relatively easy to set up. It comes with a variety of built-in features and is highly configurable.

An alternative that sees continuous growth in popularity since its release in 2004 and made its way in the the top three of the most commonly used web servers is nginx. As opposed to Apache, it uses an asynchronous approach, resulting in significantly lower resource consumption. Being able to handle simultaneous requests in a single thread, it performs faster than Apache in most scenarios, especially for delivering static content. [Hutchinson, 2011]

If the main point of concern when chosing a web server is maximum scalability, Yaws (short for "Yet another web server) is a viable alternative. Written in Erlang and using a lightweight threading system, it is built to handle a high number of concurrent processes and therefore interesting for high-traffic websites and applications that face a high number of simultaneous requests. [Yaws, 2013]

For Java-based solutions, various open-source and commercial web servers are available, with Jetty, Tomcan and Glassfish being amongst the most popular ones.

Database solutions

The third layer of the LAMP stack is formed by the database management system, traditionally this would be MySQL, a popular and well supported solution that is used by many high-traffic websited including Wikipedia, Facebook and Flickr. While MySQL is generally a robust and flexible database managment system that is easy to use and very fast for simple queries, it shows significant performance issues on more complex queries. [Chisnall, 2006]

An alternative to MySQL that is optimized to perform much better on complex queries is PostgreSQL. PostgreSQL also has the advantage of coming with a permissive free licence that allows for free use even in commercial projects, while MySQL requires paying for a commercial licence when used in non-open source projects.

Another SQL-based solution with a completely free licence is SQLite. The main difference between SQLite and most other databases is that SQLite is not running on a server – it is merely represented by a flat file on a disk. This has both advantages and disadvantages. Making a complete back-up of the database for example could not be any simpler as it requires nothing more than copying one file. On the other hand, handling concurrent clients that try to access the database at the same time becomes a problem. SQLite is therefore mainly used in embedded applications, where only a single client will need to access the the stored data. [Chisnall, 2006]

Besides the traditional SQL-based approach, a new wave of non-relational database solutions has drawn a lot of interest during the past few years. The so-called NoSQL systems use a different approach of storing and retrieving data, making it more scalable for large amounts of data. However, as most systems use simple key-value storage mechanisms which may be structured, but are lacking the possibility of modelling relationships between different fields, the use of a NoSQL database is

only advantageous in certain scenarios. Where relationships and complex queries are of little interest, large chunks of data can be accessed and written significantly faster than with any SQL-based solution. Popular systems are MongoDB, which holds at least some SQL-like functionality such as queries and pre-defined indexes, CouchDB, which is optimized for high consistency and Redis, which is famous for being extremely fast.

Server-side programming languages

The top layer of the LAMP is represented by the server-side programming language, typically a scripting language. Released in 1995, PHP is still by far the most popular and wide-spread choice when it comes to server-side languages – as of April 29, 2013, it is used in 78.7% of all websites recorded by the W3Techs survey [Usage of server-side programming languages for websites, 2013] and is still showing an upwards trend. [Historical yearly trends in the usage of server-side programming languages for websites, 2013]

PHP's immense popularity can undoubtedly be seen as an advantage in itself when considering it for a new project – PHP developers are easy to find, learning the language is relatively straightforward and a big community of experienced developers offers great support for beginners. There are countless libraries and extensions available and it is widely supported by other frameworks. It is also highly polarizing. While it is often sharply criticized by blogging developers (such as Alex Munroe in his blog post "PHP: a fractal of bad design" [Munroe, 2012]), many others (such as Fabien Potencier in his blog post "PHP is much better than you think" [Potencier, 2012]) argue that while it may not be the most perfect language in the world, it has come a long way in the past few years and evolved into a mature language that offers full support for OOP, is highly flexible and convenient to write.

Some alternatives to using PHP as the server-side programming language are Perl, Python and Ruby. While Perl has shown a steady decline in popularity over the past decade (as indicated by a Google

Trends query comparing the web search interest of keywords "Perl", "Python" and "Ruby") it might still be a viable option where high flexibility is a key concern. Perl 6, a complete redesign of the Perl language, is currently under development and might have the potential of reviving Perl to a comeback.

Python, released in 1991, is widely appreciated for its elegance and readability. It is a powerful yet simple language that, in spite of its relatively small market share (0,2% of the websites recorded on April 29, 2013 by W3Tech [Usage of server-side programming languages for websites, 2013]), still remains a viable option for web and mobile development. The most commonly used web framework for Python is Django. Labeling itself the "web framework for perfectionists with deadlines" [Django Framework, 2013], Django is designed to enforce clean design patterns and make developers produce well-organized, reusable code. Django is used in popular websites such as Pinterest and Instagram and also serves as the underlying framework of many mobile apps, web-based and native alike.

Another increasingly popular solution is Ruby in combination with the Rails framework. It has drawn a lot of attention especially in the start-up world, where agile development methods and fast-changing requirements benefit from a flexible language that enables fast production of results and iterative approaches. Though Ruby is significantly slower its direct competitors PHP and Python, Ruby on Rails seeing vast growth in popularity as a framework for websites and mobile apps. It is used by platforms such as Hulu and GitHub. In 2013, Rails released three urgent updates due to the discovery of highly critical vulnerabilities, putting 240,000 websites at risk. [Constantin, 2013] It remains to be seen whether those recent discoveries of security threats will cause the framework to lose some of its popularity.

Some other languages that are highly suitable for server-side programming are Scala, Erlang and Haskell. Of course, good old Java remains a viable option as well and still holds a market share of 4% [Usage of server-side programming languages for websites, 2013].
Very recently, there has also been a lot of fuss in the developer community about Node.js, a lightweight and highly scalable platform that brought JavaScript to the back end world.

2.3.2.2. Microsoft's ASP.NET framework

With a market share of over 20% [Usage of server-side programming languages for websites, 2013], ASP.NET is the second most common server-side web application framework on the internet.
Applications based on the ASP.NET framework run on Microsoft's web server solution IIS (short for Internet Information System). IIS is simple to configure and maintain and monitor through a graphical user interface, however, it also has a reputation for poor security. Typically, ASP.NET application choose Microsoft's SQL Server as the underlying database, although third-party solutions, including NoSQL databases, are supported by the framework. On the programming side, the ASP.NET framework supports languages that conform to the Common Language Infrastructure (CLI) specifications, including C# and Visual Basic.
While the ASP.NET framework is a powerful and highly configurable solution for web development, it is important to note that is is underlying several restrictions: Micosoft as the underlying platform and IIS as the operating web server are without any alternative when choosing .NET. Furthermore, licencing costs for commercial applications as well as costs for add-ons and third-party tools can be high.
There are several extensions that make it possible to use the .NET for mobile development. While the regular ASP.NET framework is suitable for mobile websites, a particularly interesting solution for

mobile apps is Xamarin (formerly called MonoTouch) [Xamarin, 2013]. Xamarin is a cross-platform implementation of the .NET framework, allowing to write mobile apps for iOS and Android Windows Phone using C#. Apps developed with Xamarin can share the same business logic, data access, and network communications code while native platform APIs can be called directly through the C# code, enabling full access to the device's specific features and therefore offering high performance paired with a native user experience.

2.3.2.3.Backend as a Service

In traditional software development lifecycles, building a back end takes up a great chunk of time, often accounting for up to or even more than half of the total development time. With cloud computing becoming more and more ubiquitous throughout the last couple of years, a new trend in the world of software development emerged: Backend as a Service (BaaS). With a cloud-based out of the box back end, providers of BaaS offer clients to exchange the traditional web server stack for a simple yet powerful solution that can be customly tailored to fit individual needs. Commonly used features that can be added include user management, location services, push notifcations and integration for social networks. Especially in the mobile sector, choosing BaaS may often be a advantageous business decision, saving developers a lot of time and enabling them to ship their apps faster then potential competitors. Amongst the many BaaS providers that are on the market today, the most popular ones (according a post by Antonio Martínez in the tutorial blog run by iOS guru Ray Wenderlich) include StackMob, Appcelerator, Parse, Applicasa and Kinvey [Martínez, 2012].

3. Concept

The first part of this chapter will outline the system's underlying economy and the rules that will be applied within the system. The second part will focus on details about the technical implementation and, discuss the technical solutions decribed in the previous chapter and choose a concrete approach for both the prototype developed in the scope of this thesis and a possible real-world implementation.

3.1. Concept of the System's Economy

The system is based on points serving as a virtual currency. There is no exchange of real-world currency involved and no initial or recurring membership fee – signing up and participating is free of cost for everyone. Since people have a very clear notion of the value of money but will most likely have difficulties assigning a value to their work using an unfamiliar system of points, one point is intended to have roughly the same value as one unit of the respective country's real-world currency. This will make it easy for people to understand the points concept, come up with realistic price estimations for their own offers and requests and put other people's prices in relation to that.

Everybody who signs up freshly starts out with an account balance of 0 points. Both negative and positive balances are allowed only to a certain limit. This will avoid the system to establish massive imbalances where some people accumulate too much credit without spending enough and other people go into enormous debt by constantly taking from the community but not giving anything back to it. When a user first signs up, the initial limit for negative balances is 50 points and the limit for positive balances is 250 points. This means users could initially earn up to 250 points before they would be forced to also spend some of their points if they want to be able to generate more earnings

again. On the other hand, they are initially allowed to spend only 50 points before they would be blocked on spending until they have given something back to the community. Setting the initial negative limit relatively tight is intended to avoid large-scale exploits of the community through people who sign up once to have somebody do something for them and then just silently keep their accounts without ever giving anything back to the community. While this might still happen, the 50 points limit for first-timers will ensure that such exploits remain small and can't harm the system in profound way. Active participants who regularly give to and take from the community in a relatively balanced manner will get their credit and debt limits gradually increased. Similarly to the concept of credit ratings for bank accounts, those users have proven their trustworthiness to the community and are rewarded by being allowed bigger scales of transactions.

There is no interest associated with account balances – this means that there is no benefit from saving points and also no disadvantage from being in debt. Having a negative account balance should not have the same negative connotation that it has in the real world. In face, negative account balances are completely acceptable and even necessary for the system to even out – nobody could ever have a positive balance if there weren't at least some people with a negative balance. The objective for every participant should be to balance out their account as good as they can, taking a negative account balance as an incentive to apply themselves more by doing something (perhaps actively browsing nearby requests and see if there is something that they could take on) while a positive account balance should motivate them to spend some of their points. If all accounts within the community would be summed up, the outcome will always be exactly 0 points.

The system is generally based on the assumption that every person will be able to contribute something to the community according to their individual talents, interests and capabilities. Some people might have very specific skills like knowing how to fix a car or design a business card while

others might prefer to take on tasks that are rather time-based than skill-based such as doing somebody's grocery shopping or walking somebody's dog.

However, all this only applies to a world full of able-bodied, mentally and physically healthy people. This is far away from reality. In our society, and in our direct neighborhood, there are people who rely on the help of others without being able to give anything back other than love and gratitude. Gladly, in most cases these people are supported by a net of social services and health care. But the social and political institutions backing this system only work up to a certain extent. They will do what is necessary, but often enough there is no money to go any further than that. Especially older people who have little or no relatives to help and support them often fall into poverty and loneliness.

The platform has great potential to establish some structures to care for and support people in need. With every transaction that two users engange in, both of them will be asked if they want to donate a share of the agreed points to the needy. If a service would be compensated with 50 points, the performer of the service can chose whether they want to receive the full 50 points or whether they would be willing to receive a smaller amount, for example 48 points, with the difference going directly into a virtual donation pot. Simultaneously, the receiver of the service has the option of donating by increasing the sum booked from their account by one or more points. Incorporating the donation call into the transaction process makes a lot of sense as people are already spending or receiving points anyway and will be much more likely to say "Sure, one or two points for the good cause, why not?" than proactively making an unprompted donation.

The donation pot will be used to compensate people for performing tasks for people in need. The people performing such tasks may of course choose to do them for free, but offering them at least a small amount of compensation for their efforts could turn out to be a highly effective way to give people an incentive to help others. Obviously, this aspect of the community will require working closely together with social institutions and aid organizations to figure out where help is needed most. In the best-scenario case of the community growing into a flourishing economy with many

transactions taking place every day, this could turn out to be a powerful contribution to relieving social issues.

3.2. Technical Concept

3.2.1. Front end

The final platform should be available both as a regular web version and as an app for smartphones. In an actual release version, both client sides (website and app) would be built in HTML5, CSS3 and JavaScript to fulfill the critierium of universal availability. Building the website in Flash could be problematic as it requires an additional browser plug-in that not all potential users might have installed. Especially less experienced users might not be able or willing to install anything in order to view the site. Furthermore, Flash is not supported by iOS devices and therefore there would be no way of accessing the website from an iPad. HTML5 has become the de-facto standard technology for building modern websites and will ensure that the platform is accessible by a broad audience.
The final mobile app is intended to be a hybrid solution, using PhoneGap as a wrapper to deploy a standalone app. Choosing a hybrid over a native or web-based approach is also attributed to the criterium of universal availability. The app should be available to all smartphone users, regardless of their operating system. To achieve this with a native solution would require the app to be developed from the ground up for every supported platform, resulting in significantly higher costs of initial development and maintenance. While a web app would be instantly available to all web-enabled devices, it lacks the ability to access some essential device capabilities that may be important for the app, for instance the possibility to send push notifications to the user. It would also be harder for users to access the app, as they have to go through their browser and type in the address (or use a bookmark) every time they want to use it. All of the presented issues can be solved by a hybrid

approach: It reduces the costs of development and maintencane, even enables re-use of some of the code that is used for building the website, it offers an interface to access specific device features and it makes the app discoverable in app stores and market places and easily be accessed through the device's home screen.
The decision of choosing PhoneGap over similar frameworks like Appcelerator Titanium and Adobe AIR is based on the fact that apps developed with PhoneGap can utilize HTML5/CSS3 standards and access native device capabilities through a JavaScript SDK. Appcelerator Titanium uses a different approach: Using the Titanium JavaScript SDK, the entire application will be written in JavaScript, which will be compiled down to native code for deployment. This accounts for better performance, but is usually only relevant and noticable for highly CPU-intensive apps. Apps developed with Adobe AIR use Flash an dActionScript 3 as the underlying technologies. Using PhoneGap has the clear advantage of allowing the developer to re-use code across the desktop web platform and the mobile app and would therefore be the framework of choice.

3.2.2. Back end

The back end should be designed for high security and easy scalability. Known as one of the most secure operating systems on the market, OpenBSD will serve as the web server's underlying platform. Apache will be the web server of choice as it is highly configurable and proven as a viable solution in many large-scale projects. Coming with a free licence and being known for outstanding reliability and consistency, the platform's database will be PostgreSQL. The back end API will be built in Python using the Django framework. Due to its clean design and syntax, Python is known to encourage developers to write efficient code that is easy to read and maintain. Therefore it is an excellent language to develop a flexible and highly scalable back end that allows for quick changes without having to recode large parts of the system.

In order to minimize bugs and hence also minimize security threats, the development process should include thorough unit testing. Once the system is live, extensively monitoring and analyzing traffic and performance (by using Nagios or a similar tool) will be crucial to identifying potential bottlenecks and predict when it will be necessary to take steps to scale up the system. Depending on where the bottleneck is located, appropriate measures to cope with increasing server load can be upgrading hardware (scaling up) or creating a cluster by adding more web or database server nodes and a load balancer to distribute requests (scaling out).

3.2.3. Prototype and Design

In the scope of this thesis, a prototype version of the app will be developed to demonstrate the basic functionality as well as the look and feel of the platform's mobile version. There are many different approaches to prototyping, the two main categories of which being rapid prototyping and evolutionary prototyping. In rapid prototyping (also called "throwaway prototyping"), the goal is to quickly produce a simple working version to gather feedback on the requirements and functionality. The code will eventually be fully discarded and will not become part of the codebase for the actual release product. In evolutionary prototyping, the core of a system is developed thoroughly to outline the most basic functionality. The system will then be constantly refined and extended with new features. The difference here is that the prototype code will eventually turn into the final system instead of being thrown away.
As the thematic priority of this project is set on the design of user interface and user experience, a rapid prototyping approach focussing on front end development and usability optimization will be applied. In order to save the time and effort that it would undoubtedly take to build a functioning back end while still being able work with real data instead of just mock objects, the app will hook into a back end based on Parse, a BaaS provider that offers an extensive free plan. Using this solution, a full

data model will be created with the associated database hosted in the cloud. Adding, modifying and deleting data as well as running queries to fetch data will be possible using a RESTful API provided by Parse.

As outlined above, the rapid prototyping approach will cause the prototype to be discarded before development on the actual production version commences. Since the prototype is intended to demonstrate the system's functionality regardless of the underlying technology, it is acceptable and even advisable to develop the prototype using a language that is different from the one chosen for the production version. This will prevent developers to give into the temptation of re-using chunks of code from the prototype for the actual development and ensure that the prototype will really be "thrown away".

The prototype for this project will be developed as a native iOS application using Objective-C and the Cocoa Touch framework. This will allow for a relatively fast generation of visible and functional output without the need of tying together various frameworks to support different platforms. The main purpose of the prototype is to demonstrate the overall interaction concept and show how the application will function and feel from a user perspective. It will implement core functionality and set its focus on creating a rich and intuitive user experience. Internal aspects that would be crucial to an actual production version but are mostly invisible to the user, such as performance optimization, thorough exception handling or security measures will be largely disregarded. The goal is to create a prototype that is functional and polished to the point of being ready to be handed to a group of test users and having them verify that the basic idea is conveyed in a clear way, that the app is easy and fun to use and its underlying core user flows are coherent. With the prototype, test users will be able to create their own accounts and log into the system, browse through existing offers and requests or make specific searches, contact and be contacted by other users to engange in a deal with them and create, edit or delete offers and requests of their own.

In addition to the mobile app prototype, the structure of the web platform will be drafted in the form of wireframes with one sample screen being designed as a mock-up to demonstrate how the web version could look like. All graphics and assets required for the mobile app as well as the visual design for the website will be created using Adobe Photoshop and Illustrator.

4. Specification

4.1. Global navigation

The website will have a navigation bar at the very top of the site. This bar will be a global reference point that will be available on every page and always look the same. Users will be able to access different sections of the platform through this bar at all times, no matter how many levels deep they might have been navigating through sub-pages. The navigation bar will consist of the following elements:

- Menu button that opens up a menu with the following sections:
    - Open Deals
    - Messages
    - Account Balance
    - My Listings
        - My offers
        - My requests
    - Profile
    - Settings
    - Sign Out
- Message icon for quick access to inbox, visually indicates if there are unread messages (and how many)
- Handshake icon that will take users to their currently open deals
- Add icon that will allow users to quickly create a new offer or request
- Help link that opens a drop-down menu with links to global FAQs and some context-sensitive help
- Icons to maximize and minimze fonts and switch to black-white mode and back
- Button to browse categories
- Search bar

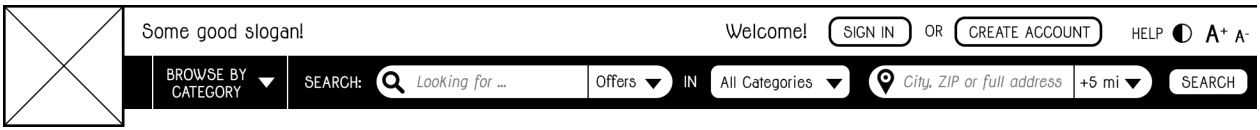


Fig. 4-1: Navigation bar for web platform, logged out state

Fig. 4-2: Navigation bar for web platform, logged in state with menu expanded

The icons for messages, open deals, adding offers/requests and help will have a tooltip-like mouseover effect indicating their respective functions.

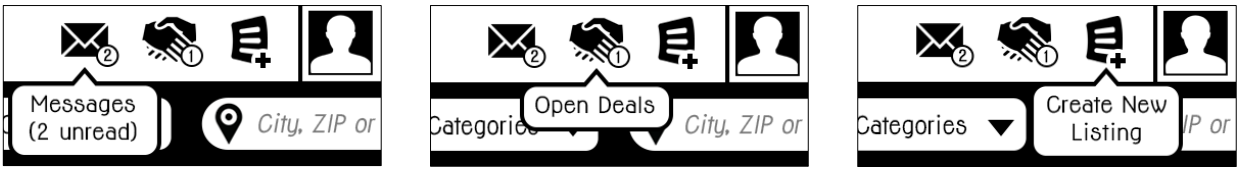


Fig. 4-3: Navigation bar icons with mouseover

The mobile application will follow a different navigation approach. The global reference point will be a navigation bar at the top that lets users know where they currently are. In top-level screens, the navigation bar will display a menu button on the top left. Tapping this button will make the current view swipe to the right side until only about 1/3 of its original width are visible, revealing an underlying menu containing the same elements as the website's menu specified above, additionally a search and a browse element. Tapping the menu button again will make the current screen swipe back in, while tapping any of the elements in the menu will change the current screen according to the user's selection before automatically swiping it back in.

In addition to the menu button on the top left, the menu can be opened and closed at any point by a simple swipe gesture anywhere on the screen. This will allow users to open the menu whenever they need it, even if the screen they're currently viewing is one or more levels down in the navigation hierachy, in which case the menu button will have been replaced by a back button to give users the opportunity to navigate back up in the hierachy.

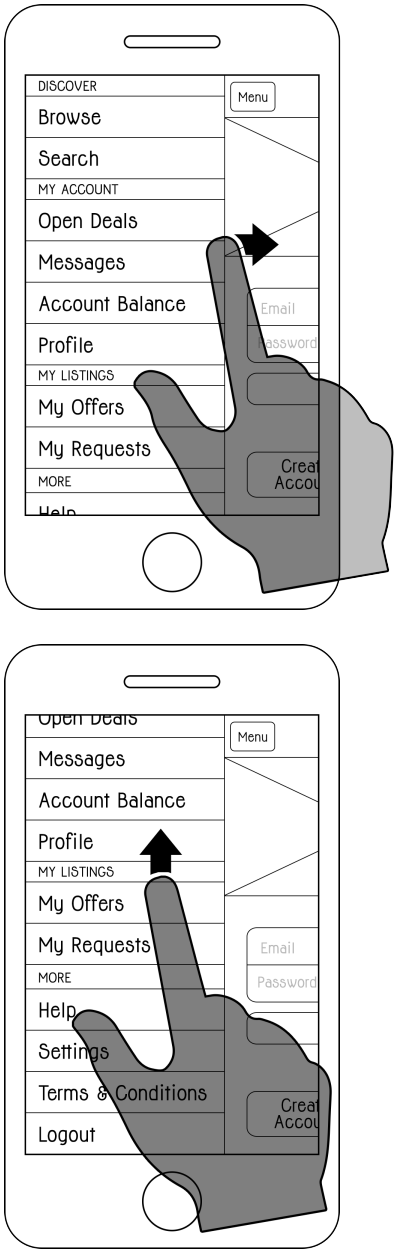


Fig. 4-4: Hidden menu in mobile app

4.2. First-time users

People who visit the website or open the app for the first time will have the possibility to learn about the platform and find out what kind of offers and requests are listed without having to sign up. On the website, new users will be greeted with a welcome screen that concisely presents the platform's vision and includes two prominent buttons: one for creating an account and one that takes them to a section where they can read about how exactly the system works in greater detail.
Even without having an account yet, people will be able to browse through offers and requests and make specific searches in order to decide if it's worth for them to sign up for the platform. While listings will be generally open to be viewed by non-members, any further actions – contacting another user, viewing somebody's profile or posting an offer or request of their own – will require them to become a member. A section within the global naviation element will give them the opportunity to create an account from whatever page on the website they might currently be viewing. In addition to that, the attempt to take certain actions like clicking on a user picture or clicking the button intended for contacting the user who belongs to a request or offer will prompt a "Create account or sign in" overlay with the notice that they have to be a member in order to do continue

Users opening up the mobile app for the first time will be presented a sign-in screen. This screen will also include buttons to create an account and to learn more about the platform. People who use the app are not required to ever use the website. All features available on the website will be realized in the mobile application as well. A global menu will give first-timers access to browse and search functionality without signing in, while the attempt to tap on sections reserved for registered members will trigger a modal pop-up asking the user to create an account or sign in. Once a user successfully signed in, they will remain signed in on the device, even in they close and re-open the app. They may choose to sign out by selecting the "Sign Out" option in the menu.

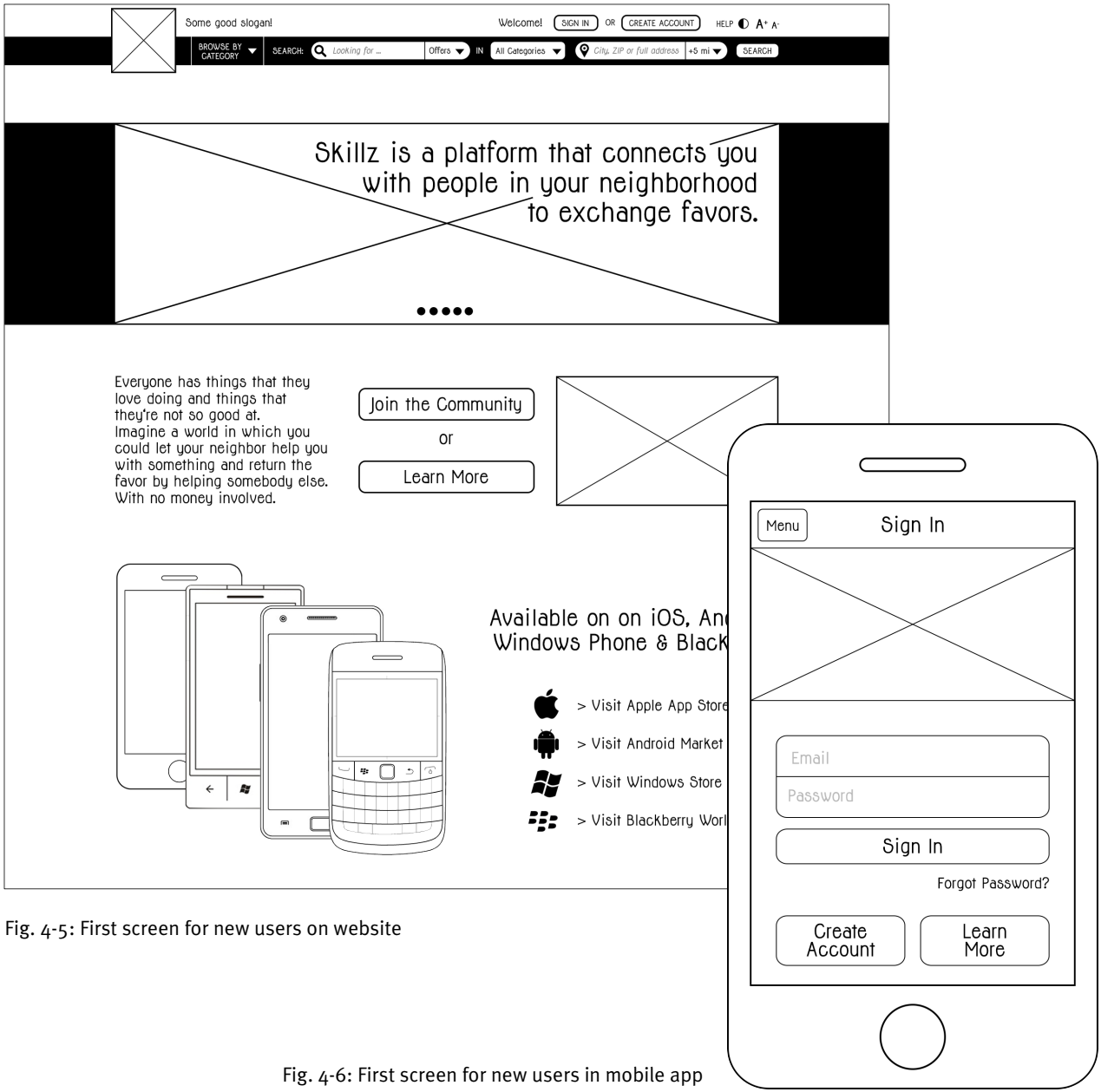


Fig. 4-5: First screen for new users on website

Fig. 4-6: First screen for new users in mobile app

4.3. Creating an account

To create an account, people have to enter their Email address and choose a password. Users will have to verify their password a second time to avoid typos. In addition to Email address and password, which will serve as their sign in credentials, users have to provide their first and last name, their ZIP code and the state they live in. There are no username aliases – users are identified by their Email address and will be displayed to other users with only their first name. To successfully create an account, users will have to accept the platform's terms and conditions, which will be available through a link on the create account page and also through a section in the menu (in the mobile app) and a link at the bottom of every page of the website.

4.4. Start screen

Once users signs in, they will be presented with a dashboard-like start screen that displays key information: Unread messages, open deals and account balance.

The section displaying the user's account balance will offer some quick links depending on their balance. If the balance is significantly positive, the start screen will show buttons to create a request or browse offers. If the balance is significantly negative, the screen will show links for creating an offer and browsing requests.

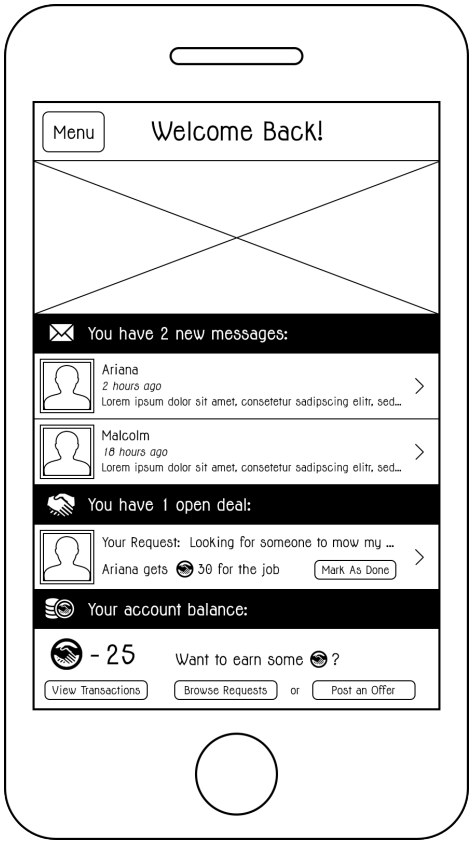


Fig. 4-7: Start screen in mobile app

4.5. User profile

In their personal profile section, users will have the option to personalize their profile and edit their private information. Initially, their profile picture will be displayed with a placeholder image. Users may upload a photo of themselves and exchange existing profile pictures. They have the opportunity to add a personalized description to their profile to tell other members a little bit about themselves. They will also be encouraged to list their personal skills and talents on their profile. Those listed skills will be used to display relevant members within search or browse results even if they don't have active offers listed. This feature will be explained in further detail on p. 56 of the "Browsing and Searching" section.

Furthermore, the profile page will show a little review graphic displaying the user's average review score based on a star-system of 0 to 5 stars and also indicating the number of reviews. Initially, the 3 latest reviews will be displayed on the user profile with a "Show More" button that will expand the view and display all received reviews.

A user's public profile page, when viewed by another member, will consist of the upper part (name, photo, average rating, description and skills) and the review part. If the member viewing the user's profile has an engaged in a deal with that user, the profile page will also display their telephone number.

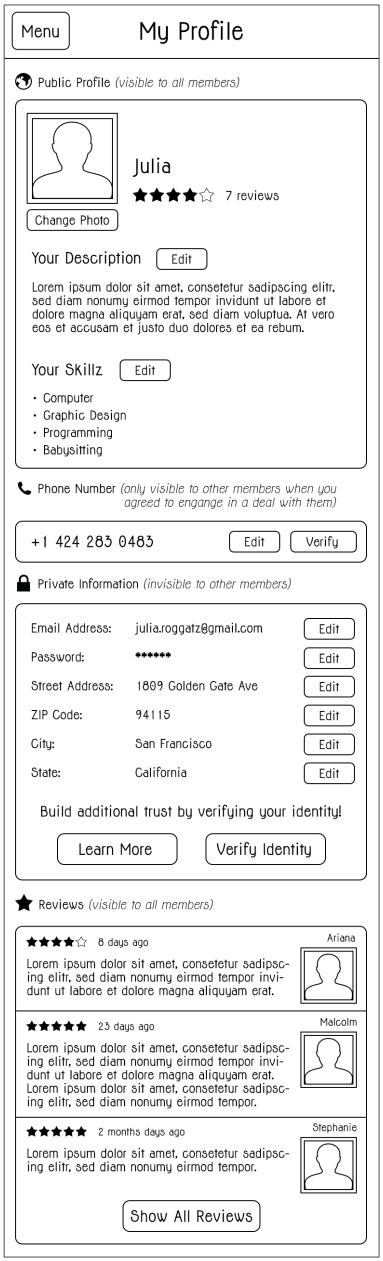


Fig. 4-8: User profile in mobile app, inside scroll view

4.6. Posting and managing offers and requests

Members can create their own offers and requests and post them on the platform. They will be guided through the process in a step-by-step dialogue.

In the first step, they have to select a category for their listing. A comprehensive list of categories will be offered to them, with some categories being split into more detailed sub-categories to choose from. In case users don't find a fitting category, they may choose the "Miscellaneous" category but will also be offered the option of suggesting a new category. If a category gets suggested several times by different users, it will be added to the list of categories.
In the second step, users will be asked to enter a title for their listing and add a text that describes the offer or request in greater detail.
The third step requires the user to specify location details about their listing. Initially, users may chose from 3 options: "I'm willing to go somewhere else", "People have to come to me" or "This can be done remotely". If they select the first option, they will be asked to specify how far they're willing to travel. They may choose to limit the listing to their own ZIP code area, specify a location and a maximum distance they would be willing to travel from this location or decide to keep the travel distance open to negotiation. If they select the second option ("People have to come to me"), they will be asked to specify their address. However, an annotation will ensure them that their address will only be used to display the listing's approximate location on a map and will only be disclosed to another user if both users have agreed to engage in a deal with each other.
In the next step, users have the option of specifying what they want to be paid to perform the task (if they're posting an offer) or what they're willing to pay someone to perform the task (if they're posting a request). They may choose from an hour-based rate or a task-based rate and also have the option to keep the compensation open for negotiation.

Specification

In the last step, users may specify a time frame for their listing. This would make sense for users who post requests like babysitting on a specific day or similar tasks that rely on a certain time frame. Instead of setting a complete time frame with start and end times, users may also just set an end time on their listing if the timing is generally flexible but the task will have to be done by a certain deadline. This will cause the listing to be automatically deactivated after the end time has passed. The timing specification is generally optional, so users may also choose to keep their task timing flexible.

There are two exceptions that will slightly change the user flow of posting a listing: Ride shares and accomodation. Users who choose the ride share category will not enter a title for their listing but instead specify the start and end location and the number of available seats for the ride they offer or need. If they're offering a ride, they will be asked to provide more detailed information about the exact start address (or may offer to pick up passengers) in the description section. Instead of choosing whether or not to specify a time frame, they will be asked to specify the take-off date time (if they're offering a ride) or take-off date and time frame (if they're looking for a ride). They have the option to set a fixed price on the ride or leave the compensation up for negotiation.
Similarly, users who choose to post an accommodation offer will be asked to specify the property's location and indicate how many people it can accommodate. They will be able to add photos of the property/room they offer. Furthermore, they will have the opportunity to set one or more time frames during which the the property will be available. Users who post an accommodation offer will have to specify when, where and for how many people they are seeking accommodation.

After having entered all necessary information, users will be presented a preview of the listing to see how it would be displayed to others and double-check all information. If they're happy with everything, they can post the listing right away or save it to be activated later.

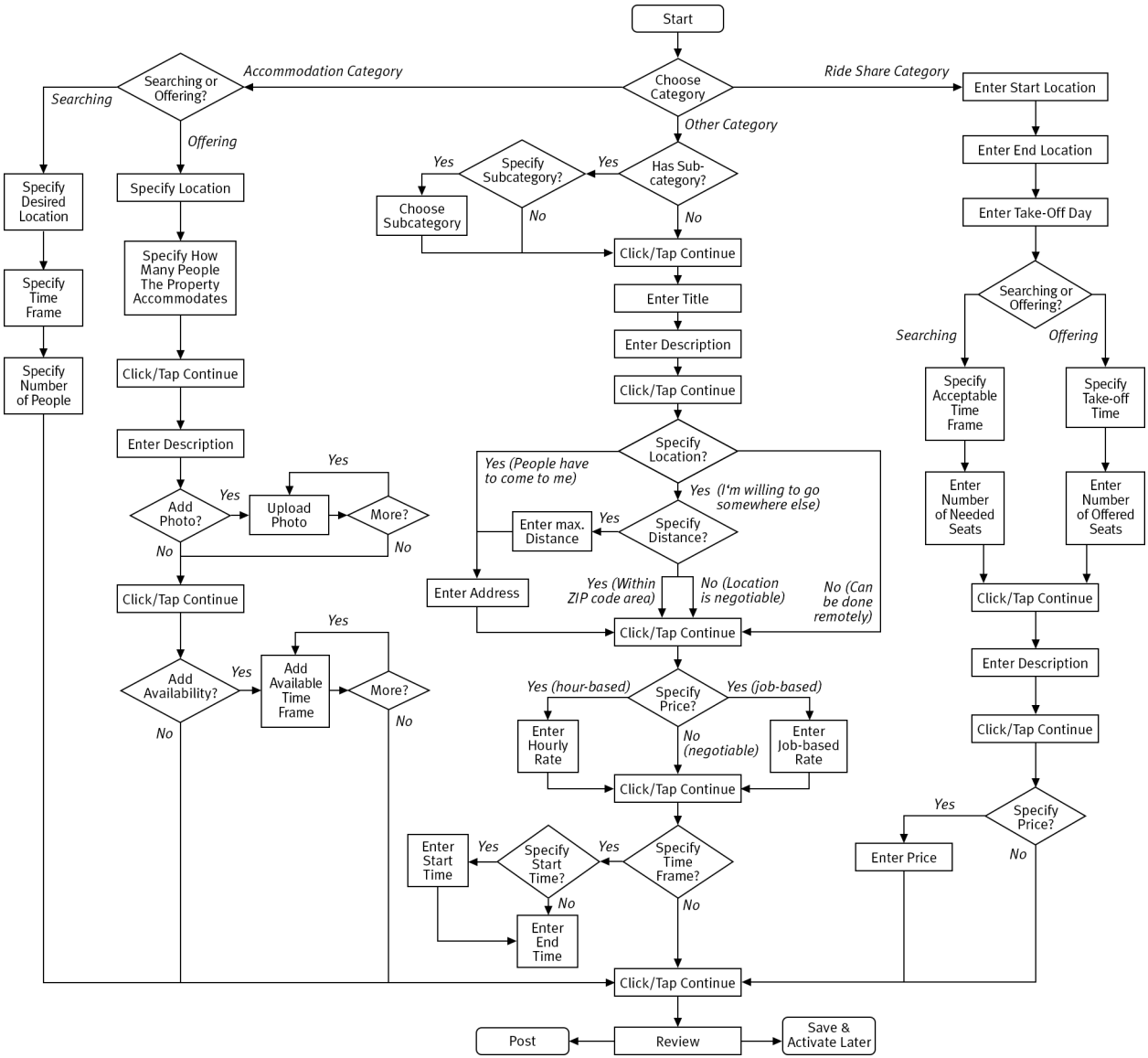


Fig. 4-9: User flow of posting a new listing

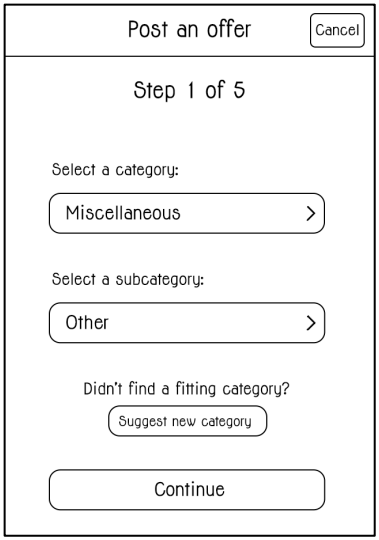


Fig. 4-10: Post an offer, Step 1

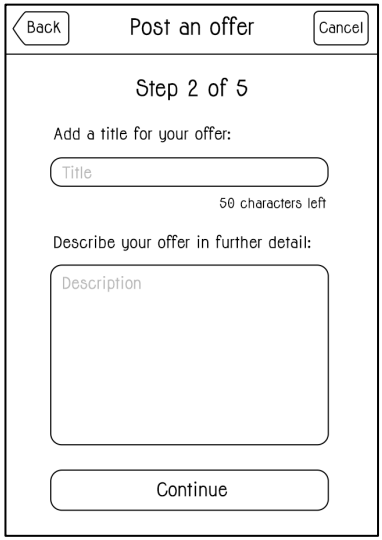


Fig. 4-11 Post an offer, Step 2

Fig. 4-12: Post an offer, Step 3

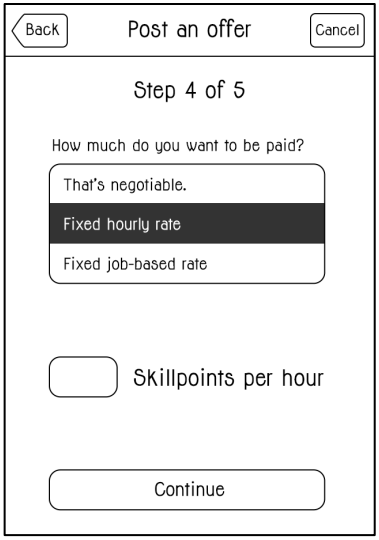


Fig. 4-13: Post an offer, Step 4

Fig. 4-14 Post an offer, Step 5

Fig. 4-15: Post an offer, Review

The sections "My Offers" and "My Requests" shows users a list with all their current listings and gives them the option to edit or delete them. Listings also have a switch to deactivate/activate them, giving users the opportunity to temporarily pause a listing without having to delete and recreate it when they want that same listing back online.
On the mobile device, user's listings will be cached on the device, allowing for offline creation, modification and deletion of listings with the changes being updated on the server once the device is back online.

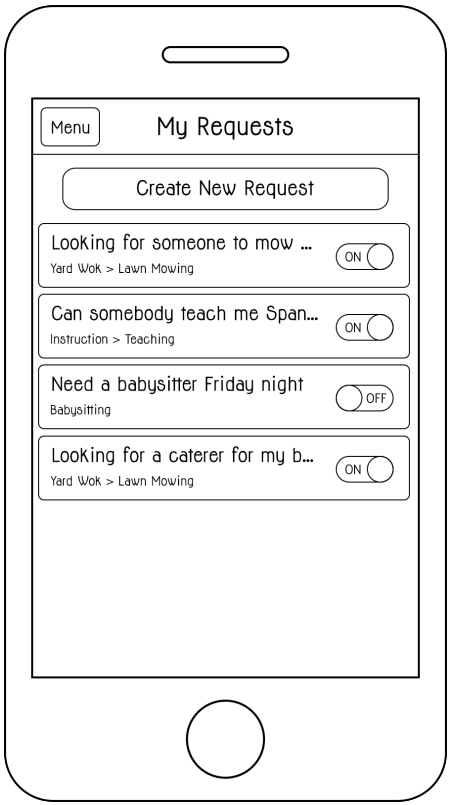
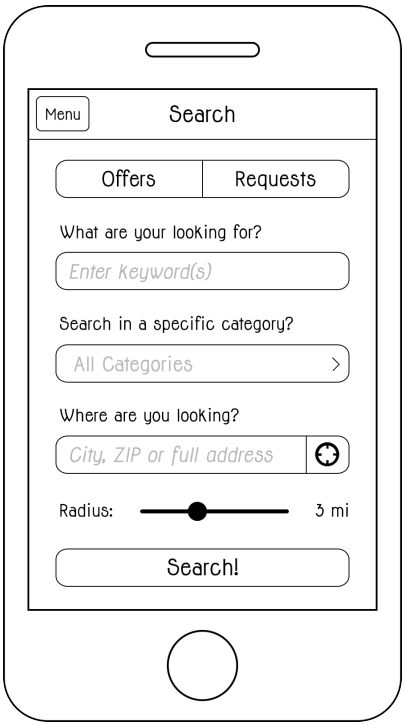


Fig. 4-16: List of a user's requests, mobile app

4.7. Browsing and Searching

Users have two different options of viewing active listings: Browsing through categories or making a concrete search. They can choose to view all currently open listings by selecting "All Categories" or narrow down the selection by choosing a specific category and, if existing, an associated subcategory. The search section on the website's top navigation bar as well as the search screen on the mobile device takes the following parameters:



- What type of listing to search for (offers or requests)
- Search keywords – entered in a text field
- Category (optional) – drop-down menu on the website, picker view in the mobile app
- Location (optional) – entered in a text field, accepts cities, ZIP codes or full addresses, will suggest cities based on the first few letters entered by the user, for the mobile version it will include a button to choose the user's current GPS location
- Search radius (only active if location specified) – drop-down menu on the website, slider element in the mobile app

Fig. 4-17: Search screen, mobile app

In the web version, all results will be displayed in a preview-style list. Each result will show the title of the listing, the picture, name and average rating of the user who belongs to the listing and the price (if specified). A menu offering sort and filter options will be displayed on the left side of the page. Sort options include sorting the results by user rating, by price or by distance from a location specified in the menu section. If the user's browser and computer support location services, the option of allowing the site to acces the user's current location will be available as well. Users will also have the opportunity to filter results by minimum user rating, price range and area radius, which will remove all listings that don't match the specified parameters from the results list. Listings that have don't have certain parameters specified (e.g. keeping the price or location negotiable) will be kept in the list but

displayed at the very bottom. The filter menu will also include a small map view that shows the currently displayed listings as points on a map. Instead of filtering by a specified address and radius, users will also have the option of dragging and zooming the map area and having the result list updated according to the current map section.

Clicking on a listing will bring users to its detail view, where all information available about the listing will be presented to the user, along with the options to directly contact the user who posted the listing or to navigate to their profile in order to find out more about them and read their reviews.

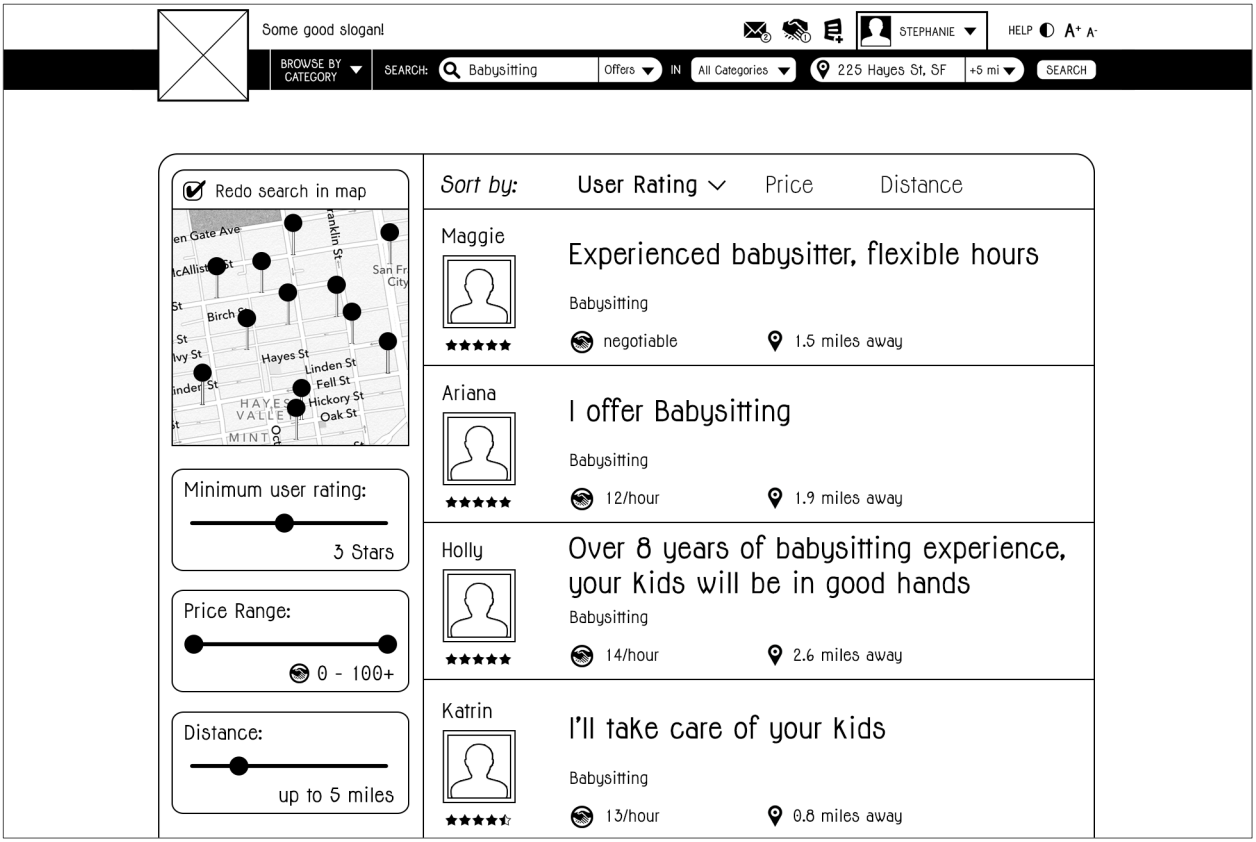


Fig 4-18: Search results screen on the website

In the mobile app, the user will be able to switch between map view and list view by tapping a segmented button control on the navigation bar above the search result view. Search results in the map view will be displayed as pins, which can be tapped to display a little info view displaying the listing's title, the user picture, name and rating and a button to open up the listing's detail page.

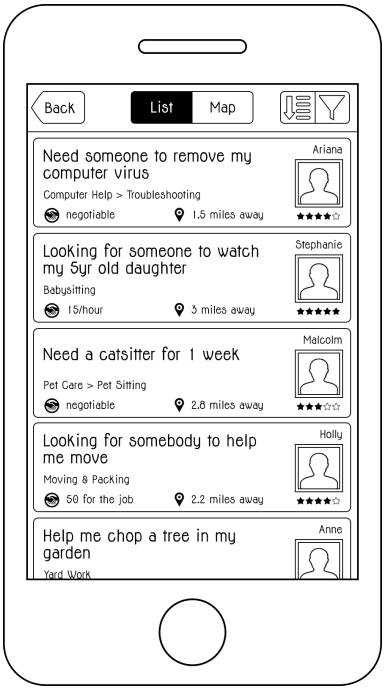


Fig 4-19: Search results in mobile app (list view)

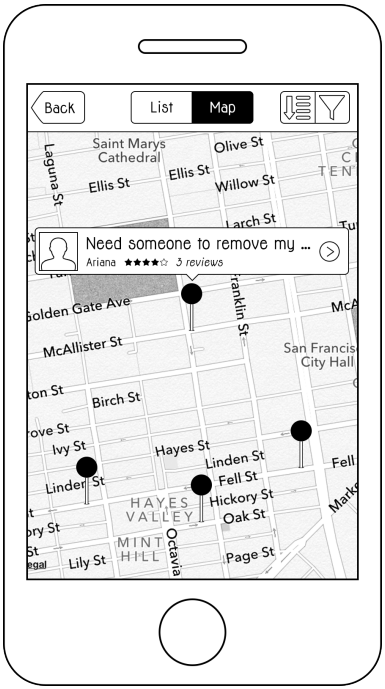


Fig 4-20: Search results in mobile app (map view)

Sort and filter menus will be available through buttons on the top right of the navigation bar. Similarly to the global menu on the left side, the sort and filter menus are hidden on the right side behind the result view and can be made visible by tapping the sort or filter button on the right side of the navigation bar above the search result view.

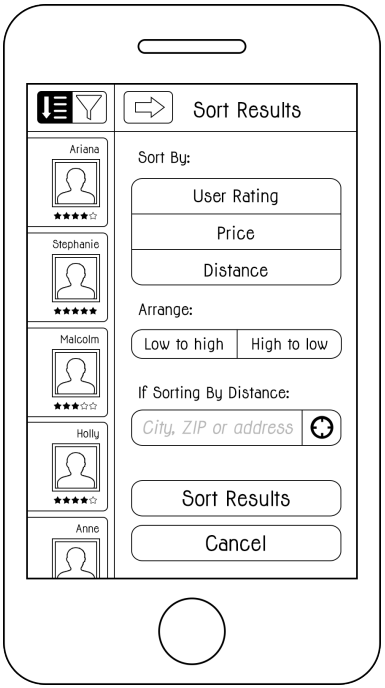


Fig 4-21: Sort menu in mobile app

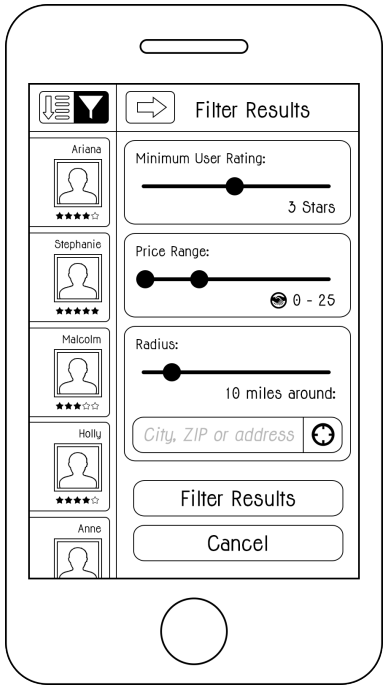


Fig 4-22: Filter menu in mobile app

Users who search or browse offers will be presented an additional section at the end of their search results page, listing users whose skills and talents match the search parameters, even if those users currently have no active offers. For example, if a user searches for somebody to fix a flat tire on their bike but none of the search results seemed like the right fit (or there were no search results at all), this section will show them users in their specified area who have mechanical skills listed on their profile page. The user with the flat tire may choose to proactively contact one of these users and ask them for help, attaching a deal that the contacted user can choose to accept or decline.

4.8. Contacting members and engaging in deals

Users may get in touch with other members either through the detail page of the listing they're currently viewing or through another member's profile page. If they're contacting a member with the intention of engaging in a deal with them, they may attach a deal suggestion and "stretch out their hand" to the other user. A deal suggestion consits of a concrete offer regarding the compensation of the task. If the listing has a price specified, this will be the default setting for the attached deal. However, members can modify the deal's price and thereby try to bargain with the other user. If the listing doesn't have a price specified, users will have to come up with a price for the deal anyway. Attaching a deal suggestion to the contact message is optional – users may also choose to send a simple message to clarify details or make a non-binding suggestion.

Fig. 4-23: Contacting a user

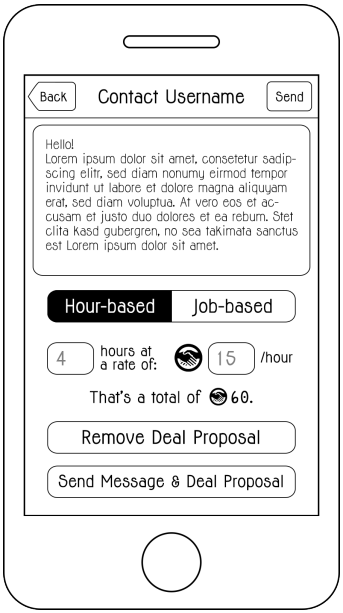


Fig. 4-24: Attaching a deal (hour-based)

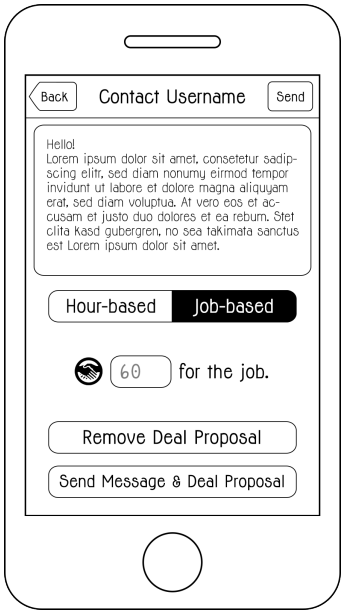


Fig. 4-25: Attaching a deal (job-based)

A user receiving a deal suggestion may choose to make a counter-suggestion with a modified price specification or simply decline the suggestion. If they want to accept the suggested deal, they will virtually shake the hand that was stetched out to them, resulting in a binding agreement among both parties for the transaction to take place. The stretching out and shaking of hands are metaphors that will be visually incorporated into the user flow to illustrate the process of engaging in a deal. Once a deal is sealed, both members will have the option of making a donation for people in need (see p. 34 for more explanation about this).

Messages will be cached on the device to be available at all times, even in offline mode. The "My Messages" list view will group message threads by users.

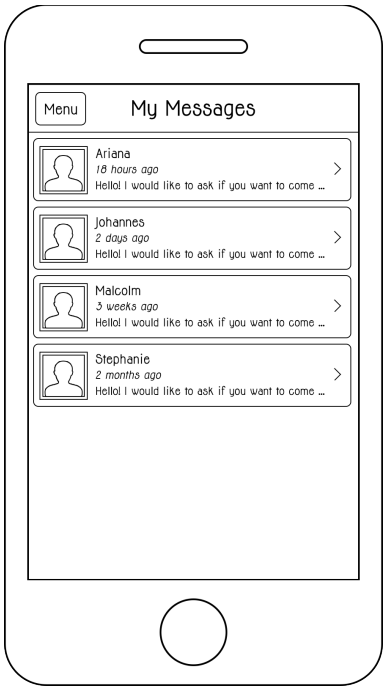


Fig. 4-27: List view of all message threads

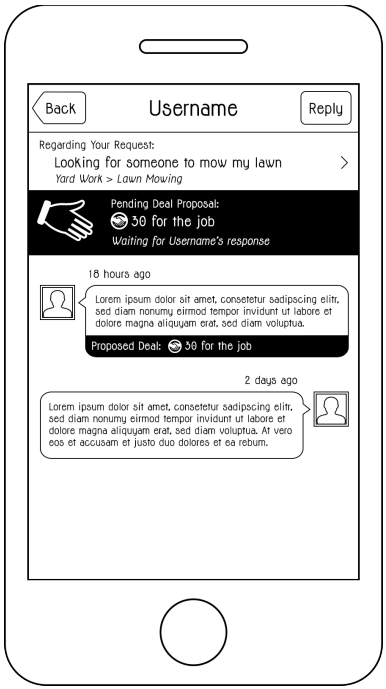


Fig. 4-28: Sample message thread with with attached deal proposal

4.9. Open deals, payment flow and review process

When two people engange in a deal, the amount of points that both parties agreed on will be debited from the account of the user receiving the service. However, the user performing the services will not immediately receive those points. Instead, the points will be withheld by the system until the deal is marked as done.

Open deals will be available for reference in the "Open Deals" section for both parties. Through this section, users will be able to quickly view relevant details about those deals, for example the address of the other user (if applicable) or their phone number. They may contact the other person to clarify details and logistics in order to get the job done. Both parties will see a "Mark as Done" button, a "Cancel Deal" button and a "Report Problem" button. If the user who received the services marks it as done, this will automatically mark it as done for the other user and the points will be credited to their account. If the user who performed the service hits the "Done" button first, the other user will be notified about this and asked to confirm that the deal is done.

Once a deal is marked as done, both users will be asked to review each other. A review consists of a star-rating (0 to 5 stars), a public review text open to the community that will be displayed on the user's profile page and optionally a private message to the other user where further feedback or just a simple "Thank You" can be exchanged. Users will not be able to see the review they got before they haven't submitted their own review for the other user.

The "Cancel Deal" button allows users to cancel a deal if necessary. There could always be unforeseeable circumstances (for example illness) that require a deal to be cancelled. If one of both parties hits the cancel button, they will be asked to attach a message explaining the other user why they have to cancel. A notifiation including this message will be send to the other user, who has to accept the cancellation in order for the points to be booked back to the account that they were originally debited from.

There are various things that can potentially go wrong in the process: Somebody doesn't respond after a deal has been agreed on, delivers unacceptable work or doesn't show up at all, doesn't mark the deal as done or even tries to cancel the deal after the work has already been performed. That's what the "Report Problem" button is for. Users who hit this button will first be asked if they have already tried to solve the problem on their own by contacting the other user. If they haven't, they will be encouraged to first write a message to the other user to see if they can find an agreement. If they still want to report the problem, they will get the opportunity to contact customer service to explain the problem and seek assistance in solving it.

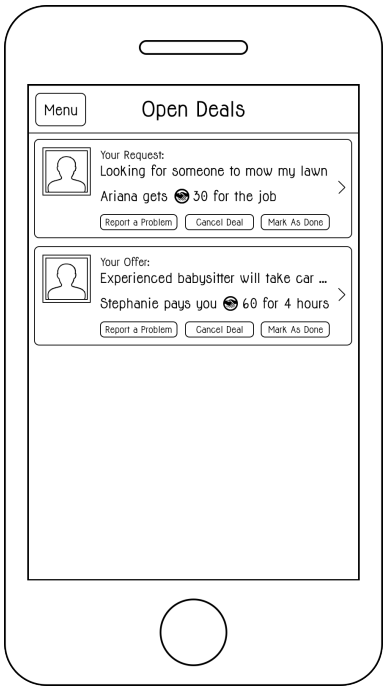


Fig. 4-27: List view of all message threads

4.10. Security and trust

To avoid fake profiles, users will be required to verify their Email address before they are able to sign in. After a user successfully created a new account, an Email will be sent to the address they provided with a link to verify the address. After clicking this link, their account will be activated, giving them full access to all of the platform's features.

To build additional trust, users have the opportunity of verifying their phone number. If they provided a phone number in their personal profile settings, they will see a button labeled "Verify Phone Number". Upon clicking that button, a text message will be sent to their phone with a confirmation code. If they enter this code on their profile page, their phone number will be marked as verified.

Through cooperation with a third-party service called Virtrue [Virtrue, 2013], users will also have the opportunity to have their identity verified through a link in their personal profile settings. Virtrue uses a set of algorithms that analyze user's online data to verify if they really are who they say they are. If a user has successfully verified their phone number and/or identity, this will be indicated on their public profile – the data itself will be kept private.

4.11. Notification settings and subscriptions

By default, users will be notified about incoming messages through an Email sent to the provided Email address. On the mobile device, people will also have the opportunity to enable push notifications in order to be instantly informed about messages.

Users will also have the opportunity to subscribe to newly posted offers and requests within their area. They may do so by specifying one or more categories and choosing to either subscribe to new listings matching those categories within their ZIP code area or providing a specific location and radius around this location. Whenever a new listing matching the critea is posted, they will be instantly notified via Email and, if they're using the mobile app and have the service activated, will receive a push notifcation on their device.

4.12. Accessibility

To help users who suffer from weak eyesight or visual impairment, both the website and the mobile app will offer the opportunity of enlarging fonts and switching to a black-white mode to support vision through increased contrast.

All elements, including pictures, buttons and icons, will be supplemented with metadata and accessibility labels to support blind users who rely on screen readers.

5. Implementation

5.1. Backend integration

The iOS prototype runs on a backend provided by Parse [Parse, 2013]. All data is stored and managed in the Cloud and accessed through the interface provided by the Parse SDK. The following sections will give a brief overview about how the framework is integrated and used and explain some its most useful features.

5.1.1. Connecting to the backend

To hook up the app to the Parse backend, the following steps had to be completed:

- Sign up for Parse and create an app within the Parse client
- Download the Parse SDK and include it in the Xcode project
- Add additional iOS libraries required for the Parse framework to work properly
- Register the client with the Parse app by running the following code upon application lauch:

```
[Parse setApplicationId:@"ltQB4UH8RtuQ84RTJOWg16IfJh0fojlzrYEbwwUr"
              clientKey:@"Hi4lrAsfSq0iWDi6npeYMlgrmL65l5iWFtoKl5Ef"];
```

5.1.2. Storing Data and Building the Data Model

Once conncected to the backend, the client has access to all stored data and is able to create, retrieve, modify and delete datasets. Data objects are represented by the PFObject class. Properties are stored as key-value pairs. Supported value types are: String, Number, Boolean, Date, File,

GeoPoint, Array, Object, Pointer and Relation. As indicated through the Pointer and Relation type, the framework supports relational structures and provides a convenient interface for relational queries.

The data model can be created through a user interface on the Parse website. However, data management within the Parse framework is schemaless, which means that it is not required to specify the schema of a class or even the class itself before creating instances of it. If an object with a certain class name is sent to the server and there are currently no other stored objects of that class, Parse will lazily create a new class for the received object.

The following code snippet illustrates the process of adding an object to the persistent storage:

```
PFObject* messageObject = [PFObject objectWithClassName:@"Message"];
[messageObject setValue:@"Hello" forKey:@"messageText"];
[messageObject save];
```

In addition to the simple saving function, the frameworks offers a variety of additional functions for saving objects such as saving in background, running a callback function or block upon server response or, if the device is currently offline, caching the object and saving it as soon as network connection is restored.

5.1.3. User Creation and Management

The framework also offers extensive functionality to create and manage user objects. For this purpose, the PFUser object is used. User accounts can be created by sending a PFUser object with a username, an Email address and a password to the server. The server will verify the uniqueness of the Email address and username and send out an Email asking the new user to verify their Email address.

Once verified, user may sign in with their credentials. After successfully signing in, the user object will be cached and globally available through the [PFUser currentUser] function. Unless the client specifically signs out the current user by calling [PFUser logout], the user will remain signed in even after closing and re-opening the app.

5.1.4. Retrieving Data

To query for stored data, the PFQuery class is used. A query can retrieve a single object through the object's unique objectId property, or search for objects that match one or several specified conditions and return those objects in an array. The following example shows a simple query to retrieve offers that match a certain category.

```
PFQuery* query = [PFQuery queryWithClassName:@"Entry"];
[query whereKey:@"entryType" equalTo:@"offer"];
[query whereKey:@"category" equalTo:@"Babysitting"];
[query findObjectsInBackgroundWithBlock:^(NSArray *objects, NSError *error) {
   if (objects && [objects count] > 0) {
   // found something! Display the results to the user
   }
}
```

The PFQuery class offers a wide range of functions to constrain searches and sort and filter results. This was particularly helpful for implementing the search feature in which not only entry titles, but also their corresponding descriptions should be included in the search. For such complex queries, several subqueries can be linked with a disjunction using the following function:

```
+ (PFQuery *)orQueryWithSubqueries:(NSArray *)queries;
```

It turned out that the PFQuery class was also able to take care of filtering results by a given geolocation and a corresponding radius. This is exactly what was needed for the search feature if a user wants to limit their search to a certain area. The user input representing an address, ZIP code or city or a combination of those will be geocoded into latitude/longitude coordinates and, along with the user-specified search radius, can be set as a constraint for the query using the following function:

```
- (void)whereKey:(NSString *)key nearGeoPoint:(PFGeoPoint *)geopoint
                                  withinMiles:(double)maxDistance;
```

5.1.5. Push Notifications

The Parse Frameworks supports push notifications in a very easy-to-use way. Through the web interface of Parse, custom push notifications can be sent to all devices that installed the application (provided they gave permission to receive notifications), for example to inform users about an update. Additionally, the framework allows for client-side notifcations, a feature that is used within the app to let users know that they received a new message. This is easily achieved through the following code:

```
PFQuery *pushQuery = [PFInstallation query];
[pushQuery whereKey:@"user" equalTo:self.recipient];
PFPush *push = [[PFPush alloc] init];
[push setQuery:pushQuery];
[push setMessage:@"You received a new message!"];
[push sendPushInBackground];
```

5.2. Data Model

The data model is kept relatively simple. By default, all objects inherit from PFObject, which automatically assigns every newly created object a unique objectId, sets the createdDate property and updates the updatedAt property whenever the object is modified.

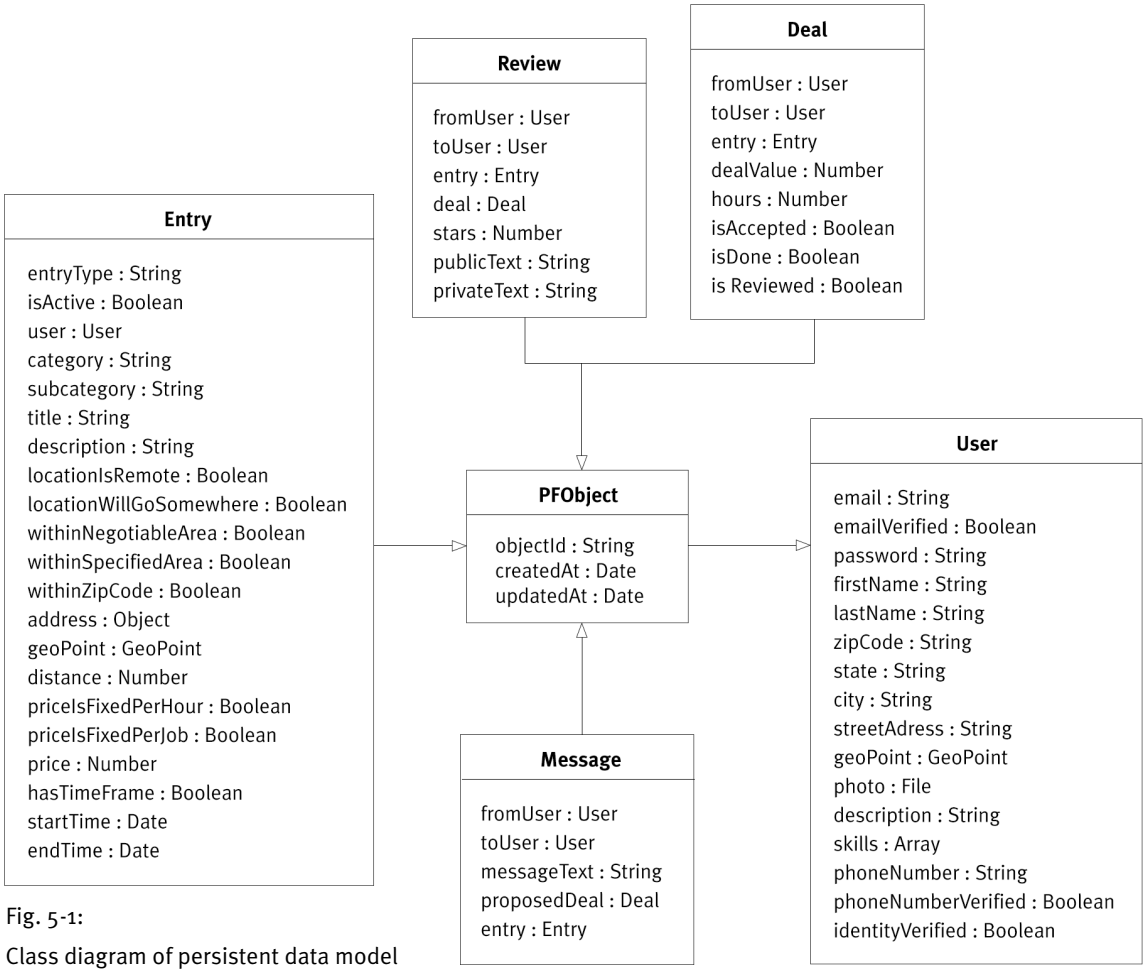


Fig. 5-1:
Class diagram of persistent data model

5.3. View Hierachy

The root of the app's navigation hierachy is represented by the SZNavigationController, a subclass of UIViewController containing two essential components: The hidden menu and a UINavigationController serving as the root for all added subviews. The hidden menu, represented by the SZMenuVC class, is designed as a singleton and subclasses UIViewController. It contains a UITableView to layout and skin menu cells and arrange them into sections. The menu uses delegatation, a common design pattern in iOS applications, to inform its delegate when a user selected a cell. Menu cells are assigned a corresponding class name that is passed on to the menu's assigned delegate to perform the view switch requested by the user. The menu protocol declares the the following function that its delegate has to implement with the appropriate action:

```
@protocol SZMenuVCDelegate <NSObject>
- (void)menu:(SZMenuVC*)menu switchToViewControllerWithClassName:(NSString*)className;
@end
```

The SZNavigationController encapsulates the core navigation logic and serves as the delegate for the SZMenuVC. When it receives a message from the menu to that the user requested to navigate to a different section of the app, it creates an instance of the according class, sets this class as the root view controller of its main view container and hides the menu:

```
UIViewController* vc = [[NSClassFromString(className) alloc] init];
[self.mainViewController setViewControllers:[NSArray arrayWithObject:vc]];
[self slideInMainViewAnimated:YES navigationType:SZNavigationMenu];
```

View Controllers that serve as the root of one of the app's sections will display a "Menu" button on the top left of their navigation bar. Tapping this button will slide the current view to the right of the

screen, revealing the underlying menu. Tapping the button again moves the view back into place and hides the menu. However, once a user navigates one or more levels down in a section's view hierachy, the "Menu" button will be replaced by a "Back" button. This is one of the most common interaction paradigms in iOS applications and must be preserved in order to ensure proper usability. On the other hand, it would be inacceptable to require users to navigate all the way back to the top of a section's view hierachy to access the menu button.
Therefore, the menu is made available globally through a swipe gesture. By swiping across the screen to the right, users can make the menu appear at any given time, regardless of how deep in the view hierachy they currently are. This is realized through a UIGestureRecognizer set on the SZNavigationController's main view container. The gesture recognizer detects swipe gestures and informs the SZNavigationController, which takes control of sliding the main view in or out accordingly.

The main challenge with implementing the menu behaviour was the special case of modal view controllers. Normally, new views are presented to the user as a new level in the view hierachy, meaning they will be pushed on the current view controller stack and visually slide in from the right. If a user taps the back button, the view will slide out to the left and popped from the view controller stack, bringing the user back up in the view hierachy. In some cases, users should be presented a view controller that is not really part of the current view hierachy but instead encapsules its own view hierachy for which it serves as the root view. Often this applies to distinct user flows that require a "Cancel" option.
With regard to this project, a good example for this scenario is the "My Listings" section. Assuming the user is currently viewing a list of their current offers, selecting one of those offers will open up the offer's detail view. The user may go back to the list view by simply tapping the "Back" button at the top left of the navigation bar. In the list view, a button titled "Create New Offer" lets the user create a new offer. This action brings up the view for creating a new offer, which consists of several steps, all

of which are represented by a separate view controller that will be pushed onto the view stack. Assuming the user already reached step 5 before changing their mind and wanting to cancel the offer creation, it would require the user to tap the "Back" button 5 times before reaching the list view again. This is where modal view controllers come into play.
Modal view controllers typically slide in from the bottom, leaving the underlying view controller in place and introducing their own view hierachy. They usually display a "Cancel" button at the top right of their navigation bar which lets the user dismiss the currently displayed view hierachy and instanly go back to the underlying view hierachy. The user flow for creating a new offer or request is implemented in such a way. While modal view controllers usually only allow for navigation within their own hierachy, in the special case of this project it is necessary to slightly vary from this standard to ensure a consistent user experience and keep the hidden menu accessible at all times. This could be realized by using an additional instance of the SZNavigationController class to serve as the container of modal view controllers and offering those controllers the same swipe functionality.

Since the menu class is handled as a singleton, setting its delegate to a new SZNavigationController instance and assigning its view to that instance resulted in the issue that once a modal view controller was dismissed, the underlying view controllers were suddenly missing the menu. After many failed attempts to solve this issue, which always resulted in some kind of odd behaviour by producing unexpected side effects, the solution that actually worked was so simple that it was almost too good to be true. The UIViewController class has a built-in function that is called whenever a controller's view appears on the screen, regardless of whether it is displayed for the first time, re-appearing after being hidden by another view controller or popping back up in the view hierachy. Overriding this method in the SZNavigationController class ensures that the menu view will always be hooked up to the SZNavigationController instance that is currently being displayed to the user:

```
- (void)viewDidAppear:(BOOL)animated {
      [self addChildViewController:[SZMenuVC sharedInstance]];
      [self.view insertSubview:[SZMenuVC sharedInstance].view atIndex:0];
      [[SZMenuVC sharedInstance] setDelegate:self];
}
```

5.4. Code Architecture

The code is generally designed to encapsulate logic into distinct modules and minimize unneccsesary interdependencies between classes. Every screen is represented by it's own UIViewController subclass which only holds references to the information it really needs.
To give a concrete example, the SZSearchVC implements the search screen in which the user can input keywords, select a category and specify a search area. The SZSearchVC class takes care of creating and laying out its user interface and handling the user input. The SZSearchResultsVC on the other hand is responsible for displaying the search results. It needs to know what to search for, but giving it a reference to the SZSearchVC would increase interdependency between those two classes. Therefore, the SZSearchVC takes care of validating the user input, wrapping up all search parameters into a PFQuery object and passing this query into the SZSearchResultsVC. For this purpose, the SZSearchResultsVC interface defines the following public initialization method:

```
- (id)initWithQuery:(PFQuery*)query;
```

The SZSearchResultsVC can now simply run the query and present the search results to the user without needing to know about the specific search parameters itself and without keeping a reference to the SZSearchVC class. Thus, the SZSearchResultsVC is designed flexibly enough to also be used within the browsing feature, in which the user only selects a specific category (or browses within all

categories at the same time) but doesn't add any further constraints to the query that is passed into the SZSearchResultsVC.

Sometimes objects need to be accessible throughout several different view controllers and may be modified by each of those controllers. A good example for this is the case in which a user wants to create a new listing. As mentioned before, this is done in several steps, all of which have their own logic encapsulated into a distinct view controller. Upon tapping the "Create New Offer" or "Create New Requst" button, a new instance of PFObject is created to store all the information that the user wants to specify for this listing. Instead of passing the object from one View Controller to the next, the object is assigned to the SZDataManager, which is designed as a singleton class and therefore globally accessible from everywhere in the code. Once a listing is complete and ready to be posted to the server, any class may call the following public function on the SZDataManager singleton:

```
- (void)saveCurrentEntry:(void(^)(BOOL finished))completionBlock;
```

The SZDataManager will dispatch a completion block when it has successfully saved the listing to let the calling class handle whatever it wants to do when the saving process is done. In this case this would be removing the spinning wheel that was indicating the ongoing saving process for the user and navigating back to the list view that is displaying all of the user's current offers or requests.

Outsourcing the actual saving process to the SZDataManager results in the benefit of having the saving function available in one single place. Therefore, when a user decides to edit for example only the title of a specific listing, which is done in Step 2 of 5, the saving function can be called directly from Step 2 without requiring the user to go through all of the other steps and without having to implement the save functionlity in every single of the five view controllers.

Data that needs to be globally available to all classes for convenience is stored in a class named SZGlobalConstants, which is included in the project's header prefix and therefore by default available in all classes without the need to explicitly import it. This class defines certain enumeration types that are used in different places of the code and offers a few class methods that will return values used for skinning the app. For examlpe, setting any random label used anywhere in the code to the app's specific font in a certain font weight and giving it the typical dark petrol color can be done like this:

```
[textLabel setFont:[SZGlobalConstants fontWithFontType:SZFontSemiBold size:16.0]];
[textLabel setTextColor:[SZGlobalConstants darkPetrol]];
```

Another convenience class that offers some more complex functionality used in different places of the app is the SZUtils class. It mainly takes care of simple tasks such as converting an NSDate object into an NSString that represents the date formatted in a humanly readable way, but also offers some more complex functionality like taking an array containing numbers from 0 to 5 that represents a user's review profile and returning a view that display's the average of those review scores in an typical "star view".

5.5. Custom UI Elements

There are several UI elements that used within different view controllers that should always have the same behaviour and look the same way. Sometimes they are just generic control elements offered by the iOS SDK that need to be skinned to integrate into the app's visual design, but in other cases they need to incorporate more complex control logic to store and process user input. For this purpose, creating custom classes that subclass either the UIView class or other specific UI classes was the right approach.

SZButton

The SZButton, a subclass of the UIButton class available in the UIKit of the iOS SDK, will generate a button that fits into the app's visual design with the following initialization method:

```
- (id)initWithColor:(SZButtonColor)color size:(SZButtonSize)size width:(CGFloat)width;
```

The color and type properties are defined by enumeration types to ensure consistency. Available button colors are petrol *(SZButtonColorPetrol)* and orange *(SZButtonColorOrange)*, available button sizes are heights of 24px *(SZButtonSizeSmall)*, 32px *(SZButtonSizeMedium)*, 40px *(SZButtonSizeLarge)* and 52px *(SZButtonSizeExtraLarge)*. The button width is fully flexible and specified by the width parameter.

SZForm

An example for a more complex UI element is the SZForm class. An SZForm instance is an abstracted component that lets users input data in different ways. It is displayed as a view element containing one or more fields in which the user can enter information. Each of these fields can be configured individually by using an SZFormFieldVO to customize the way this data is supplied. For instance, one field may be behaving like an ordinary text field that uses the default keyboard, the next field may be requiring a numeric value that uses a number pad instead of the keyboard to supply the input while another field is requesting the user to pick from a pre-defined set of options and uses a UIPickerView to let the user choose one of these options. A form configured in such a way would look like this:

Fig. 5-2: Form input, regular keyboard

Fig. 5-3: Form input, decimal pad

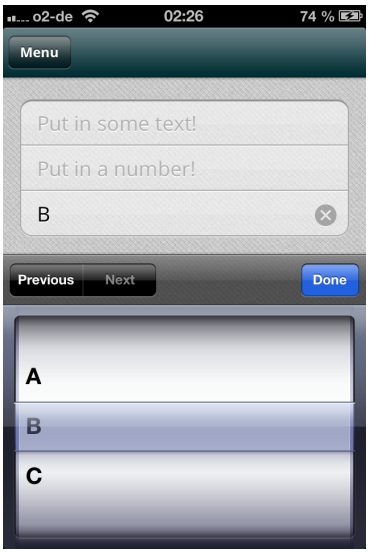


Fig. 5-4: Form input, picker view

The corresponding form component can be easily created using this code:

```
SZForm* form = [[SZForm alloc] initWithWidth:290.0];
SZFormFieldVO* textField = [SZFormFieldVO formFieldValueObjectForTextWithKey:@"text"
              placeHolderText:@"Put in some text!" keyboardType:UIKeyboardTypeDefault];
SZFormFieldVO* numberField = [SZFormFieldVO formFieldValueObjectForTextWithKey:
  @"number" placeHolderText:@"Put in a number!" keyboardType:UIKeyboardTypeDecimalPad];
SZFormFieldVO* pickerField = [SZFormFieldVO formFieldValueObjectForPickerWithKey:
                                      @"picker" placeHolderText:@"Pick!" pickerOptions:
                                    [NSArray arrayWithObjects:@"A", @"B", @"C", nil]];
[form addItem: textField showsClearButton:YES isLastItem:NO];
[form addItem: numberField showsClearButton:YES isLastItem:NO];
[form addItem: pickerField showsClearButton:YES isLastItem:YES];
```

Internally, the SZForm class takes care of various tasks such as retrieving and displaying the correct background view for the field depending on whether it is on top, in the middle or at the bottom of the form, configuring the text fields with their placeholders, input types and visual properties. With the help of an open source component called "BSKeyboardControls" [simonbs / BSKeyboardControls, 2013], it creates a toolbar for convenient switching between fields through "Previous" and "Next" buttons and dismissing the input view with the "Done" button. When selecting a field that is positioned in the lower part of the screen and would therefore be hidden by the input view, the SZForm also ensures that the screen will be scrolled up accordingly to make the respective field visible for the user. User inputs are stored in an NSDictionary (using the keys provided by the SZFormFieldVOs) that can be accessed through a public property by the class that is using the form. Additionally, the SZForm can be assigned a delegate and defines a protocol with various functions that delegates may implement if they wish to be informed about certain events, for example to validate input after a user tapped the "Done" button.

5.6. Caching on the device

While features like searching or browsing for listings obviously need a working network connection, there are certain sections of the app that wouldn't necessarily require the user to be online in order to be functional. Reading old messages, viewing one's profile page or accessing a list of one's own posted listings for example are examples of scenarios that would make sense to have available even in an offline environment. However, all data is stored in the cloud by default and therefore even trivial information such as the current user's name is only accessible when the device is connected to the internet. To solve this problem and offer users some useful offline functionality while at the same time increasing performance, certain data is cached on the device.

The SZDataManager class takes care of all caching tasks and uses a .plist file stored in the app's default documents directory to save data to the disk and structure it in a way that makes it easy to retrieve and convert back into a format that the code can process. Taking the example of caching received messages, locally stored message objects are stored as a nested dictionary. A message itself is represented by a dictionary using keys like "messageText" and "fromUser" or "toUser" to store values. The message dictionaries are in turn stored in a dictionary that uses timestamps of when they were received as unique keys. This makes it easy to determine the newest of the stored messages and also allows for a fairly trivial sorting algorithm to arrange the messages appropriatly before passing them on to the class that will take care of displaying them to the user. The SZDataManager offers the following function to check for new messages:

```
- (void)checkForNewMessagesWithCompletionBlock:(void(^)(BOOL finished))completionBlock;
```

Within this function, the SZDataManager will determine the newest of the stored message keys, transform it into an NSDate object and run a query for all messages that were received after that time. If the query doesn't return any results, the local store is up to date, if there were new messages received, they will be added to the local store. Finally, the completion block that was passed into the function will be executed. Depending on which class is calling the message check function, this block could do different things. If it's called by the SZAppDelegate upon application launch, it would be informing the user about new messages, if called by the view controller that's displaying a list of all message threads (grouped by users), it would simply be updating its view accordingly.

5.7. Third-Party code

The project's folder structure and naming convention clearly point out third-party classes. All custom classes developed within the scope the Skillz project have an SZ prefix in front of their class names. The only exception to this rule are custom class categories, which generally follow the naming convention of putting the name of the existing class in front and a short keyword describing the extension of functionality in the end of the classname, separating the two by a "+" character.
At the top level of the project's class directory, a sub-folder called "ThirdParty" collects all external code used within the project:

- **BSKeyboardControls**
  An open-source library to add a toolbar with buttons to a keyboard or picker input view
  https://github.com/simonbs/BSKeyboardControls
- **MBProgressHUD**
  A screen overlay to indicate an ongoing saving or loading process and block user interaction
  https://github.com/jdg/MBProgressHUD
- **NMRangeSlider**
  Custom iOS control for a slider that lets the user select a range (min and max value)
  https://github.com/muZZkat/NMRangeSlider
- **NSDate+TimeAgo**
  A category class for NSDate to convert a date into a string indicating how much time has passed since that date ("33 seconds ago", "8 hours ago", "2 weeks ago" etc.)
  https://github.com/kevinlawler/NSDate-TimeAgo
- **Reachability**
  A helper class to conveniently check for working a network connection
  https://github.com/tonymillion/Reachability

6. Conclusion

Blablabla