Design and Implementation of a Mobile Peer-to-Peer Crowdsourcing Platform

Gongbin Guo

Abstract—**Crowdsourcing platforms have been really popular in recent years. People from all over the world can ask and solve questions at any time and anywhere they want. From the standpoint of the people who ask questions, they could get answers that are beyond their expectations and then think outside of the box. From the standpoint of the people who answer questions, they could gain personal accomplishment and sometimes make money from it. In this lab, two properties are added to the crowdsourcing platform. The first one is real time property. This means users can get tasks instantly after they are submitted to the server, and users can get answers instantly when answers are submitted to the server. And old tasks are not displayed on the platform, which means a filter on the time is applied. The second is the location property that is also termed as geographical information. So every task is associated with geographical information so that users can find tasks that are near to them. People know things better that are around them. This would improve the quality of the answers. And users who can get accurate answers are willing to use this crowdsourcing platform again and pay more. So this is beneficial for both the worker and the requester.**

I. Introduction

Crowdsourcing platforms help individuals or organizations to obtain needed services or ideas. Although crowdsourcing was coined as a portmanteau of crowd and outsourcing, actually it is not outsourcing. Outsourcing means the hired people provide the specified work within a specific time span. It emphasizes on the speciality of the people. Crowdsourcing platforms distribute work among people from the Internet. It takes advantage of diversification of the human resource. The people from the Internet are undefined, the background, the experience, the culture and so on. They could be professionals or amateurs. No constraint is set. This is to lure people to cooperate and innovate.

In this lab, two properties are added to the crowdsourcing platform. The first one is real time property. This means users can get tasks instantly after they are submitted to the server, and users can get answers instantly when answers are submitted to the server. By doing this, people get instant feedback from the people on the Internet. This is for the people who are in agent need for help. On the other hand, real time property also requires that old tasks be not displayed on the platform, which means a filter on the time is applied. For example, tasks that are two or three days old usually are not displayed on the platform. So the basic idea is that users need instant help from the Internet, not the help after two or three days later. So basically tasks that are beyond the requirement of the time limit are not displayed even though they have not been answered yet, however people could set this filter in the Settings tab of the App. The second is the location property that is also termed as geographical information. So every task is associated with geographical information. People look around the tasks that are near to them and answer the tasks that are interested to them. By adding this property, users, to a large extent, will get accurate answers. Because people generally know things better that are around them. This would improve the quality of the answers. And users who can get accurate answers are willing to use this crowdsourcing platform again and pay more. So this is beneficial for both the worker and the requester.

For me, I always wanted to build an App out of curiosity. I would like to see what an App could be capable of and what features might be added to the existing framework. And there are also amazing open-source third-party frameworks achieving stunning effects and functionalities. People creating these frameworks must have excellent imagination and talent. It is always good to learn from these people and keep up with the Internet.

II. Related Work

This lab requires me to build an app that features question-and-answer functionality. There are two apps that I use for reference. The first one is Quora which is a question-and-answer site where questions are asked, answered, edited and organized by its community of users. The other is Zhihu that is a Chinese version of Quora. When designing this app, I borrowed the style of the two apps.
There is also a paper I used for reference which is “Location-based Crowdsourcing: Extending Crowdsourcing to the Real World”. It introduces system architecture of a location-based question-and-answer app. There are three components in this architecture. A web interface is used for creating tasks and a server including a database and client application. It also gives an overview of all the tables that are going to be created in the database. So I refined the architecture and get the following requirements. First, this mobile app can be used both by seekers and workers. Second, a graphical map is provided to the users to specify their location information. This allows for creating tasks not only based on the current location but also based on other locations. For example, users could use the map to find their desired location and ask a question in terms of that specific location.

Third, location information should be converted to human-readable text. A pair of latitude and longitude is only for machines and not for users. Human-readable location address provides better usability.

III. APPROACH

Figure 1 shows the entire architecture of this app. And the main design pattern used in this app is Model-View-Controller. The entry of this app is the log-in view controller. The log in scene is managed by Login View Controller. Users need to input their email account and password in order to log in. If users don't have their account, Sign Up Scene is used for them to create their account and it's managed by Sign Up View Controller. When users sign in, the main screen of the app will show up. There are four tabs on the main screen. They are Tasks, Maps, Search and Settings.

Figure 2 shows what the Tasks tab looks like. In the Tasks tab, Tasks Scene is managed by Tasks View Controller. When the Tasks Scene is first showed up, the app will connect to the server and fetch the tasks created by users and then it will load up the table view with the fetched results. For each cell of the table view in the Tasks Scene, there are user name by who this task is created, image profile of the user, time that the task is created, summary of the task and location that is associated with the task. Users can tap on one of the tasks to see the detail of it and then decide whether to answer it or not. The Task Detail Scene shows the detail of a task and it's managed by Task Detail View Controller. All the details will again show up in this scene plus the answers if any. There are user name and time that the answer is submitted and the answer itself for each answer. If a user decides to answer that specific task, the Answer Scene will show up and it's managed by Answer View Controller. There is a text view in Answer Scene. Users can type their answer there and click the save button to submit it to the server if they want to answer this task. After submitting their answer to the server, the app will go back to the previous scene which is Task Detail Scene so that this user can instantly see the answer on the app. And other users can also see the updated answer on their phone. If users decide to not answer this task and want to cancel it, they can click the cancel button on the left-top of the scene. The app will go back to the previous scene. There is another functionality in the Task Detail Scene. Users can create tasks by clicking on the right-top button. This will load up the Compose Scene. And it's managed by Compose Task View Controller. Here users can create a task that they want to ask. There is a button that reminds users to add location information to the task. Task Locations Scene will load up if users click on the button. It's managed by the Task Locations View Controller. It shows the user's favorite locations that are stored previously when they created other tasks. Clicking on one of the locations will select it and navigate to the previous scene. The location information will be added and displayed in the Compose Task Scene. Again there are two buttons in this scene, cancel and save button.

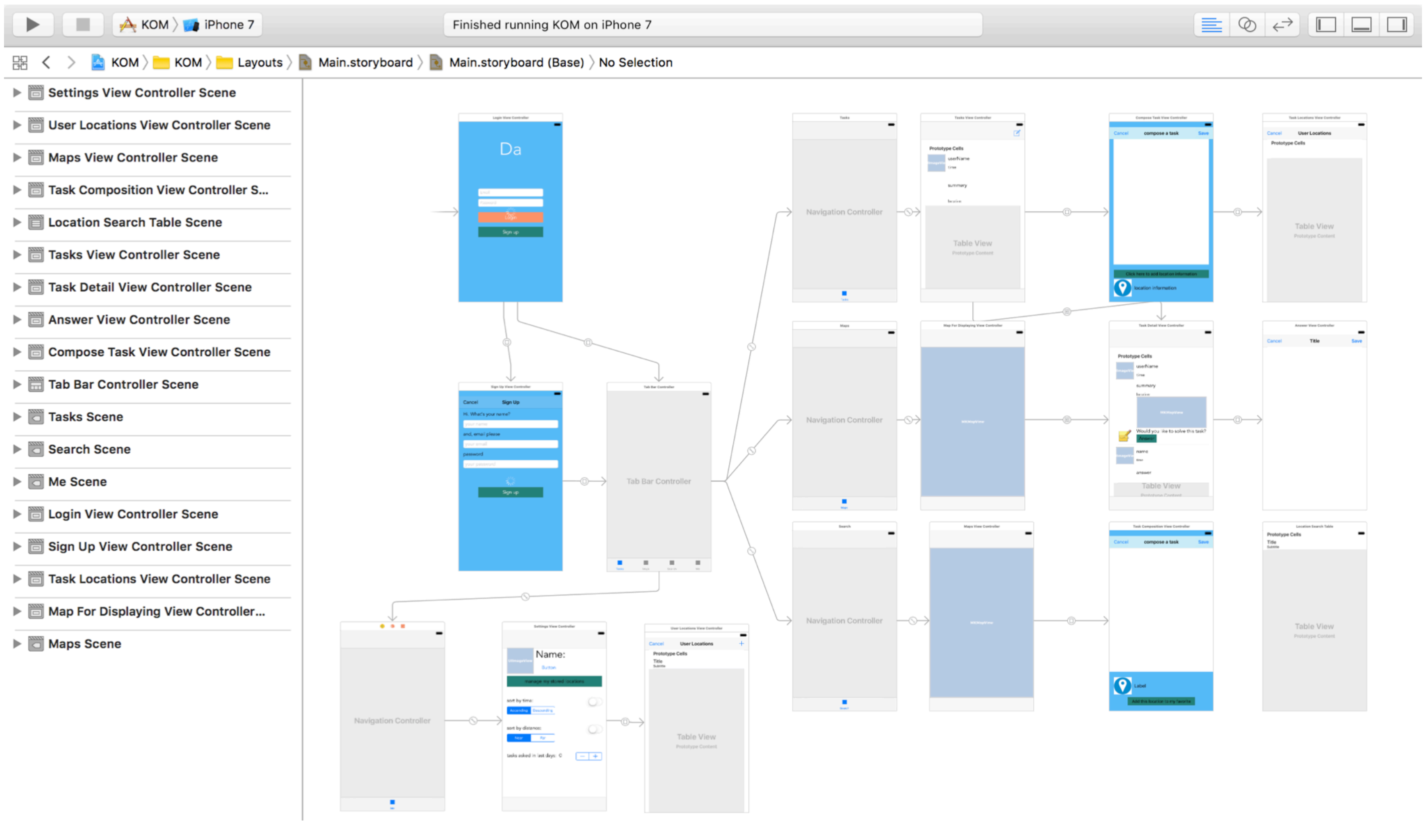
Figure 3 shows what the Maps tab look like. In the Maps tab, Map For Displaying Scene is responsible for displaying a graphical map and showing tasks. It is managed by Map For Displaying View Controller. Tasks are represented as pins on the map. Users can click on the pin then a small label pops up with location information displaying on it, as well as name of the user who created it. There is also a callout accessory button on the pin. The detail of the specific task associated with the pin shows on the Task Detail Scene if users click on that pin. It is the same scene as described previously. Users can perform the same actions within that scene as described previously.

Figure 4 shows what the Search tab look like. In the Search tab, Maps Scene is used for displaying a graphical map and searching for a location that matches user's need. It is managed by Maps View Controller. There is a search area on top of the scene. Users can use the searching function to find an address that best suits their task that they are going to create. When searching, a list of possible matches shows in another scene. It is displayed by Location Search Scene which is managed Location Search View Controller. The two view controllers work together to achieve the searching functionality. Here delegation design pattern is used in order to make them work seamlessly. When clicking on one of the available results, a pin will be dropped on the map. Clicking on the pin will show name of the location associated with the pin as well as the address. There is also a callout accessory button on the pin that will lead to creating a task with the location preset. The scene for creating a task is called Task Composition Scene which is managed by Task Composition View Controller. There is a button in this scene, which is used to add the location to the user's favorite location list. So next time, this user can create a task by just choosing one of his or her favorite locations, without searching for it.

Figure 5 shows what the Settings tab look like. In the Settings tab, Settings Scene is used for managing the app settings. Users can turn on or off the sorting functionality. Users can sort the tasks by time that the tasks are created and by distance. The distance is calculated from the user's current location to the task's location. Users can also specify the filter that filter out tasks that are pretty old. Be default, when users first load up the Tasks Scene, no filter is applied and no sorting is applied.

So in a nutshell, the Tasks tab is used for displaying tasks in a table view. Users can view the tasks and choose the one they like to answer. The Maps tab is used for displaying tasks in a graphical map. Tasks are represented as pins that users can click them to perform actions with the task associated with the pin. The Search tab is used for searching for a location and then creating a task with the location. The Settings tab is used for setting user's preferences.

So there are basically two ways for users to create tasks. One is using the top-right button on the Tasks tab. When creating tasks on this tab, users have to use their location that are previously stored, otherwise users can't create tasks in this way. The other way to create tasks is on the Search tab. Users need to search for a place first in order to create their task. The reason why I am doing this is to make creating tasks simple. Most of the time, Users first search for a place and then create

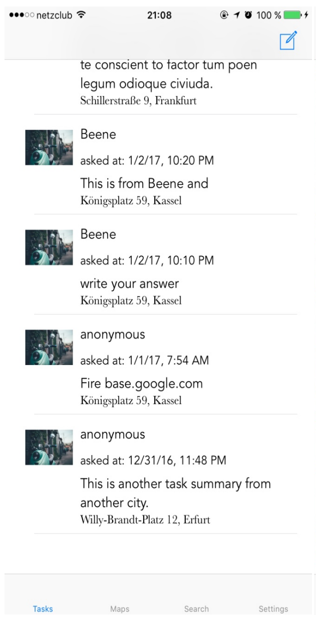

KOM 〉 iPhone 7
Finished running KOM on iPhone 7
KOM 〉 KOM 〉 Layouts 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 No Selection

- Settings View Controller Scene
- User Locations View Controller Scene
- Maps View Controller Scene
- Task Composition View Controller S...
- Location Search Table Scene
- Tasks View Controller Scene
- Task Detail View Controller Scene
- Answer View Controller Scene
- Compose Task View Controller Scene
- Tab Bar Controller Scene
- Tasks Scene
- Search Scene
- Me Scene
- Login View Controller Scene
- Sign Up View Controller Scene
- Task Locations View Controller Scene
- Map For Displaying View Controller...
- Maps Scene

Figure. 1. The architecture workflow of the project

a task. If users don’t have favorite location addresses, users have to type text and select the matched address. This process would be annoying if they want to create tasks with their favorite location address each time. Providing an alternative in this case is good for users to improve the efficiency. And there are also two ways for users to view tasks. One is using the Tasks tab. The table view in the Tasks tab is used to display all the tasks. Users can select one of them to view the detail of it and then decide whether to answer it or not. The other way is using the Maps tab. Tasks are represented as pins showed on the map so that users know where exactly the tasks is located. And what’s more, the graphical interface is also more friendly to users

In terms of the database, the reason why I didn’t choose to use traditional database, like Oracle, mysql and so on is that I like new technologies. So the database I use for this app is Google Firebase. Firebase is a mobile and web application platform with tools and infrastructure designed to help developers build high-quality apps. Firebase is made up of complementary features that developers can mix-and-match to fit their needs. I spent some time digging into it and found that it can be used to support this app. It supports database, authentication, storage, notification and more. There are a lot of APIs on firebase.google.com that I can refer to when I develop this app.


●●●○○ netzclub 21:08 100 %

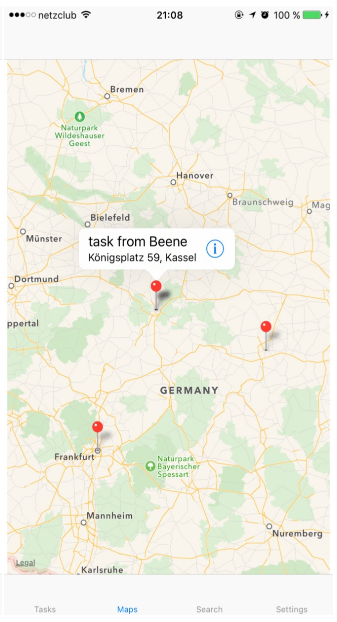
te conscient to factor tum poen legum odioque civiuda.
Schillerstraße 9, Frankfurt

Beene
asked at: 1/2/17, 10:20 PM
This is from Beene and
Königsplatz 59, Kassel

Beene
asked at: 1/2/17, 10:10 PM
write your answer
Königsplatz 59, Kassel

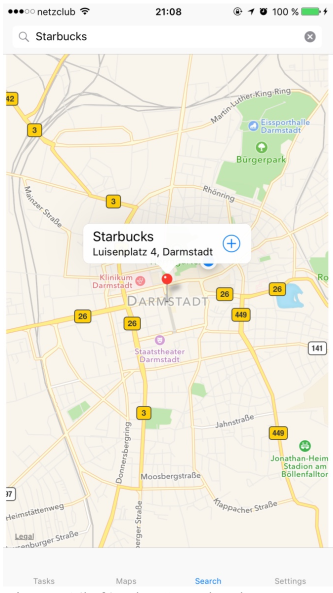
anonymous
asked at: 1/1/17, 7:54 AM
Fire base.google.com
Königsplatz 59, Kassel

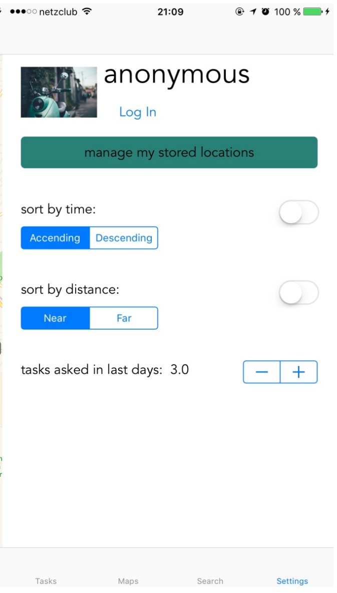
anonymous
asked at: 12/31/16, 11:48 PM
This is another task summary from another city.
Willy-Brandt-Platz 12, Erfurt


●●●○○ netzclub 21:08 100 %

task from Beene
Königsplatz 59, Kassel

.
Figure 2(left): the Tasks tab
Figure 3(right): the Maps tab





tasks asked in last days: 3.0

Figure 4(left): the Search tab
Figure 5(right): the Maps tab

IV. Implementation

There are three main models in this app, which are Task, Person and Answer. Each of the property is explained in the table.

var name: String	an user's display name
varlocations:[CLLocation Coordinate2D]?	User's favorite locations stored online
var image: UIImage?	An image for displaying an user's image profile
var uid: String?	A handle for getting the location of this user record that is stored on the firebase
var email: String?	The email for the use to sign in

Table 1: Person table

var person: Person	the person who submits the task
var summary: String	The summary of the task
var time: String	Time when this task is submitted
var location: CLLocationCoordinate2D	The location information associated with the task
var answers: [Answer]?	The answers for this tasks if any
var coordinate: CLLocationCoordinate2D	This task object conforms to MKAnnotation, coordinate is the required property
var taskReference: FIRDatabaseReference?	A handle that specifies the task on line
var taskKey: String?	//the string that specifies the task online
var title: String?	The optional property of MKAnnotation protocol
var subtitle: String?	The optional property of MKAnnotation protocol

Table 2: Task table

var person: Person	the person who answers a task
var answer: String	the answer of a task
var time: String	the time a person answers a task

Table 3: Answer table

This app needs to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices. The authentication type that is used for this app is email and password based authentication. The Firebase Authentication SDK provides methods to create and manage users that use their email addresses and passwords to sign in. The method is createUserWithEmail:email:password:completion:. The completion handler is used to handle the result whether creating a user is successful or not. If it is successful, signInWithEmail:email:password:completion: method is called to sign in the user. There is also a completion handler to deal with the result of this signing in process. These two methods are mainly associated with the LogIn scene and SignUp Scene. In both cases, there is one detail that needs to be handled. The interaction with the view should be disabled when a user clicks on the Login or Sign Up button. The code view.isUserInteractionEnabled = false is used to handle this.

When the app loads up, it connects to the firebase database to fetch the tasks. To read or write data from the database, an instance of FIRDatabaseReference: is needed. let ref = FIRDatabase.database().reference() is used to get the root of the database. And then use the child(_ pathString: String) method to get the child reference. To read data at a path and listen for changes, use the observeEventType:withBlock orobserveSingleEventOfType:withBlock methods of FIRDatabaseReference to observe FIRDataEventTypeValue events. The FIRDataEventTypeValue event is used to read the data at a given path, as it exists at the time of the event. This method is triggered once when the listener is attached and again every time the data, including any children, changes. So this method is perfect for loading the tasks and then listening for any changes when the app first launches. And this can be done in viewDidLoad method which is called only once in the lifecycle of a view. In terms of sorting and filtering the tasks, viewWillAppear is more appropriate which is called every time a view appears on the screen. Filtering should always be executed before sorting because of efficiency. And a user has to pull down the table view to trigger the filter action. Because each time a filter is applied, the app needs to connect to the firebase to fetch the tasks and then check if each task should be filtered out. When filtering tasks, each time associated with its task is calculated with current time to get the difference. There are two methods in regard to sorting the tasks, sort by time and sort by distance. But only one of them can be applied at a time. They are conflicted to a great extent. When sorting by distance, the location for the app has to be turned on, otherwise the app couldn't get the user's current location. The distance is calculated from the location of the task and the user's current location. And then comparing all of the distances would get the sorting result. Regarding the table view cell, the height of it should be adjusted to the length of the summary. And there is a delegate method in UITableViewDelegate called tableView(_ tableView: UITableView, estimatedHeightForRowAt indexPath: IndexPath) -> CGFloat. I use this method to calculate the dynamic height for each cell based on the length of each task.

Data types that can be set on firebase are: NSString, NSNumber, NSDictionary and NSArray. So when writing data to the firebase, all of the data have to be converted. Because firebase doesn't know the local type of data used by Swift. I made all of the converted type to be Swift type of Any, to be more specific, for example, the method below shows how I convert Task type to Any which actually is NSDictionary type that can be used to store data on firebase. all other types that need to be converted use a similar conversion method.

```
func toAnyObject() -> Any {
        return [
           "personName": person.name,
           "summary": summary,
           "time": time,
           "latitude": location.latitude,
           "longitude": location.longitude
        ]
    }
```

As required, each task must have a location with it. A location is actually a pair of latitude and longitude. But this isn't readable for human beings. So I convert the location to human-readable text, for example, Luisenplatz 4, Darmstadt. There is a class called CLGeocoder that can be used to deal with it. The method func reverseGeocodeLocation(_ location: CLLocation, completionHandler: @escaping CLGeocodeCompletionHandler) Submits a reverse-geocoding request for the specified location.

In the Search tab, there are two different scenarios. If the user's location for this app is turned on, the map will use the user's current location as the center of the map and zoom in with the amount of north-to-south distance of 4000 meters to use for the span. Otherwise, the map just shows the default country where the app is used, for example, Germany. When searching for a location, the class MKLocalSearchRequest is used for processing the user's input. The map region of this class is the same as the currently displayed region of the map view. The returned result of the request is used to fill up the location search scene that is showed on top the map. Clicking on one the results shows a pin on the map. The location search table view couldn't handle this job, it's the map's responsibility to do this. So a delegation is used here. The map view controller needs to extend the protocol below to handle the pin. An instance of MKplacemark is passed in and it's one of the possible matching results displayed in the location search table view.

```
protocol HandleMapSearch {
    func dropPinZoomIn(placemark: MKPlacemark)
}
```

As of the pins showed on maps, they are actually reused for the sake of scarce memory. A delegate method mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? has to be used to deal with problem. And actually the table view cell of a table view is also reused. The corresponding method is tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell.

When writing an answer or writing the summary for a task, there are some method that I use to deal with the placeholder and automatic showing up and hiding of keyboard. This is for convenience's sake. These method are all declared in UITextViewDelegate. And the methods are textViewShouldBeginEditing(_ textView: UITextView) -> Bool and textViewDidChange(_ textView: UITextView).

V. EVALUATION

A. *Relevance:*

The app basically fulfills the requirements of the lab. It supports real-time update and location-based tasks.

B. *Error:*

I have tested the app thoroughly and dealt a lot of bugs, right now it works and shows prompt messages when something happens.

C. *Customization:*

Users can customize the settings in the Settings tab.

D. *Performance:*

The database I use for this app is firebase, so the connectivity depends on the user's network. The login procedure is proximately around 1s and loading data takes a little longer, it's around 1.5s. Both of them are within acceptable tolerance.

E. *Security:*

Currently no security issue is considered. And also it's beyond my competence.

F. *Efficiency:*

There are two ways to create tasks and there are also two ways to answer tasks. So here redundancy is used to improve efficiency.

G. *Scalability:*

Scalability issue is not considered either since it's beyond the scope of this project. However there are scalable methods that can be used to deal with the data that loads up the table view if the amount of the data is pretty high.

H. *Usability:*

There are no complicated prompt messages. And the steps for both creating and answers tasks are pretty low. Users know what to do without extra hint. They can use this app without any obstacles.

VI. CONCLUSION

The aim of this lab is to develop a mobile Peer-to-Peer crowdsourcing platform. Particularly, two fundamental properties need to be achieved, which are real-time and geographical information. Through looking into some of related work and delving into development, now the app is capable of achieving aforementioned requirements. However, this app is not a one-size-fits-all application. This app is intended for tasks that are in urgent need and are associated with geographical information. And also, there is always room for improvement. People who are interested in this kind of app can extend this app. For example, in the Settings tab, showing a list of tasks that a user has asked could be regarded as a feature. Another feature, like this app could let anonymous users ask questions and answer questions.

REFERENCES

[1] www.quora.com
[2] www.zhihu.com
[3] firebase.google.com