CrowdMarket

A crowdsourcing approach for price awareness and supermarket expenses planning

Introduction

In many cities, customers may choose from multiple chains of supermarkets. Often a single city region has many supermarket units that are easily accessible by nearby customers, allowing them to choose and even combine products from different chains. In other cases, customers may have to perform more distant journeys to reach a supermarket unit of their choice, which imply transportation costs.

As supermarket expenses can represent a significant part of a person's or a family's budget, improvements to this aspect is likely to result in sensible gains to people's quality of life. Today, despite the advent of breakthrough of information technologies, supermarket customers in many cities still face the challenge of manually comparing prices from different grocery stores when deciding where to perform their expenses.

The emergency of Internet and the popularization of mobile computing brough innovative opportunities in terms of applications that can help the life of people, especially those living in urban areas. In addition to the exiting volunteer and crowd-based technologies and methods that make use of personal computers and groups of users to solve the most diverse kind of problems, mobile computing adds yet new possibilities as users carry their devices with them throughout their daily activities.

In this work, we propose a crowd-based application to collect and share information about the product prices of different supermarkets in a region. Not only customers become responsible for the collection of data, but they shall also benefit from

the consolidated information regarding where they should perform their expenses in order to save money and where they can find their preferred products.

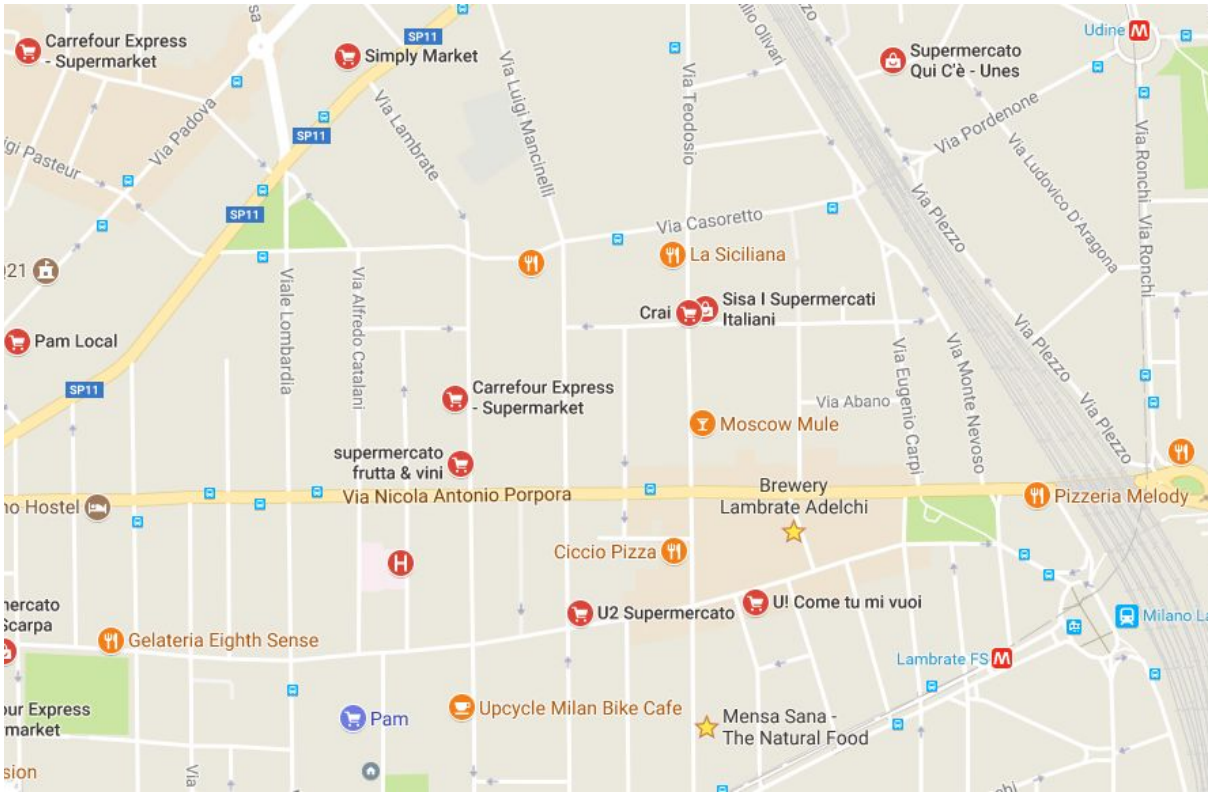

Fig. 1 - Multiple supermarket units located at the same city area of Milan

Problem Definition

The use of crowdsourcing techniques for the collection of meaningful data from users presents several challenges for its successful implementation and adoption.

The first and foremost challenge refers to *how* the target data will be collected and consumed. The Internet is the main technology enabling disruptive and novel ways of crowdsourcing information. Moreover, the consolidation of mobile computing is another important factor enabling users to participate both in the collection and consume of information.

In the context of collecting supermarket product prices, time is an important but not critical aspect, as prices are not expected to change before users can reach the comfort of their home where data could be extracted from the shop receipt. That been said, it is also important to provide users the option of collecting data while they are performing their expenses. First, because some users may feel more encouraged to do it while they pass by the products or while they pick up products. Second, because even if prices are not expected to change so quickly, the less time is elapsed between the moment it is seen and reported to the

system, the more accurate that information is likely to be. The later is particularly true for products in discount, as they tend to have their prices changed within a short period.

In order to provide a reliable source of information, data collected from crowdsourcing must be properly validated. In the context of supermarket prices, the validity of data is given by the correctness of data (i.e., the price of products) input by users. Common problems that may arise in this process are the unintentional or intentional input of product prices that do not correspond to what is announced or the input of product prices that correspond to a different product (wrong price); the input of product names or brands that is not found in a given supermarket (wrong product); or the input of product prices that have already been changed (outdated price).

As with any crowdsourcing campaign, it is of great importance to motivate users to participate. With the emergency of mobile devices embedded with sensors and artificial intelligence able to extract information without requiring the user specific actions, *opportunistic crowdsensing* aims to reduce the effort required by users in the collection of data. Nonetheless, this approach is not always possible nor well appreciated by users. Also, even opportunistic methods often require the consent from users, who may be unwilling to contribute if they are not able to identify and agree with the benefits of doing so. In order to aggregate a greater number of participating users, they must be provided with immaterial and perhaps material rewards.

The type of crowdsourcing campaign proposed in this work can be considered self-contained: in addition to the rewards that may be given to contributors, an important motivation factor for collecting product prices from supermarkets will be to have access to the catalog built by the community and to be able to optimize their supermarket expenses. Notwithstanding this, some well-known and innovative techniques may need to be employed to to avoid users willing to consume information without contributing (also known as free riders).

Another aspect that plays a role both in the participation of a crowdsourcing campaign is the quality of the tools to be employed by users. Accordingly, it is also important to provide users with well designed applications so that: i) the quality of the experience (QoE) can mitigate the participation effort; ii) the QoE can also stimulate the usage of the system by consumers of the information. In that sense, not only the visual and interactive aspects must be considered, but also the functionalities that can be offered based on the catalog of information available.

Proposed Solution

The proposed solution relies on the premise that customers should be allowed to explore the competition among different supermarket chains and units for their own benefit. For this, they must be empowered with the information about the prices of products from different supermarket companies, as well as different units from the same company. Additionally, based on this information and users' preferences and shopping list, a tool should provide suggestions for the *shop decision* problem.

Motivating the User Participation

The strategy for motivating users to engage with the application and collaborate with data collection and validation will be twofold:

1) The nature of the crowdsourcing campaign is per se an important motivational factor, as the data gathered by users should become valuable information that may help them to reduce their expenses with supermarket, an important type of expense for individuals and families. This causality relation (more participation, more information) should be stressed by the advertisement of the application.
2) The campaign should be approached with a gamification strategy that rewards users for their participation. Each user action, from the collection of data until its validation, must be rewarded. As described later in this document, we envision different user levels to be unlocked as users gain more *points*. Also, users should be rewarded with badges as they accumulate more points. This kind of gamification strategy has been proved successful with many applications relying on crowdsourcing like *Google Maps, Waze, StackOverflow,* just to name a few. At this point, we do not envision material rewards. Nonetheless, as the system evolves, some kind of partnership with supermarkets and other kinds of material rewards for user participation may be employed.

Data collection

Today, almost everyone owns and carries a smartphone. These mobile devices have rich resources (i.e., sensors, computational power, network access). These features makes smartphones a good platform candidate for applications with crowdsourcing purposes.

We propose a mobile application aiming to provide users with functionalities for registering and updating supermarket product and prices, as well as consulting prices from different products and stores in order to minimize the supermarket expenses. Regarding data collection, the proposed features are: *in-locus data collection* and *receipt-based data collection.* For a description of the features that provide information based on the collected data, please refer to the Functionalities section.

In-locus Data Collection

The *in-locus data collection* targets the situation in which users can operate the mobile application meanwhile they perform their expenses. It should happen by means of specific

application features allowing users to register missing products and add or update their prices as seen live in the supermarket.

The benefit of the in-locus approach is to allow customers to collect live data, which is more reliable in terms of accuracy. Another advantage of this approach is to let users to build a list of products they are buying with the corresponding price of each product, as well as the total price. The later can be confronted with the prices charged at the counter, as in some cases, which happen more often than one would expect, the announced product price differs from the price to be paid. This is especially problematic with products advertised with a discount, as it may diverge from the price in the internal supermarket system.

The *in-locus data collection* approach can be described by the following activities:

- The user opens the application in her accompanying mobile device
- The user navigates to the *in-locus data collection* application interface
- The user searches for a product by *barcode, name, brand, or category*
    - The user can add the product if it can not be found (see *Missing Products Registration)*
    - The user selects the product if it is found
- The user inputs the normal product price
- The user inputs the discounted product price, if any, in terms of absolute value
- The user is asked whether she wants to add another product
    - If yes, the *in-locus data collection* interface is refreshed
    - If no, the process finishes and the application navigates back to the landing page

Missing Products Registration

The success of the proposed CrowdMarket application depends first on the extent of the products catalog registered. We propose to tackle this challenge from multiple approaches. As it is not realistic to assume that all products can be parsed directly from supermarket websites and other catalogs, users are given the possibility of contributing with the registration of missing products meanwhile they perform their expenses.

The in-locus missing products registration is based on what users see, i.e., how the supermarket names that product (in case of unpacked products) or the information found in products packages.

For registering a missing product, the first data to be inputed is the barcode, which provides an identifier for a product that may be used by the system to fetch details from the product (e.g., a picture representing the product, its official name and brand, etc) from Internet services (endpoints). As already employed by other mobile applications (e.g.,

banking applications), there are many libraries that allow the scanning of a barcode from the device's camera. This automated process is important to avoid the manual input of the many digits that compose a product barcode.

Once the barcode is captured and parsed, the system should try to fetch product details from Internet services. In particular, it should try to obtain the product name and brand. Failing to fetch these details from Internet services must be handled with the activation of the corresponding form fields for mandatory user input: a *brand* must either be chosen among the existing ones found after a search by name, or a new brand must be registered by the user and associated to the product been registered. Brand registration is simpler and requires only the input of the brand name.

Continuing with the registration, a *category* must be selected from a predefined list. Finally, a specifiers must be inputted. This field may correspond to the quantity or size of the product (e.g., a pack of 500g of a given product, or a pack with 6 units of a product). Once the product has been registered, it becomes available in the system for having its price from different supermarkets updated by this and other users (see Data Model).

As for the validity of the data inserted by users, we argue that binding products to their barcode will limit the number of fake products registered as part of malicious user activities. Nonetheless, we propose a feedback mechanism to identify both invalid product information (incorrect name, brand, specifier, category, etc) and fake products. More details on the proposed validation strategies are provided later in this document.

The *missing product registration* can be summarized by the following activities:

- After failing to find a specific product from the *in-locus data collection*, the user is asked if she wants to proceed with the *missing product registration*
- The device's camera is activated and the captured scene displayed for guiding the user in targeting the product barcode
- The user is asked to fulfil the remaining data about the product (see Data Model)
- The app navigates back to the price update interface, with the new product already selected

Recept-based Data Collection

The *recept-based data collection*, in contrast, targets the participation of users after they have finished with their expenses. Once in possession of a supermarket receipt, the user can add or update the corresponding product prices from a specific interface in the application.

The benefit of the *receipt-based data collection* is to allow users to participate from a more comfortable place like their home. That been said, this approach complements the

in-locus data collection, as different users may have different preferences for using the application and collaborating.

Supermarket purchase receipts are rich sources of information. To help improving the quality of the information in the system, users must provide the corresponding receipt along with the price updates for the products from a specific interface in the mobile application. For this, users can use their device's camera to take a picture of the receipt and attach it to the input form, or choose it from the local device storage (e.g., if the receipt has been scanned and downloaded). Later on, the receipt shall be employed by other users responsible for validating the data collected from that receipt. More details on the proposed validation strategies are provided later in this document.

The *receipt-based data collection* can be described by the following activities:

- The user opens the application in her accompanying mobile device
- The user navigates to the recept-based data collection interface from a landing page
- The user searches for a product by barcode, name, brand, or category
    - The user skips the product if it can not be found
    - The user selects the product if it is found
- The user inputs the normal product price
- The user inputs the discounted product price, if any, in terms of absolute value
- The process continues with more products until all products in the receipt have been inserted
- The user attaches an image from the corresponding supermarket receipt
- The user inputs the date and time as found in the receipt
- The user inputs the total price as found in the receipt
- The process finishes and the application navigates back to the landing page

Data model

Products

Since data is a fundamental aspect of the proposed application, a high-level description of the application data model is provided in this document's section instead of the technical document containing other details of the application specification.

Regarding the data model of the main asset in the system, i.e., supermarket products, we propose a simple taxonomy in which products belong to specific categories. Categories are pre-defined by system administrators and, at this point, not subject to user modification. The following list presents an example of product categories:

- **Beverages** – coffee/tea, juice, soda
- **Bread/Bakery** – sandwich loaves, dinner rolls, tortillas, bagels
- **Canned/Jarred Goods** – vegetables, spaghetti sauce, ketchup
- **Dairy** – cheeses, eggs, milk, yogurt, butter
- **Dry/Baking Goods** – cereals, flour, sugar, pasta, mixes
- **Frozen Foods** – waffles, vegetables, individual meals, ice cream
- **Meat** – lunch meat, poultry, beef, pork
- **Produce** – fruits, vegetables
- **Cleaners** – all- purpose, laundry detergent, dishwashing liquid/detergent
- **Paper Goods** – paper towels, toilet paper, aluminum foil, sandwich bags
- **Personal Care** – shampoo, soap, hand soap, shaving cream
- **Other** – baby items, pet items, batteries, greeting cards

The categorization of products shall help users to find products when they fail to recover the product's name or brand when searching for a product and, for some particular reason, cannot search the product by its barcode (e.g., the product is not in reach).

In addition to categories, products are also bound to a given manufacturer's brand. The goal of mapping products to a brand is to allow the registration of and comparison between similar products from different manufacturers. The later is aligned to the main goal of the crowdsourcing campaign of enabling users to optimize their supermarket expenses by exploring concurrence. As predefining all existing manufacturers would be difficult, we propose that brands to be crowdsourced along their corresponding products whenever a given brand is missing and can not be fetched from Internet services by means of the product barcode.

Products should also be given a *specifier*. The purpose of this generic field is to distinguish between different versions of a product, e.g., different sizes and/or number of units. Needless to say, this is a fundamental aspect of the crowdsourcing campaign targeting product prices.

A high-level description of the products data model is given by the table below:

Field	**Description**	**Mandatory**
Barcode	- The product barcode, as captured from the product using the user device's camera	yes

Name	- As seen in the product package; or - As announced by the supermarket (unpacked products)	yes
Brand	- As seen in the product package; or - As announced by the supermarket (unpacked products like fruits and vegetables)	yes
Specifier	- The size of the product; - The quantity of the product; or - Other meaningful attribute distinguishing this product from similar ones	yes
Category	- The category that best matches the product among the predefined ones	yes

Supermarket Stores

As the main purpose of the crowdsourcing application is to enable the comparison of product prices from different supermarket stores, the later must also be represented in the system.

A high-level description of the data model for a supermarket is provided by the table below:

Field	**Description**	**Mandatory**
Name	- As seen in the top or bottom of the receipt	yes
City	- The city in which the store is located	yes
Address	- As seen in the top or bottom of the receipt, composed of street name and number	yes

Product-Store

The relation between a product and a supermarket store models the commercialization of that product by the store. The main field in this relation is the product *price* as announced by the corresponding store. Another important field is the *update date*, which keeps track

of the date and time in which the price was last updated. Its importance is twofold: i) to let users know the accuracy of the price information they get from the system for different products and stores; ii) to guarantee that the most recent price among those collected by different users for the same product/store combination shall prevail.

The table below describes the high-level data model of a product-store association:

Field	**Possible user Instruction**	**Mandatory**
Supermarket	- The supermarket store selling the product	yes
Product	- The product been sold	yes
Price	- The price of the product in that specific supermarket store	yes
Update-date	- The date and time in which the product price was last updated in the system for that specific supermarket store	yes

Receipts

The *receipt-based data collection* is addressed with a complementing data model associated to each receipt that is parsed by the users. In addition to the supermarket - chosen from a list of registered supermarkets - the model contains a field for the *shopping-date* of the receipt (when the shopping took place, used for validating price updates from products in the receipt) and also the total price (used for validating the total input price corresponding to all products in the receipt).

The table below describes the high-level data model of a receipt:

Field	**Possible user Instruction**	**Mandatory**
Supermarket	- The supermarket corresponding to the receipt	yes
Receipt image	- Readable picture(s) of the receipt	yes

	- Covers the whole receipt	
Shopping-date	- The date and time of the purchase, as seen in the top or bottom part of the receipt	yes
Products	- The collection of products in the receipt, each one modeled by a product-store association	yes
Total price	- As stated in the receipt, it must correspond to the sum of all prices of products in that receipt	yes

Data Validation

As important as the process of collecting data is the process of data validation. The application can provide correct recommendations to users if and only if the provided information by the crowd is correct (valid). Hence, it is paramount to validate the correctness of the data collected by users.

The crowdsourced data may be invalid for different reasons:

- Temporal: inputted product price is outdated and does not correspond to the actual value in the supermarket;
- Content: product price is inputted incorrectly and does not correspond to the actual value in the supermarket; and
- Association: a product is selected incorrectly and the inputted price does not correspond to that product.

To address data validation, we envision two complementing approaches. First, the system must employ automated data analysis to identify any kind of inconsistency in the data without human intervention. Among others, significant discrepancies between the previously known price of a product and the price been inserted by an user may be seen as an indication of invalid data. Moreover, the system should define a threshold for the age of data in the system: products that have been last updated too long ago (e.g., more than a year) are candidates to be removed from the corresponding supermarket. Second, the system may also benefit from the contributions of other users in the community with the purpose of validating data. As such, users can act not only as data providers, *but also as data validators*.

Next, each validity aspect is further explained along with the validation strategies adopted.

Temporal Validity

To address the issue of outdated prices, the system must take into account factors like the economic inflation in the region or country of the target supermarket, as well as the price type: products without discount are likely to follow the inflation, whereas products with a discount should be expected to suffer abrupt variations in their price within a short interval.

First and foremost, the temporal validity of collected data has a greater importance for the *recept-based data collection* approach. In this method, users can rely on receipts to update product prices. These receipts may happen to be from expenses performed several days before.

The *automated validation strategy* for this aspect is simple: for each product in the receipt, the system must check if the current product price is newer than the one from the receipt, i.e., if that product price has already been updated after the receipt date (shopping-date < update-date). If that is the case, the current price for that product is kept and the price in the receipt should be ignored. Other products whose price in the system is older or non-existent should be updated with the price in the receipt. Needless to say, if the receipt date is too old, the whole process must be aborted and the data coming from that receipt ignored.

The *manual validation strategy* for this aspect consists of distributing the validation task to other users in the system so that the inputted data, including the receipt date, the total price, and the price from different products in the receipt can be checked and validated. This validation approach is specially important to assure that the date of product prices in the receipt corresponds to what the original user reported to be, as having the wrong date inserted could drastically change the price results.

Still regarding temporal validity, prices inputted by means of the *in-locus data collection* approach must satisfy a *locality* constraint in which the mobile device geolocation must be in proximity to the coordinates of the supermarket store indicated by the user. This verification has two goals: to prevent malicious usage of the application by users outside supermarket stores and to increase the chances that prices inserted by means of the in-locus feature follow its specification, i.e., happens inside a supermarket store with the most updated prices from products.

Content Validity

The content validity encompasses both types of data collection. Also in this case, we propose an hybrid validation approach.

In the automated content validation approach, price values inserted by users must not substantially differ from the current one, unless pointed out as a discount price. The system should take the official inflation as a reference and make sure that the difference between the current price value and the value input by the user should not extrapolate *k* times the official inflation corresponding to the period between the current price has been inserted and the date of the new price (*current date* for *in-locus data collection* or *shopping date* for the *receipt-based data collection*).

The value of variable *k* is left as a parameter that must be adjusted according to factors like the variance of the price updates systemwide. For example, the system could start with a value of 2 times the official inflation, i.e., *new price* / *current price* <= *2*. As price updates are pushed to the system, the value of *k* may be adjusted to reflect the ratio in which invalid prices are inserted in the system.

Prices that suffer substantial variation - i.e., a variation higher than *k* - should be marked for subsequent manual validation. This transient condition of a product price should be solved by letting other users to validate price updates market for validation. This manual, crowd-based content validation can be achieved by asking users to check for the validity of a given product price once they are in the corresponding supermarket. If the validity of the inputted price is confirmed, it should be accepted; otherwise, it should be discarded and the current price for the product kept.

The manual content validation strategy also applies to the *receipt-based data collection*. As described for the temporal validation, it consists of distributing the validation task to other users in the system so that the inputted data, including the shopping-date and the price from different products in the receipt, can be checked and validated. In this case, the user responsible for validating the receipt should employ the receipt image attached and confront the data provided with the data in the receipt. As with the temporal validity, this validation approach is important to assure that the price from different products correspond to the one inserted by users, including the total price in the receipt.

Association Validity

The association validation should follow a similar approach from the content validation: if the price substantially differs from the current one (variation higher than *k*), it should be seen as an indication that the wrong product have been selected. As there is no reliable way of distinguishing the case in which a product price has been misspelled or maliciously

inserted with the wrong value (content) or the wrong product has been selected (association), the system should follow the same validation approach as before: the price update should be market for validation. If the price is confirmed to be invalid, the validating user will not be able to identify the intended product and that price update will have to be discarded. Otherwise, the updated product price should be accepted and made available in the system.

User Levels:

User levels are part of a gamification strategy to motivate users to participate and gain access to more advanced levels. Some specific functionalities of the application are not accessible as soon as the user installs the app. The progression from a *basic level* to an *advanced level* is given by the participation of each user in crowdsourcing and validation tasks

Basic Level

As soon as a user profile is created, the application assigns him to the *basic level*. At this level, users are able to:

- Send updates of product prices with pending validation (see Data Collection and Data Validation)
- Search products by name, brand, category, and supermarket (see Functionalities)
- Get the best known supermarket satisfying a given shopping list (see Functionalities)

Advanced Level

Advanced level users are considered more reliable in terms of the information they provide to the system, and therefore are allowed to perform certain tasks. In particular, the *advanced level* unlocks the permission to submit receipt-based price updates without the need for peer validation. This direct procedure aims to accelerate the rate in which product prices are updated and reduce the effort required by other users in crowd-based validation.

To be promoted to the *advanced level*, users need *CrowdMarket points (CMP)*, the adopted application unit for the gamefication, which includes the reward of badges and user level progression. Users can receive CMP by:

- Engaging in crowdsourcing tasks, i.e., *in-locus* or *receipt-based data collection*. Each product whose price is updated with one of these methods rewards the users with 1 CMP. The user shall only receive these points after the prices she has provided have been validated (automatically or manually).
- Validating the data collected by other users (see Manual Validation). Also in this case, each product price that is validated, i.e., indicated as valid or invalid, should reward the user with 1 CMP.

The upgrade from the basic to the advanced user level requires a minimum of 100 CMP. At this stage, this value is an estimation based on the compromise for motivating users to reach the *advanced level* (not too high in order to be feasible) and the compromise of achieving a minimum degree of trust in the user before it can have access to the *advanced level*, which allows her to update product prices without peer validation.

As the system becomes available and is employed by more users, this parameter should be adjusted after the perceived metrics such as how often invalid data is inputted and how often users are upgraded from basic to advanced level. Also important, as the system evolves, more features and levels may be added to model different degrees of trustiness and access.

Free Account vs Premium Account

Yet another distinction is made between a *free account* and a *premium account*: the later should provide access to some exclusive functionalities for those that contribute with financial support to the continuous development and maintenance of the application. In particular, *premium account* users have access to a more advanced supermarket suggestion feature (see Functionalities section).

Notwithstanding the exclusive functionalities of the *premium account*, the core functionalities in the app should be available for users with a free account so that more users can participate and increase the volume and quality of data collected.

Badges

As part of the gamefication strategy, users that accumulate CMP shall receive badges. Badges are virtual stamps to be associated with user accounts as a reward for different accomplishments:

- The user has reached a minimum number of CMP;
- The user has participated X times within a period Y of time;
- The user has updated the price of R products;
- The user has validated W products;

The price definition of which badges and rules are to be used are out of the scope of this initial definition of the system.

Functionalities

On top of the database built by users, the application will provide some valuable functionalities to help users to achieve their goal of minimizing their supermarket expenses cost. As such, the following functionalities are proposed:

- [Free] Search for products by barcode, category, brand, or by name;
- [Free] Suggest the best supermarkets based on the user's shopping list; and
- [Premium] Suggest the best combination of supermarkets based on user's shopping list, the maximum number of supermarkets, the minimum number of items per supermarket, and the maximum distance from the user's location (as fetched from the device's GPS or provided by the user)

Product Search

The product search allows users to search for registered products and obtain information about their price in terms of *value* and *update date-time.* It also enables users to know that a given product is commercialized by a specific supermarket store. In this sense, the *date-time* is an important information as the more recently a product has been updated, the more likely it will be available.

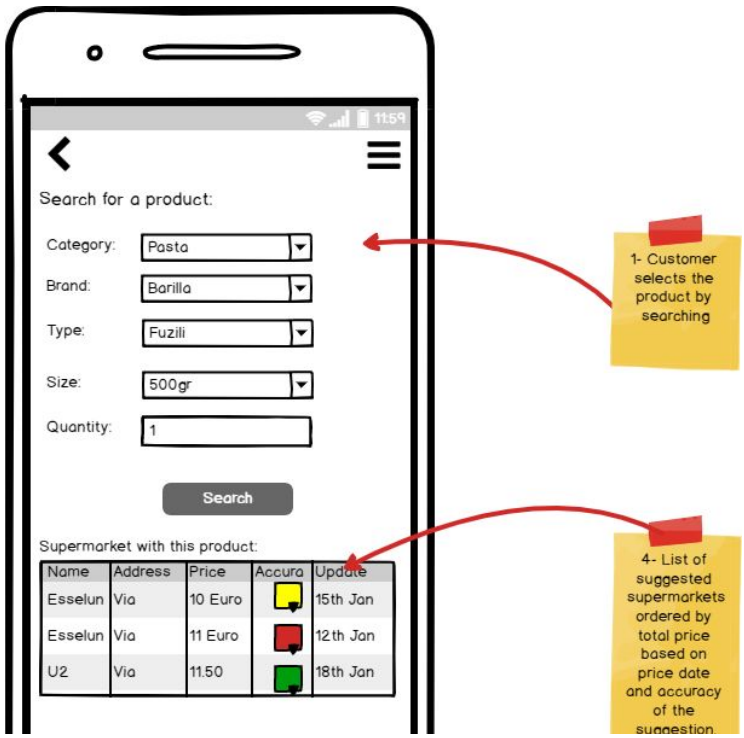

Figure 2: user searches for a specific product and receives the list of supermarkets having it with the product price, accuracy of the results in color indicators and the latest price update

Supermarket Suggestion

The supermarket suggestion is the core functionality of the application. This feature takes as input: i) a list of supermarket stores (registered in the system) the user considers as an option; and ii) a shopping list (of registered products).

As a result, the provided list of potential supermarket stores is sorted by the total price of the shopping list, from the cheapest to the more expensive one. In addition to the total price, each store is associated to a *accuracy* factor. This factor is calculated based on the average age of the shopping list prices for each store. The older the product prices, the less accurate the suggestion is considered.

Users should be aware of this factor with proper visual notations like different colors for different degrees of accuracy. Accordingly, some users may choose to go for the cheapest store suggestion and ignore a lower accuracy degree, whereas other users may

opt to choose the suggestion with the highest accuracy among the cheapest options and not necessarily the cheapest one.

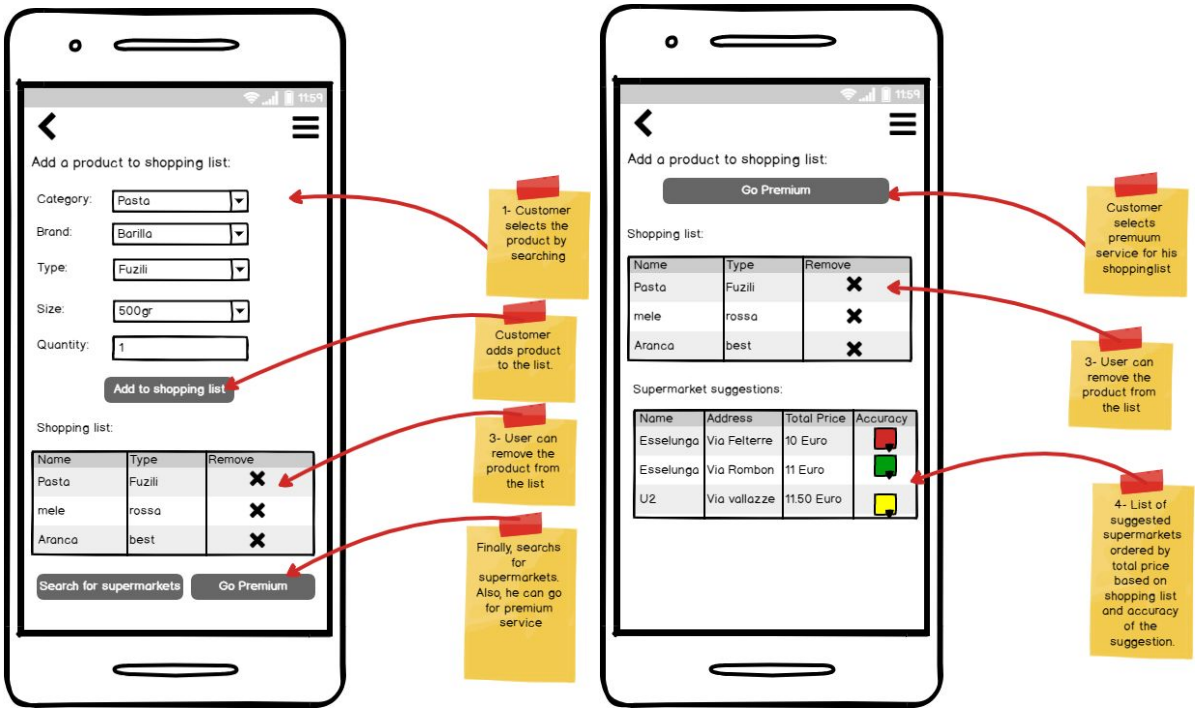

Figure 3: on the left, user selects a product and adds it to his shopping list then he searches for supermarkets. On the right, user receives the list of supermarkets with total price of the shopping list and the accuracy of the results

Premium Supermarket Suggestion

The premium supermarket suggestion complements the basic counterpart with further parameters, namely, the number of supermarket stores the shopping list can be split into, the minimum number of products in each store, and the total radius from the user location.

With the aforementioned parameters, the premium supermarket suggestion feature should also provide the corresponding sorted list of stores. The difference from the basic feature is the combination of two or more stores so that the total price of the shopping list can be further decreased by splitting the expenses and allocating each product to the store with the cheapest price.

Also in this case, the accuracy factor must be considered. To address this, the suggestion list must indicate the combined accuracy given by the weighted average between the

combined stores. In particular, the weight should correspond to the number of products in each supermarket.

Finally, the suggestion feature should consider the minimum number of products per store, i.e., it should not split the shopping list in fractions whose number of products is lower than this parameter. The later is important to avoid suggestions in which too few products are suggested to be bought in a store, as the transportation cost and time for going to different stores may not justify the expected savings.

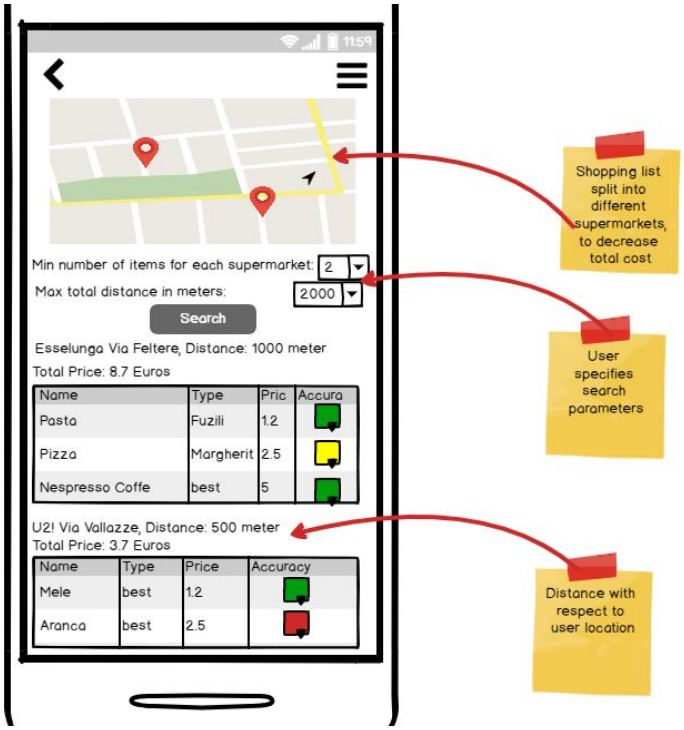

Figure 4, premium users can set shopping list split parameters to distribute their shopping into different supermarkets.

Evaluation

This section explains the evaluation part of the solution by a prototype that is provided by means of Amazon Mechanical Turk (MTurk). MTurk is a web service provided by Amazon and enables access to on-demand human workers. It helps companies to assign tasks that humans can do more efficiently than computers called Human Intelligent Tasks (HITs). Hence, human intelligence provided by workers can be embedded into their projects.

The purpose of this evaluation was to assess the feasibility of users in parsing the data in supermarket receipts, as it is a core method of data collection proposed. This section describes the experiment setup, its results, discussion, and treats to validity.

Experiment Setup

Project configuration:

MTurk allows to provide HIT description and keywords for the project, hence workers can identify what they are supposed to work on. HIT configurations is also allowed. These configurations and their descriptions are depicted in table below:

Configuration Field	Configuration Value	Description
reward per assignment	0.1 $	How much a Worker will be paid to complete an assignment
number of assignments per HIT	5	How many unique workers will work on each HIT
time of each assignment	2 hours	max time a worker has to work on a task
HITs expiration	20 days	max time that HITs are available for Workers
Auto approve and pay	7 days	max time for us to reject a submitted hit

MTurk allows to filter workers based on specified criteria. The following table expresses these dimensions:

Configuration	Configuration value	Description
master worker required	no	workers who will work on the HITs should be masters
additional qualifications	not assigned	customized filters for worker selection i.e., based on language proficiency
offensive content	no	selection of workers based on contents of the project i.e., nudity

HTML form for workers:

We designed an HTML page with relevant fields for the workers. Figure. 5 shows the specified page. On the left the images of the receipt appears to worker and on the top-right he inserts global information about the receipt. Next, he can add items to and insert the details of each item. For each field, MTurk generated a column in the result file.

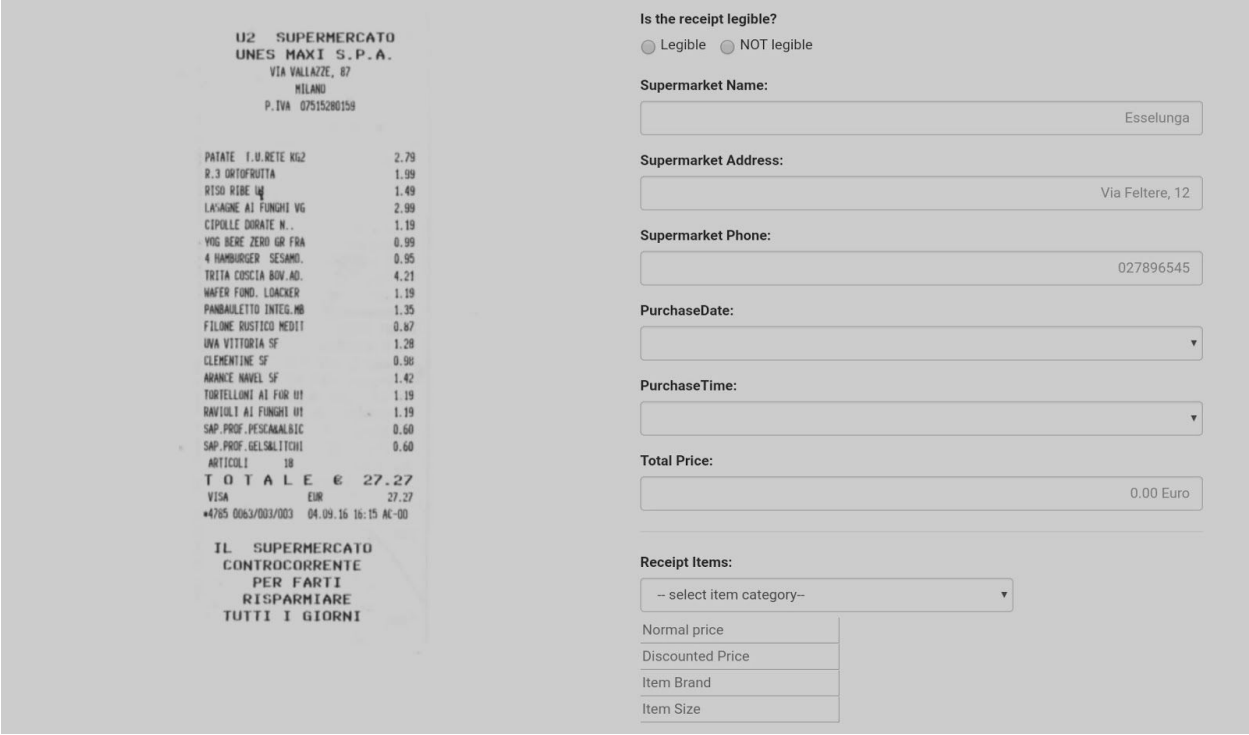

Figure. 5, MTurk HTML page for workers to transcript the receipts

CSV batch file with url to receipt images: MTurk allows to define variables in HTML forms. In this prototype we provided a variable to embed images into HTML page. The inputs are links to images which are uploaded in the cloud and a CSV file stores these urls.

Batch publishing: When everything is ready i.e., HTML form, configurations, and CSV file, we can publish the batch to workers. MTurk calculates the estimated cost based on our configurations and charges our bank account. Next, HITs would be available to workers. After a worker submits a HIT, we have the option to reject his/her submission.

Results

The results of the receipts transcriptions can be found in the link [1]. In many cases, the data expected to be parsed from the receipts by the workers was only partially correct. The main issue, as discussed in the Threads to Validity section below, consists of the language barrier,

as the works are (very much) likely to come from countries other than Italy. Therefore, considering this bias, the quality of the results must be contextualized.

[1] https://drive.google.com/file/d/1wP4DQ9n1Wtv2DQMqT7wU36J_dD-lFyk0/view?usp=drivesdk

Treats to Validity

Master workers

First we published the batch of HITs only to master workers who are identified by MTurk based on their performance over time, but after some days we notice that none of our hits assigned to any worker. There may be different reasons for this delay: first, the assigned reward to hits may be too low for them; second, as the receipts are in Italian, they may have refused to work on these HITs to avoid receiving rejections from us which would affect their reputation on MTurk framework (e.g., their master qualification).

Normal workers

Given the provided results from normal workers, we identified that many of them avoid to provide all the details which exist in the receipts. For instance, in many cases they only provided the address and did not transcript purchase items in the receipts. Hence, we have to reject their submission. However, when the number of hits increases to thousands or millions, it is not possible to validate their submissions by human. This should be reduced by providing field validators to do some minor syntax check on inserted data. However, with a standard form implementation, it is not possible to validate the semantics (meaning) of the provided information.

Long receipts:

We noticed that workers are more interested working on those HITs which are short. This is natural as MTurk rewards is the same for long and short receipts. This may affect final results of the prototype. However, in real application users should be rewarded based on both quality (validity) and quantity of items (e.g., product prices) they provide to the system.

Cultural/language bias: there are some key words in the receipts that only local people are aware of them. One instance is partita IVA (tax code) which is usually printed on top of the receipts. We notice that a high number of workers inserted partita IVA in the phone number

field. Despite this major issue in the MTurk prototype, in real applications users are expected to be local and recognize the different fields in the receipt they are supposed to parse.

Abbreviations in receipts: Many supermarkets do not print the full item details in their receipts and usually they use abbreviations instead. Hence, it would be difficult for a worker to identify the product just by looking at the receipt. This issue would be mitigated in the real application as users would have direct access to purchased products and could easily map the acronyms in the receipts with the products they have purchased.

Conclusions and Future Work

Despite the need for further validation of the proposed application, in this project we were able to reason about the challenges and potential solutions for the development and operation of a crowd-based app for supermarket expenses planning and optimization. In many cases, the problems were not visible from the first assessment of the proposed system. However, with the combination of different approaches and strategies like automated and manual data validation and *gamefication*, the application has the potential of fulfilling its goal of helping users to explore the competition between different grocery companies and reduce the amount paid for this important component of people's and families' budget.

As future work, we are interested in the implementation of the *CrowdMarket* application both to better validate the solution proposed in this project and, if it is proven valid, to enjoy the benefits offered by the application. Last but not least, this project also brings interest from the entrepreneurship point of view. For that, a business model would have to be better elaborated, as the infrastructure, development and maintenance costs for a large scale system would have to be considered.