TigerAware Android: An Improved Mobile Survey and Notification System

A Project

Presented to

The Faculty of the Graduate School

At the University of Missouri

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

Implemented and Defended by:

Weiliang Xia

Prof. Yi Shang, Advisor

May 2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Firstly, I would thank my parents for the support of my study, and the extend my sincere gratitude to my supervisor Yi Shang, for his kind guidance, not only to this project, but also though my whole oversea study, and my life. Two years goes really fast, I want to say, as the first time I talked to professor Shang, “talking with you for one moment is much better than reading books for ten years.”

Second, I want to say thank you to my girlfriend, Jiachen Ji, she must have no awareness about my graduation paper. So I want to say here, “I will return. I will find you. Love you. Marry you. And live without shame.”

I want to thank Nickolas Wergeles, Dr. Tim Trull, Luke Guerdan, Connor Rowland, William Morrison, Handrianto Yohanes Patrik and everyone I missed in the lab. I cannot forget the happiness when I show Luke around the China, I cannot forget the summer I worked together with Patrik, Peng, Yang, Junlin and Zhaoyu and our students in the summer camp. I cannot forget the patient and detailed explanation from Will and Connor and their powerful help, and thank Nick for leading me to the “Hello world” in JAVA and thank Tim, Denis, and Hannah for the great opportunity and happiness for collaboration.

I want to thank Siyang Liu, in the first semester for this project, it is him who guided me to android development and helped me a lot. I cannot forget the happiness when we solved the first bug together right before the meeting and it was a null pointer error.

I want to thank Joshua, Lewis, in the second semester for this project, we worked every Saturday on the first language learning application and went to kinds of restaurants for celebration, and meet

professor Roxana for more detail about that application and it did help us build a great foundation about the basic framework for TigerAware.

I want to thank Mohan Li, Shiqi Wang, and Runnan Dong in the third semester for this project, we stay together at Friday and I work with Mohan on Saturday for so much fun and joy.

Finally, I want to thank everyone in the lab for everything you bring to me.

ABSTRACT

From healthcare to community assessment, mobile phones have become an important tool in many research areas by collecting a variety of data, such as daily steps, walking distance and GPS data. TigerAware is a new cross platform system developed for conducting mobile surveys and collecting sensor data via smartphones. TigerAware iOS implementation is based on ResearchKit from Apple, a software framework for medical research study apps, whereas its Android implementation is based on ResearchStack, a ResearchKit-like SDK and UX framework for apps on Android. ResearchStack is designed to help developers and researchers with existing apps on iOS more easily adapt those apps for Android.

The goal of this project is to adapt TigerAware iOS apps to Android with the same functionality, and pilot some new features on Android. Based on ResearchStack BackBone and Skin, several custom question types, such as an image question type, have been implemented to support a variety of question types and active tasks for various research studies. A conditional random selection algorithm has been developed to handle survey notifications with more flexibility, such as scheduling random notification with breaks. A new algorithm has been developed to handle suspension windows in different situations, such as suspending over midnight. Finally, a new date detection algorithm has been developed to handle the start time and end time correctly.

1. INTRODUCTION

First, let us briefly introduce TigerAware before we move on to TigerAware Android. In this big data era, scientists are trying to gather as many data as they can, especially insightful new data to draw more clear conclusions. The old way is, handing out paper surveys, asking participants to answer questions in a certain time and certain place. However, this way is hard for following up surveys, while requiring a lot of budget, though it is good for a one time questionnaire. Thus, over the past years, researchers start to rely more and more on smartphones, websites and these kind of new way to gather novel data. Smartphones are cheap and convenient source for people to communicate, for information exchanging and gathering. Moreover, this kind of modern device usually equipped with a wide range of sensor to get the information outsides the world, such as motor activities, fitness, cognition, speech, hearing, hand dexterity, and vision. Researchers can use Accelerometer

Gyroscope sensor to detect the motion of participant while they are finishing a specific task, they can also use GPS data to judge the user behavior, filter out the unreliable information and so on. In a word, researchers can use smartphone surveys to approach different kinds of specific research questions.

Thus, survey based on smartphone applications become more and more popular, From health to community assessment, from science to engineering, they have become a cornerstone in many research areas.

1.1 Problem Description

Smartphone applications can collect a variety of new data, such as daily steps, walking distance and GPS data compared with old paper surveys. However, smartphone-based studies are difficult to develop and deploy, because they usually require on-sites teams with the researchers to meet the

specific study purpose, which may cause a major portion of research budgets. Current platforms for data collection, are either limiting researchers by just providing a fixed platform and lack the flexibility of develop, modify and integrate with the new data collected or requiring on-sites teams with the researchers [2][3]. For example, a health related research team may need a survey about blood alcohol level, using external sensor such as Bluetooth breathalyzer. Upon data arrival, researchers may also need to visualize and interact with data in different ways. The software development to accommodate these needs is time consuming and expensive, taking a large portion of research budget. To solve this problem, an innovative system called TigerAware is developed. TigerAware is a cross platform system which allows researchers to create a variety of surveys, question types by themselves, collecting a wide range of data including but not limited to question responses, on device sensor data, such as GPS data, and external sensor data, such as blood alcohol level from a Bluetooth breathalyzer.

However, the main SDK ResearchKit used in TigerAware can only support IOS System, thus an Android based on application is needed.

Even using a similar SDK for Android, there are still few challenges towards achieving the same functionality as IOS system or reaching some specific research study goals. The following is the list about the problems we have to solve comparing to the current most similar survey framework, Research Stack based on three aspects: question types, notifications and suspension.

(1) Question types:

Static vs dynamic: as for basic questions, there are two main problems, one problem is, most question types supported in Research Stack are static, but in TigerAware Android, we need more dynamic questions, which means researchers

Can change the question settings to achieve their study use. For example, a multiple choice question in Research Stack is a question user can select one or more than one choices, but in TigerAware Android, this question may be a strict single choice question or a more flexible multiple choice question based on how researcher set this question for their study use.

Basic questions vs custom questions:

There are only few limited question types in ResearchStack which are not enough for some study use. Then some new question types are needed. However, there are few examples or explanations about how to create a new question type based on Research Stack framework.

(2) Notifications:

The origin Research Stack Backbone frame lack the ability to receive and create notifications on the device. However, in order to tell participants to take surveys on time, each survey needs a few notifications scheduled before it. They can be random notifications and schedule notifications. To achieve the goal of research study, random notification need to be scheduled during a period of time and scheduled notification need to be scheduled at a certain time. However, both of them are not support in Research Stack Backbone.

Moreover, even for the current TigerAware android system, which has the ability of schedule notifications, for the purpose of some specific study use, more functions need to be added. Random

notifications need to be scheduled with fifteen minutes break, scheduled notifications need to be scheduled based on number of days instead of schedule based on the end day.

(3) Suspension:

According to notification system, an advanced suspension system is needed to block notifications for participants when they do not want to be bothered. During the suspend window, there should not be any notifications showing to the participants even random notifications and scheduled notifications are scheduled at that time.

Moreover, even for the current TigerAware android system, which has the ability of suspend notifications based on two timestamps during one day, it still lack the ability of scheduling one time suspension instead of scheduling it over and over again.

Table 1 The list of functions planned to be implemented in TigerAware Android

Functions	Requirements
Questions	Dynamic and custom questions
Notifications	Schedule notifications with 15 minutes break
Suspension	Schedule one time suspension

1.2 Proposed Solution

Like we just mentioned above, the main SDK ResearchKit used in TigerAware can only support IOS System, thus an Android based on application is needed. The overall goal of this Android application

is to make it easier to adapt the existing functions on iOS into android with the same functionality, and develop the new features together with TigerAware IOS to help some study specifics. In more detail, TigerAware Android is using a similar framework called ResearchStack (while TigerAware IOS is using ResearchKit) to accommodates these needs we mentioned above. To reach the same functionality as TigerAware IOS, we implement ResearchStack Backbone, and to achieve the same structure as TigerAware IOS, we implement ResearchStack Skin in android application. As for some functions which are not supported by ResearchStack but ResearchKit, which is still common, because these two SDKs' functions are not one to one, we implement it into TigerAware Android by defining and adding extra modules into ResearchStack. In Chapter three and four, we will talk about how we implement these functions in more detail.

TigerAware Android can also be easily incorporated into the existing TigerAware System with minor design and changes by using the similar SDK we mentioned above. Furthermore, the platform is highly configurable, allow researchers to design, develop and deploy their own question types using the existing modules.

By using the similar SDK Research Stack, most questions can be implemented directly with minor changes and the following is the table towards the solutions in three aspects: question types, notifications and suspensions.

(1) question types:

a) The first problem is TigerAware needs dynamic survey design and handling system, to solve this problem, TigerAware Android uses two ways: one way is to rewrite the question framework to make the question itself dynamic by adding more parameters. These

parameters all come from TigerAware Dashboard, after research create their own surveys, these parameters will go along with questions into Firebase then be received by TigerAware Android. For example, a text slide question which can only show text, after we changed the framework, an option called “show image” can be added into this question. If a text slide question contains “show image” option, it may show images going along with this survey. In this way, we make the question itself dynamic.

Another way is to keep the question static but create a new dynamic question which can select static questions dynamically. For example, in ResearchStack, there are two multiple choice questions. One is single answer question, another is single or multiple answer question. Then in order to dynamically make multiple questions based on study use. TigerAware Android create a new multiple question type which contain a parameter called “allow multiple”. If allow multiple is set to be true, TigerAware Android will choose single or multiple answer question, otherwise it will switch to single answer question.

b) The second problem is some questions researcher need for their study use are not supported by ResearchStack. To solve this problem, TigerAware Android takes two actions based on different situations. If a new question type is similar to an existing question type, then modifying that existing question type is better, if a new question type is totally different with those existing question types supported by Research Stack, then creating a new custom question type is better although it will cost more time and work than just modifying.

(2) notifications:

To ensure at least fifteen mins break between notifications, there are two algorithms planned to be implemented for this situation: the latest start time algorithm and the conditional random selection

algorithm. Both of them can solve the problem of scheduling random notifications with at least fifteen minutes break problem and we will talk about them later in the chapter three, System design and implementation.

(3) suspension:

To achieve the ability of scheduling one time suspension instead of scheduling it over and over again. We need to import a smart algorithm to calculate the start date and end date based on the start time and end time and compare the suspend date and time together with current date and time instead of only compare time without date to tell if the time is in suspend window or not. There are two algorithms planned to solve this problem by detecting date and blinding date and time while comparing. One is adding one date towards suspend window, another is an Intelligent date detection algorithm for both start time and end time. We will talk about them in more detail in suspension section.

Table 2 The list of functions and algorithms planned in TigerAware Android

Functions	Requirements	algorithms
Questions	Dynamic and custom questions	No algorithms needed, framework change only
Notifications	Schedule notifications with 15 minutes break	The latest start time algorithm The conditional random selection algorithm
Suspension	Schedule one time suspension	A Intelligent date detection algorithm for both start time and end time Adding suspend window date based on the end date

2. RELATED WORKS

TigeraAware Android is designed to work together with the TigerAware Dashboard and TigerAware IOS. TigerAware Dashboard is connected with Firebase and TigerAware IOS is using ResearchKit. Moreover, for TigerAware system, all data are saved in Firebase, Firebase is the center database in TigerAware.

In this chapter, we are going to introduce these three main frameworks we used in TigerAware Android: Firebase, ResearchStack and ResearchKit.

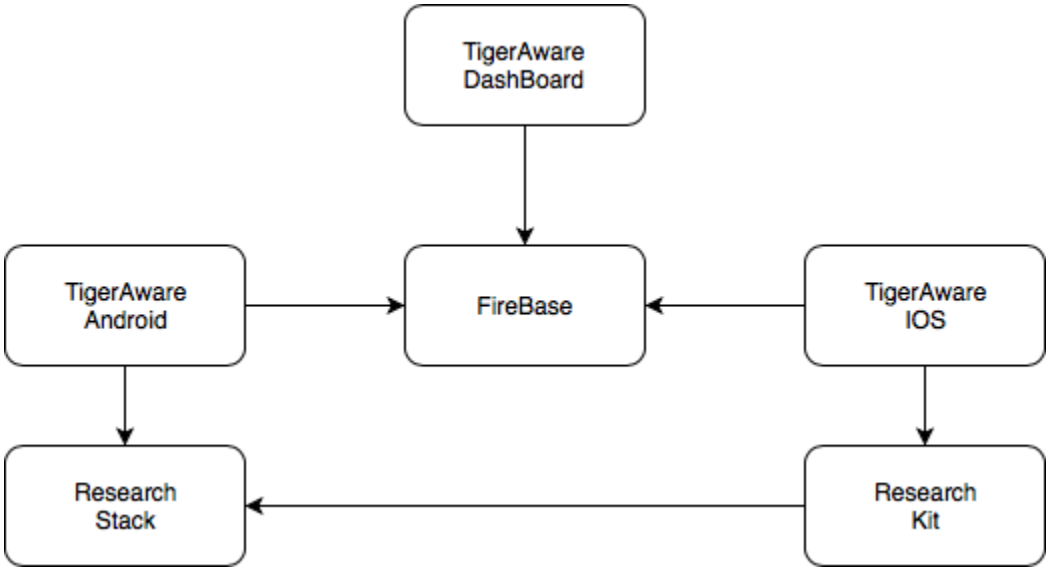


Figure 1 Main frameworks about TigerAware Android

2.1 Firebase and storage structure

Firebase is a real time database supported by google. In firebase, data are stored in table, or in more detail, data are stored in JSON file. Just like a socket, user can input a key and get the associated value with that key. In the new version of firebase, there is a database called Firestore, in Firestore, data are not necessary stored in table but page of data. There are two reasons we choose Firebase: First,

Firebase is a NoSQL database and it can greatly reduce the develop time. The process of setting up a new Firebase database may even take only few minutes. Second, Firebase can support a large number of diverse platforms, including mobile and the web. The following feature will tell how surveys are stored in Firebase, their structure and the naming format.

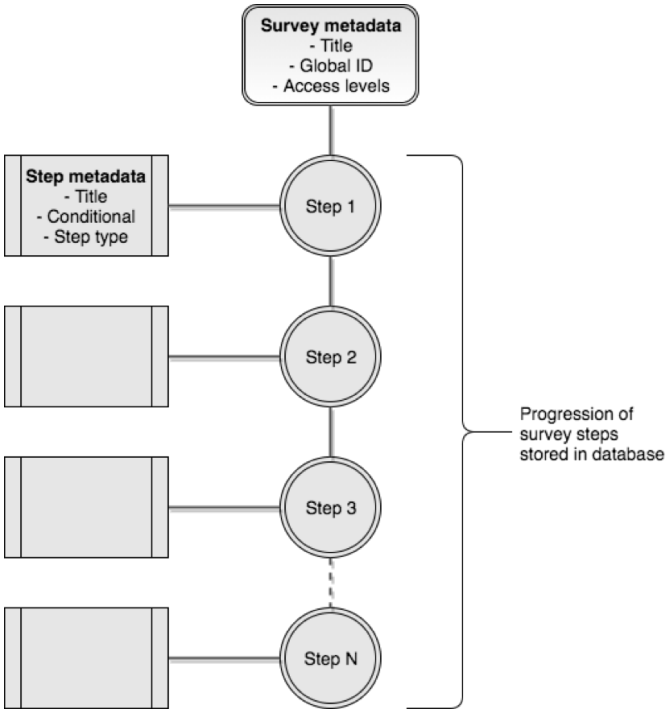


Figure 2 Progression of survey steps stored in database

Each survey is made of a set of questions or steps whose grammar are defined by extending the JSON format formally. Each step could be a point of input like multiple choice question, or a text slide, which describes a section of a survey. Each step may include its own configuration metadata, title, conditional and step type, but all steps must include its own unique id. And overall, all these steps are left generic to enable different interpretations of questions. For example, a yes or no may be combined with an image type question and if user choose yes, it may jump to a multiple choice question about that

image. In other word, researchers can add their new question types by slightly formatting the existing question steps, and the following feature shows how the surveys are actually stored in firebase.

Figure 3 The structure of surveys in Firebase

The second main component of the database is the answers data made by user. All user response data are stored in a place called data, which is at the same level with blueprints. Under each survey, there are answers made by each user. And each answer contains four main parts: which platform the answers generated from, (particularly in TigerAware Android, We will only upload android with its version number), the actually answers, timestamp and user ID.

Figure 4 The structure of answers in Firebase

The third part is the user information, including the email, the user name and the surveys they are taking.

Figure 5 The structure of users in Firebase

In a word, TigerAware Android will use Firebase, pulling surveys from Firebase and uploading answers into Firebase, and all operations are based on the storage structure we mentioned above.

2.2 ResearchStack

ResearchStack is an SDK (Software Development Kit) and UX (User Experience) framework for building research study apps on Android to help scientists and researchers build their research surveys and collect a variety of data they need in order to meet the requirement of the most scientific research.[4]

Researchers and developers can use ResearchStack to build basic Android applications without having a large extension of Android Programming skills, which means most of the objects are pre-defined

with their own fields and methods in ResearchStack. So researchers and developers can easily build a simple application by doing minor extensions or changes and Using JSON and HTML files. In addition, In TigerAware Android, we use Firebase to store the JSON files and import data from Firebase. This application can share information with users by showing the onboarding process (study overview/consent/registration) and collect data from users by showing a main activity screen with pre-scheduled tasks such as surveys with kinds of questions.

There are two main modules for building a ResearchStack application: Backbone and Skin.

Backbone: The core ResearchStack API. This includes tasks, steps, results, consent, file/database storage and encryption.

Skin: Above the structure of Backbone, Skin is built, which is based on Backbone but beyond Backbone. Compared to Backbone, Skin require less android knowledge to build a research app. Skin pull every component of backbone together and build the app in a more efficient way. Moreover, Skin is more compatible with ResearchKit's AppCore engine and skin is designed to work with minor changes to an existing ResearchKit™ iOS app's resources.

2.2.1 ResearchStack Backbone

ResearchStack Backbone is a API (Application Programming Interface) supported by ResearchStack, Backbone is the core building blocks of Research Stack which includes the survey engine, visual consent flow and active tasks [5].

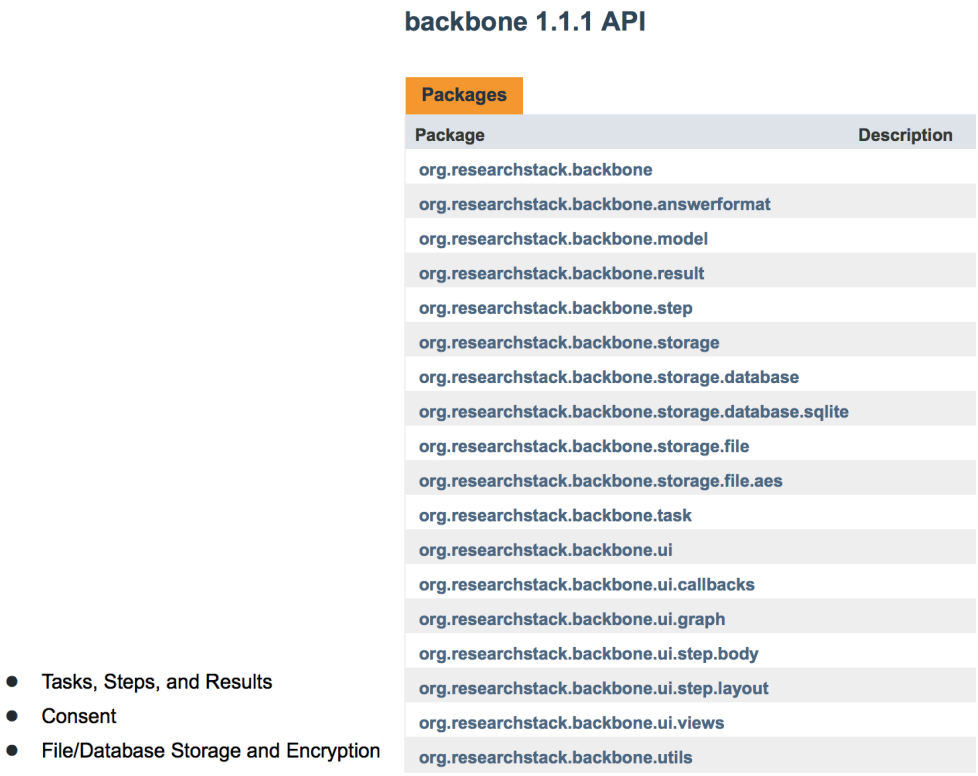


Figure 6 The backbone 1.1.1 API

(1) Tasks:

A task in ResearchStack contains a set of steps to present to users. Everything we mentioned above is in ResearchStack is represented as a tasks such as surveys, consent flows and actives tasks. By using ResearchStack Backbone, researchers can build surveys for modal presentation on android device, build consent page to inform users the detail of the study and obtain the signature from the participants and build active tasks page to lead a activity under semi-controlled conditions, gathering data from Android phone.

ResearchStack uses the class OrderedTask class to track a list of ordered steps for displaying the questions or contents to users. In ResearchKit, it has a according class called ORKOrderedTask.

```
OrderedTask task = new OrderedTask( identifier: "image_task", steps);
Intent intent = ViewTaskActivity.newIntent( context: this, task);
startActivityForResult(intent, REQUEST_AUDIO);
```

Figure 7 The programming structure of tasks

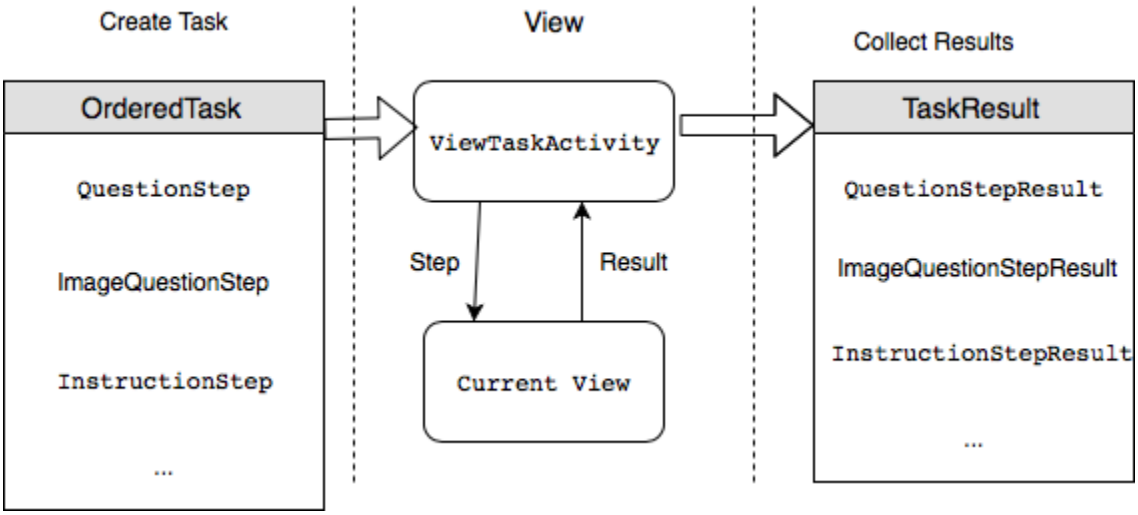


Figure 8 Task, steps, results structure

(2) Steps:

A step is a single component of task to complete a certain function. Different types of steps give different kinds of functions.

For example, an instruction step can inform the user the goal of this study or this page. A question step can ask user some questions and gather data from the user. Moreover, if we want to show user an audio_instruction_page, to give user more information about our study, here is how it works. Similarly, if we want to add a question, to let the user choose a,b,c and d,we could Similarly add a QuestionStep which include a specific question and add that step to the list.

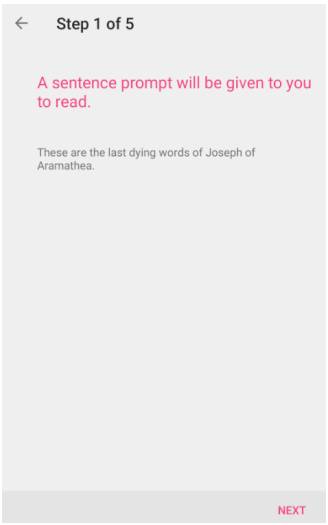


Figure 9 A sample instruction step

```
List<Step> steps = new ArrayList<>();

InstructionStep instructionStep = new InstructionStep( identifier: "audio_instruction_step",
        title: "A sentence prompt will be given to you to read.",
        detailText: "These are the last dying words of Joseph of Aramathea.");
steps.add(instructionStep);
```

Figure 10 The programming structure of step

(3) Results

In ResearchStack, there is no database associated with it. Thus, all results are stored in a class. called TaskResult.

An TaskResult object is a result that contains all the step results generated from task. After one run of a task or ordered task (that is, class `Task` or class `OrderedTask`), it will generate its own TaskResult. In other words, a task result is typically generated by the framework as the task proceeds. When the task completes, it may be appropriate to save the data by ourselves, upload them to server or database, or to immediately perform analysis on the data.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

    if(requestCode == REQUEST_CONSENT && resultCode == RESULT_OK) {

    }
    else if(requestCode == REQUEST_SURVEY && resultCode == RESULT_OK){
        TaskResult taskResult = (TaskResult) data.getSerializableExtra(ViewTaskActivity.EXTRA_TASK_RESULT);
        uploadResult(taskResult);
    }
    else if (requestCode == REQUEST_AUDIO && resultCode == RESULT_OK){

    }
    else{

    }


}
```

Figure 11 The structure of onActivityResult

2.2.2 ResearchStack Skin

Skin: Above the structure of Backbone, Skin is built, which is based on Backbone but beyond Backbone. Compared to Backbone, Skin require less android knowledge to build a research app. Skin pull every component of backbone together and build the app in a more efficient way. Moreover, Skin is more compatible with ResearchKit's AppCore engine and skin is designed to work with minor changes to an existing ResearchKit™ iOS app's resources. [6]

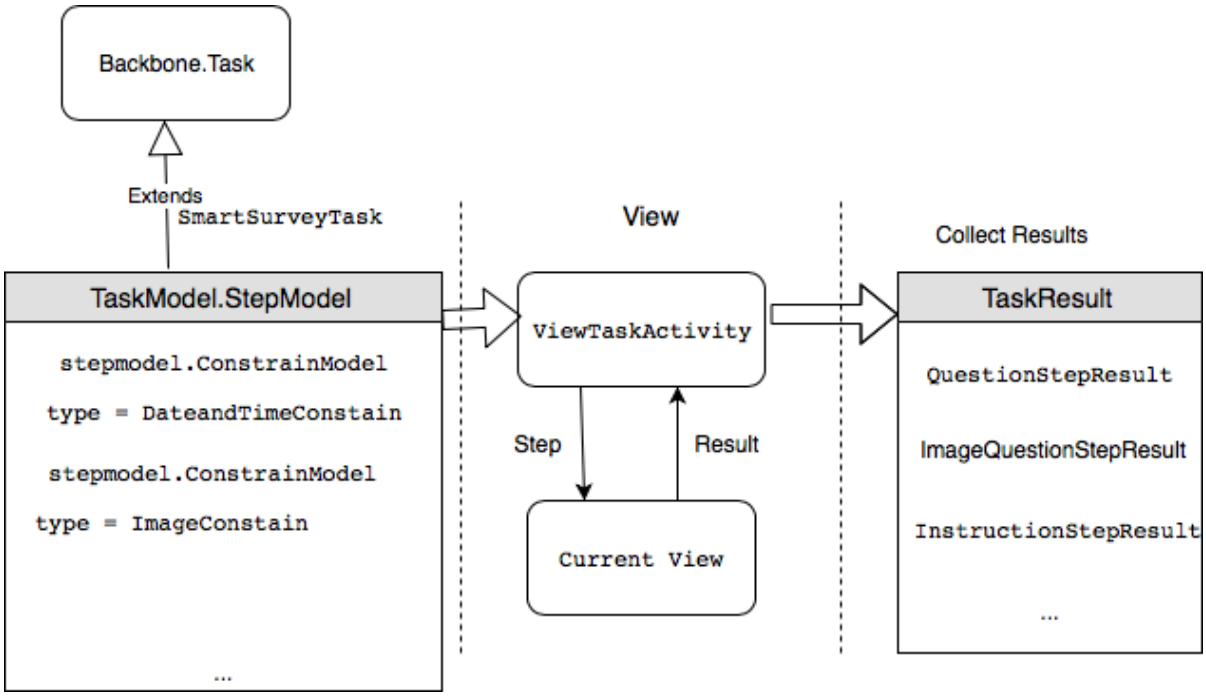


Figure 12 The structure of ResearchStack Skin

As we can see in this structure graph, Skin uses SmartSurveyTask while Backbone uses Task, and SmartSurveyTask is extended from Task. And Skin uses the same type StepModel with different constraints while Backbone uses different Step. Here is how it works. First, TaskModel get all the informations from local database or server and generate different StepModel based on what kind of questions they are. In other words, different StepModels are generated with different ConstrainModels. In this way, survey is generated. While the survey is in processing, different views are switched between each other. While the SmartSurveyTask is in processing, all the data are saved in TaskResults. When the task completes, it may be appropriate to save the data by researchers themselves, upload data to server or database, or to immediately perform analysis on the data.

2.3 ResearchKit

According to Apple ResearchKit, ResearchKit is an open source framework released by Apple. It allows researchers and developers to design powerful applications in medicine area. Using ResearchKit, researchers can build custom blocks to create the visual consent flows, surveys and real-time dynamic active tasks. Moreover, ResearchKit can work perfectly with HealthKit, which they can share data with each other [7]. In that way, researchers can visit more related research data, such as daily steps, calorie use and heart rate.

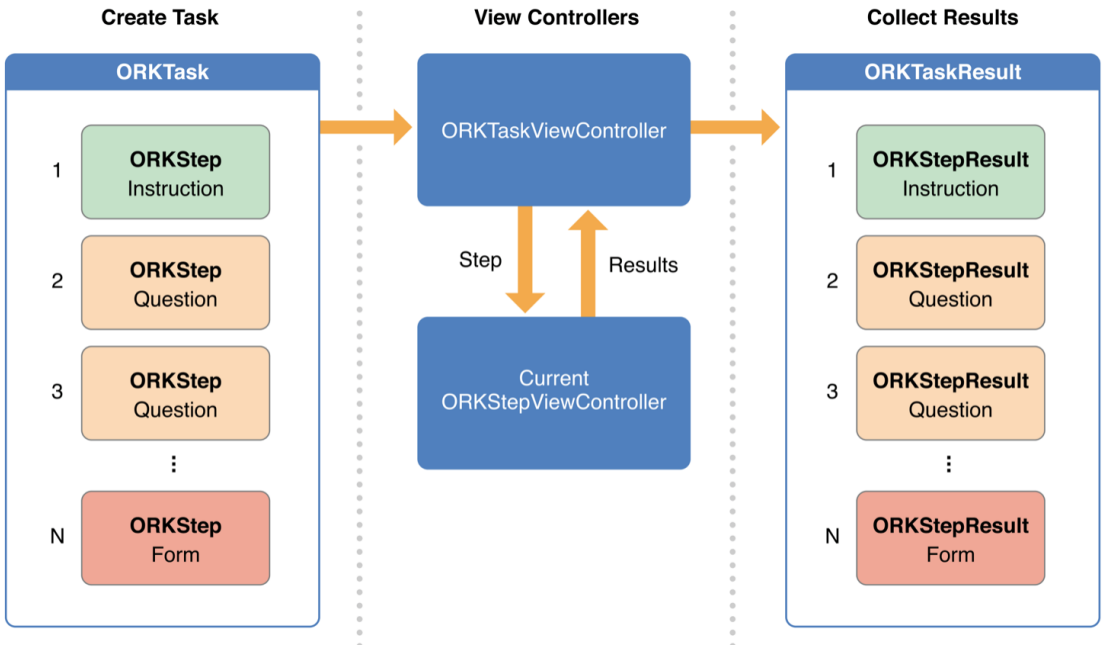


Figure 13 The structure of ResearchKit (cited from http://researchkit.org)

2.3.1 ResearchKit Active tasks

ResearchKit also has some active tasks, guiding users to do some performance which are under semi-controlled conditions, and data are collected by the iPhone. Here are the main seven

categories of active tasks based on what kind of sensor they use [8]. The seven categories are: motor activities, fitness, cognition, speech, hearing, hand dexterity, and vision.

Table 3 Active tasks in ResearchKit (cited from http://researchkit.org)

CATEGORY	TASK	SENSOR	DATA COLLECTED
Motor Activities	Range of Motion	Accelerometer Gyroscope	Device motion
	Gait and Balance	Accelerometer Gyroscope	Device motion Pedometer
	Tapping Speed	Multi-Touch display Accelerometer (optional)	Touch activity
Fitness	Fitness	GPS Gyroscope	Device motion Pedometer Location Heart rate

	Timed Walk	GPS Gyroscope	Device motion Pedometer Location
Cognition	Spatial Memory	Multi-Touch display Accelerometer (optional)	Touch activity Correct answer Actual sequences
	Stroop Test	Multi-Touch display	Actual color Actual text User selection Completion time
	Trail Making Test	Multi-Touch display	Completion time Touch activity
	Paced Serial Addition Test (PSAT)	Multi-Touch display	Addition results from user

	Tower of Hanoi	Multi-Touch display	Every move taken by the user
	Reaction Time	Accelerometer Gyroscope	Device motion
Speech	Sustained Phonation	Microphone	Uncompressed audio
	Speech Recognition	Microphone	Raw audio recording Transcription in the form of an SFTranscription object. Edited transcript (if any, by the user)
	Speech-in-Noise	Microphone	Raw audio recording Transcription in the form of an SFTranscription object Edited transcript (if any, by the user). This can be used to calculate the Speech Reception Threshold (SRT) for a user.

Hearing	Environment SPL	Microphone	Environment sound pressure level in dBA
	Tone Audiometry	AirPods Headphones	Minimum amplitude for the user to recognize the sound
	dBHL Tone Audiometry	AirPods Headphones	Hearing threshold in dB HL scale User response timestamps
Hand Dexterity	9-Hole Peg	Multi-Touch display	Completion time Move distance
Vision	Amsler Grid	Multi-Touch display	Touch activity Eye side Areas of distortions as annotated by the user

For example, a gait and balance task, can ask user to perform a walking task, user follow a movement instruction and their movement data is collected by the accelerometer and gyroscope sensor from their device. The following shows how gait and balance task is performed in more detail.

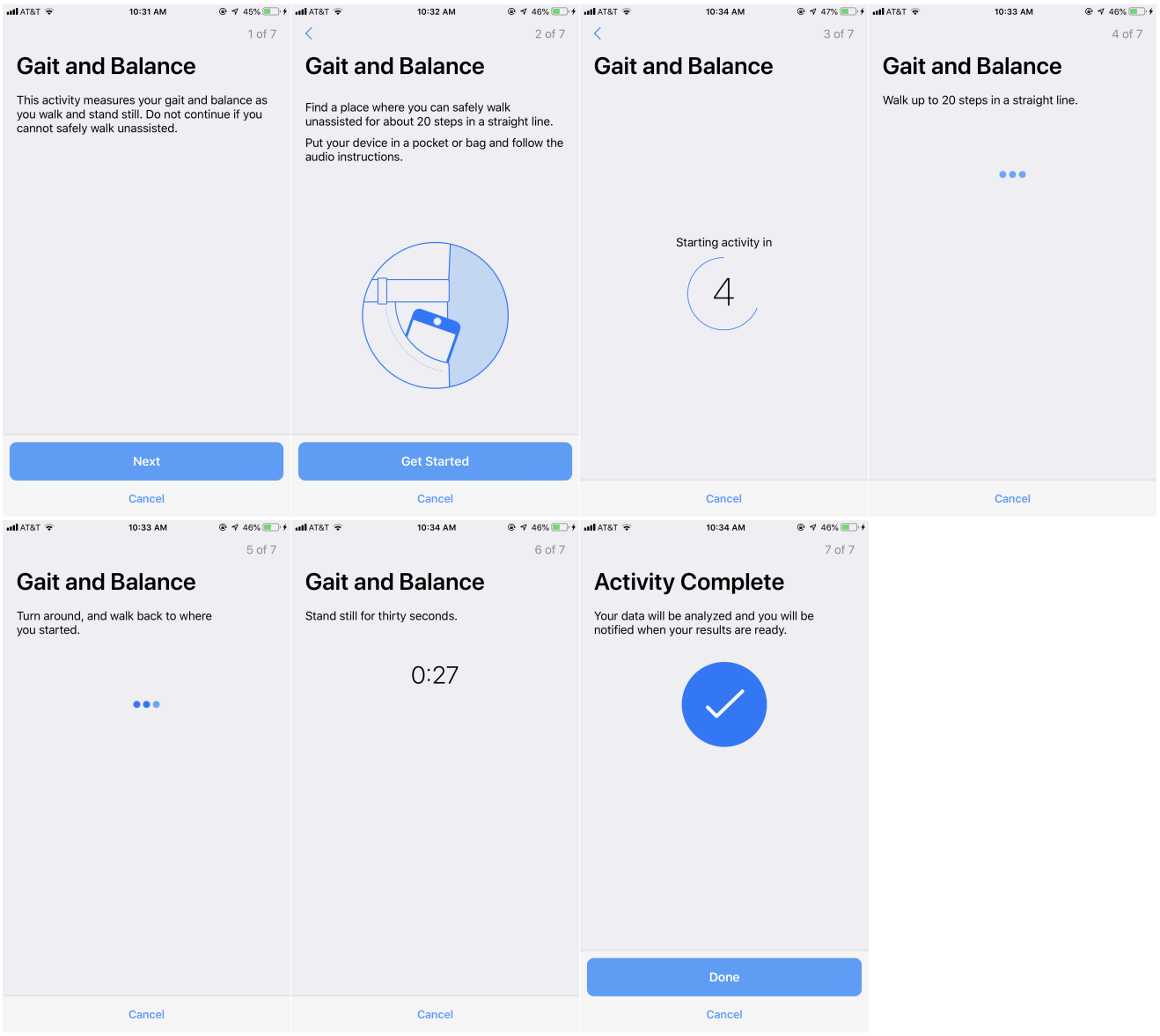


Figure 14 A gait and balance task

And Here are the three main data we gathered from the phone sensor:

(1) Accelerometer data:

Most smartphone device has three accelerometers, one along each accelerometer axis. The direction of the first accelerometer is along the sort side of the phone, the direction of the second

accelerometer is along the long side of the phone, while the third accelerometer is vertical to the plane formed by the first two accelerometers' axes.

All these data are represented in three dimensions and measured in the unit of "g", while g represents the force of gravity. If a device is dropping vertically without any force from outside, its accelerometer data should be [0,0,0], while a device is put horizontally on a flat table, its accelerometer data should be close to [0,0,-1]. Developers can define the speed of collecting rate and the number of samples to collect accelerometer data to achieve an instruction task.

(2) Gyroscope data:

Similarly to accelerometer, a device has Gyroscope sensors to measure the rotation of the device. The rotation vector represents the orientation of the device as a combination of an angle and an axis, in other words, a device has rotated through an angle θ around an axis (x, y, or z).

(3) Step data:

A device can use Step detector sensor to count how many steps user walked each time the step sensor is triggered. Particularly in android device, there are two step motion sensors: step detector and step counter. Both can count steps while step counter has more latency (up to 10 seconds) but more accuracy.

2.4 Relationships between ResearchStack and ResearchKit

First ResearchKit is built by apple, an open source framework for developers to make research apps. Accordingly, ResearchStack is built, the overall goal of researchStack is to make it easier to adapt the existing apps on iOS into android with the same functionality. Thus ResearchStack Backbone is built. However, though Backbone share almost the same functionality with ResearchKit, they have different

structures. Then Skin is built, based on Backbone but above Backbone, wrapped Backbone into some classes, that share the same structure with ResearchKit.

In a word, though the correspondence of features between the two SDKs isn't one-to-one, they will offer enough standard functionality(Backbone) and common frameworks(Skin). In TigerAware, both backbone and skin are used in order to reach the same functionality with IOS ResearchKit.

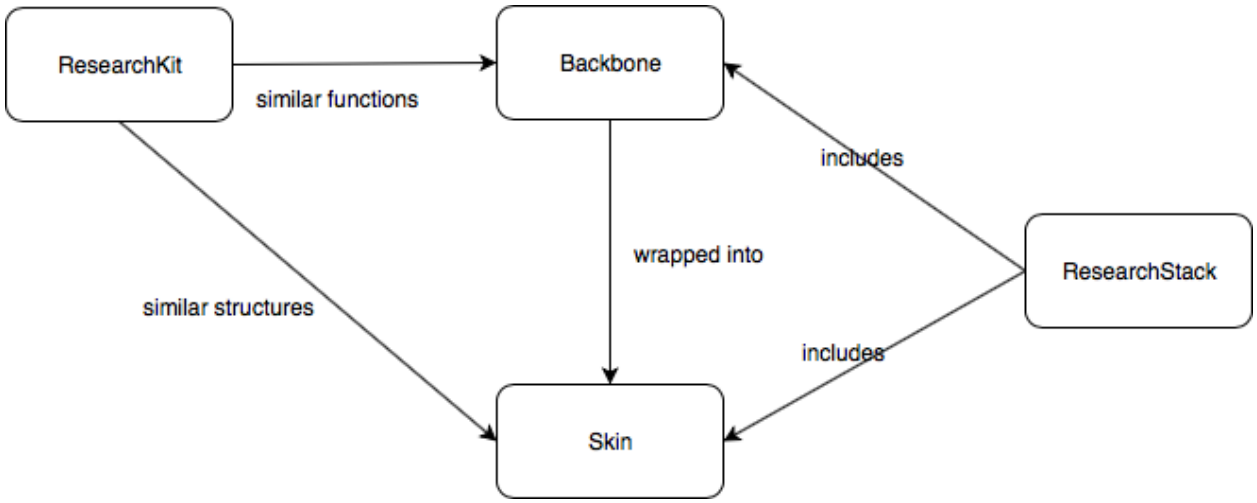


Figure 15 Relationships between ResearchStack and ResearchKit

3. TIGERAWARE ANDROID DESIGN

TigerAware Android is part of the whole TigerAware System, playing a significant role in TigerAware. In this chapter, we are going to introduce how the whole System works, what is the data flow inside TigerAware and go in more detail about TigerAware Android design.

3.1 TigerAware System architecture

TigerAware consists of four main part: TigerAware DashBoard, TigerAware database(Firebase), TigerAware IOS and TigerAware Android.

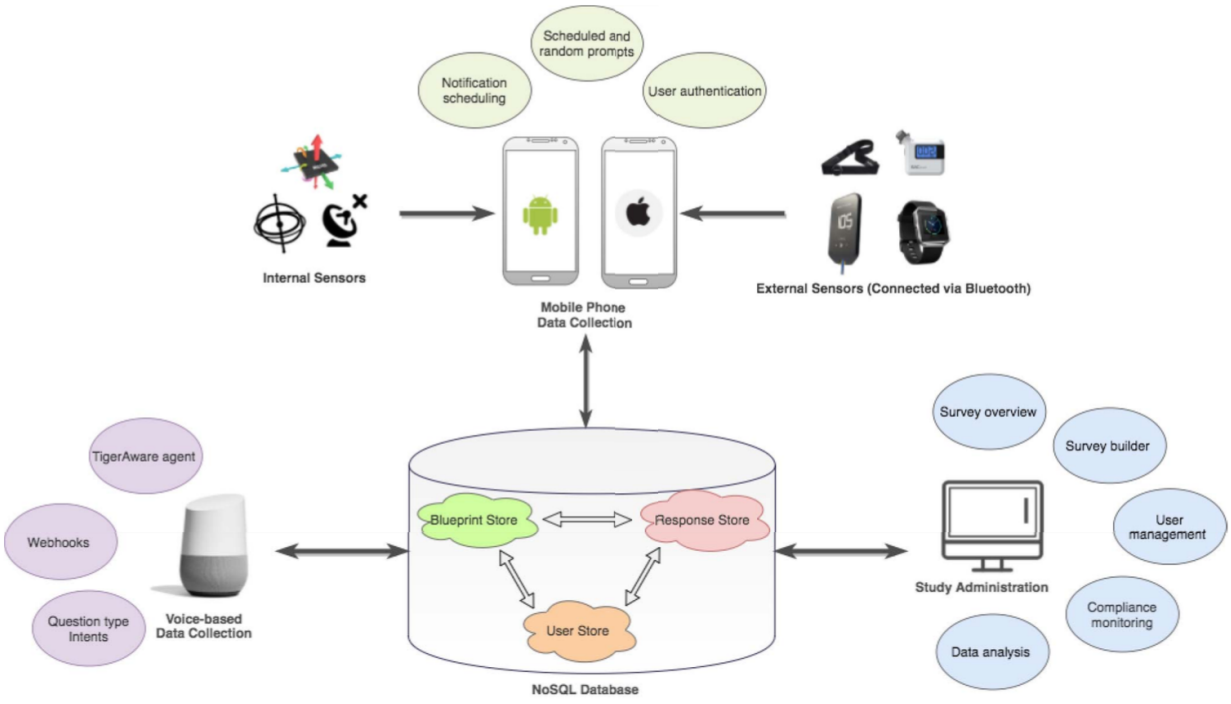


Figure 16 TigerAware system architecture and components

First, researchers can create a survey on TigerAware DashBoard, schedule surveys for participants. Then the surveys will be sent into Firebase, which is a real time database, as well as the

notifications and reminders about that survey. Then TigerAware Android will pull that data from Firebase, whenever it changes, TigerAware Android will update its data accordingly. Moreover, TigerAware Android will save that information and schedule surveys and notifications on its device. When it becomes the right time for participants to take the survey, surveys will pop up. After users taking that survey, TigerAware Android will send the results back into Firebase for later analysis and research study.

3.2 TigerAware Android architecture

3.2.1 Flow charts

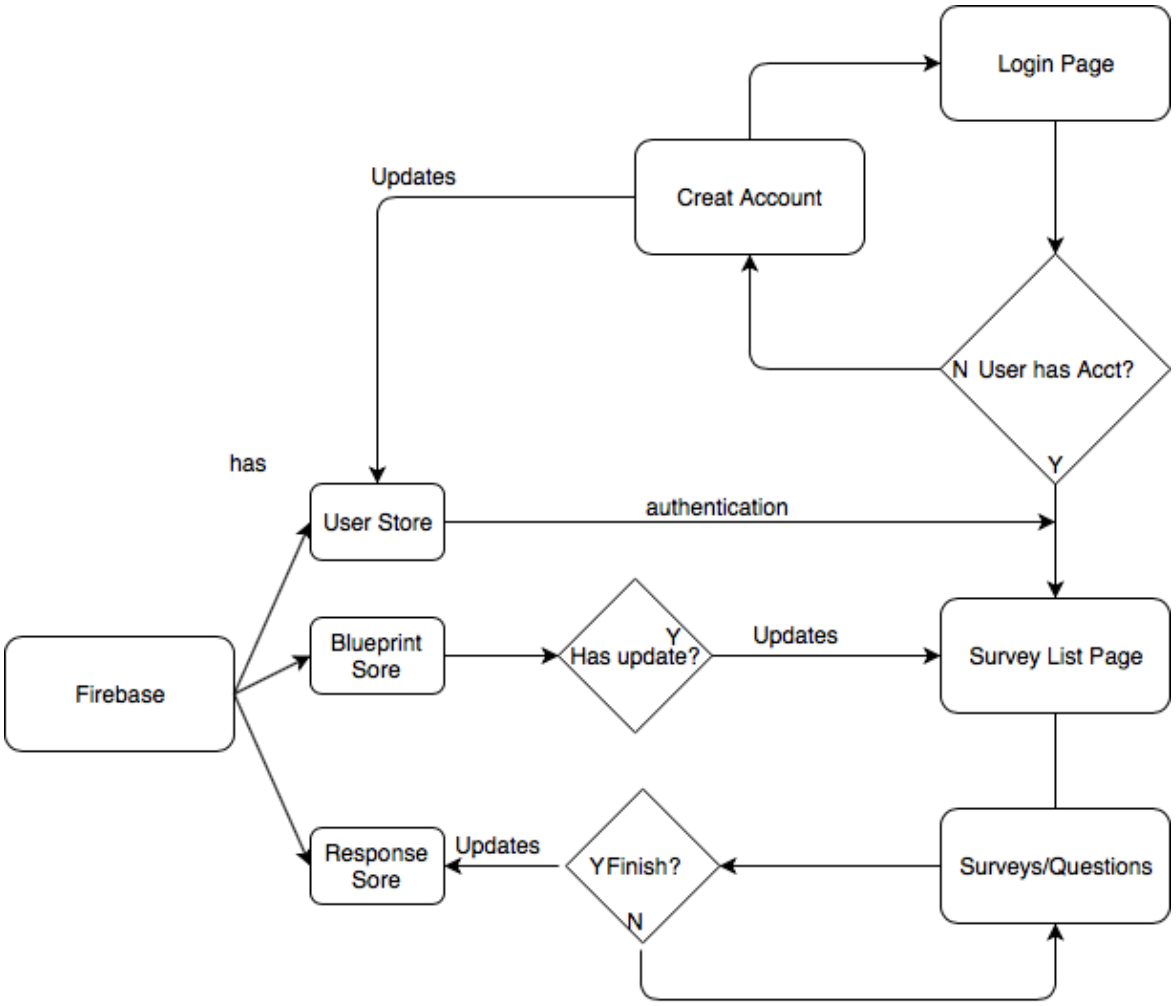


Figure 17 TigerAware Android architecture flowcharts

The first page the user will encounter will be a login page, to check the user has an account or not, if the user does not have an account, TigerAware will ask them to create one. If the user has one, TigerAware will use Firebase for authentication. After authentication, user will be prompted to the main survey page. In this page, TigerAware will show users a list of active public and private surveys. Once user click one of the surveys, kinds of questions will pop up. After user finishing taking the survey, TigerAware Android will send the results back into Firebase for later analysis and research study.

3.2.2 Authentication

So the first step comes to a user is the registration and the authentication step. In TigerAware, we use Firebase Authentication because Firebase supports an easy back end service for user authentication. That service can also support authentication by passcode, phone number, and Google, Facebook, Twitter, and so on. Particularly in TigerAware Android, we use authentication with Firebase using Password-Based Accounts.

Figure 18 Firebase

Work theory:

Firebase API provides a class called FirebaseAuth, every time users want to create an account or sign in, TigerAware Android will get their username and password as string, and by using authentication functions from that API, Firebase will check or create a account for users.

```
mAuth.createUserWithEmailAndPassword(username, password)
        .addOnCompleteListener( activity: SignUpActivity.this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                    Log.d(TAG,  msg: "createUserWithEmail:onComplete:" + task.getResult().getUser().getUid() + "-" + email);
                      Toast.makeText(SignUpActivity.this, task.getResult().getUser().getProviderId(), Toast.LENGTH_SHORT).sh
                    showToast("Sign Up Success!");
                    DatabaseReference mDatabase = FirebaseDatabase.getInstance().getReference();
                    DatabaseReference answersChild = mDatabase.child("users").child(task.getResult().getUser().getUid());
                    Map<String, Object> answerObjMap = new HashMap<>();
                    answerObjMap.put( k: "email", username);
                    answerObjMap.put( k: "username", email);
                    answersChild.setValue(answerObjMap);
                    finish();
                } else {
                    showToast("Sign Up Failed!");
                }
            }
        });
```

Figure 19 Process about signInWithEmailAndPassword

```
mAuth.signInWithEmailAndPassword(username, password)
        .addOnCompleteListener( activity: LoginActivity.this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                hideHUD();
                Log.d(TAG,  msg: "signInWithEmail:onComplete:" + task.isSuccessful());

                // If sign in fails, display a message to the user. If sign in succeeds
                // the auth state listener will be notified and logic to handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Log.w(TAG,  msg: "signInWithEmail:failed", task.getException());
                    Toast.makeText( context: LoginActivity.this, "Login Failed!",
                            Toast.LENGTH_SHORT).show();
                }
            }
        });
```

Figure 20 Process about signInWithEmailAndPassword

In the figures above, mAuth is an object from class FirebaseAuth. Once users successfully create their accounts, their user information will be stored in Firebase under authentication section. Research can also manually add, delete and search users by their email address or user UID.

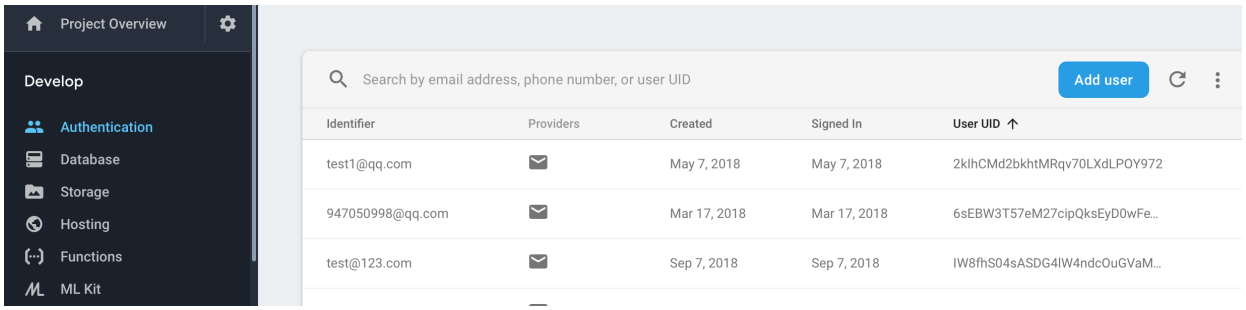


Figure 21 Firebase authentication

3.2.3 Parse Survey

After users successfully login in, the second page comes to their minds is the main survey list page. We talked about the flow charts above, basically TigerAware will send surveys to users and users will send their answers back to TigerAware, but how does it happen in more detail? As we mentioned before, all surveys are stored in Firebase in the following format.

Figure 22 Users in firebase

Figure 23 Blueprints in firebase

Developer can input a key and get the associated value with that key by some certain programming language. In TigerAware, we use JAVA and by using TigerAware API, we can simply get the associated value with the key we input, the following is the programming theory in more detail.

```
FirebaseDatabase.getInstance().getReference().child("users").child(FirebaseAuth.getInstance().getCurrentUser().getUid()).child("taking").addChildEventListener(ne
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String s) {
        FirebaseDatabase.getInstance().getReference().child("blueprints").child(dataSnapshot.getValue(String.class)).addValueEventListener(new ValueEventListener
            @Override
            public void onDataChange(DataSnapshot dataSnapshot2) {
                SurveyUtil.activity = MainActivity.this;
                SurveyUtil.parseSurveys(dataSnapshot2, mPrivateSurveys);
                for (SurveyFragment fragment : mTabs) {

                    MainSurveysTab mainSurveysTab = (MainSurveysTab) fragment;
                    mainSurveysTab.launchID = launchID;
                    if (mainSurveysTab.mView == null) {
                        fragment = MainSurveysTab.activeInstance;
                    }
                    fragment.onDataChange(mPrivateSurveys, mPublicSurveys);
                    ((MainSurveysTab) fragment).launchHotSurv();
                }
                hideHUD();
            }
```

Figure 24 Programming of get instance of FirebaseDatabase

First, by using Firebase Database, researchers can get users, UID, as well as the surveys they are taking. Every user is supposed to have its own unique user id and every survey is also designed to have its own unique survey id.

Then after getting the unique survey ID, which users are currently taking in user section, the next step is to go blueprints section and get the surveys based on the survey id we got.

Finally, we can parse that survey as well as update the UI or the survey list. Every time if there is a new survey coming to the user, or there are new data under user-taking, it will parse that survey as well as update the UI or the survey list again.

```
String surveyName = (String) surveyBody.get("name");
curentName = surveyName;
String surveyUser = (String) surveyBody.get("user");
Boolean hidden = safeGetBool(surveyBody,  key: "hidden");
List<Map<String, Object>> questionsBody = (ArrayList) surveyBody.get("survey");
List<Question> questions = parseQuestions(questionsBody);
Survey survey = new Survey(surveyId, surveyName, questions, surveyUser);
```

Figure 25 Parse survey section 1

This is the parse survey section, in this section, it calls a function called parseQuestion to help divide the whole parsing process in more detail. TigerAware Android will parse all the information for that survey and store the information in String or List, depending on what kind of information it is. Then use another function to parse questions to suit ResearchStack Format.

```
private static List<Question> parseQuestions(List<Map<String, Object>> questionsBody) {
    List<Question> questions = new ArrayList<>();
    for (Map<String, Object> questionBody : questionsBody) {
        Question question = null;
        List<String> choices = null;
        Boolean multiSelect=null;
        String conditionId = safeGetString(questionBody, key: "conditionID");//(String) questionBody.get("conditionID")
        String id = safeGetString(questionBody, key: "id");//(String) questionBody.get("id");
        String on = safeGetString(questionBody, key: "on");//(String) questionBody.get("on");
        String title = safeGetString(questionBody, key: "title");//(String) questionBody.get("title");
        String subtitle = safeGetString(questionBody, key: "subtitle");
        String type = safeGetString(questionBody, key: "type");//(String) questionBody.get("type");
```

switch (type)

```
case DATE_TIME_QUESTION_TYPE_DESCRIPTION:
    Boolean both = safeGetBool(questionBody, key: "both");;
    Boolean prior= safeGetBool(questionBody, key: "prior");;
    Boolean multiple=safeGetBool(questionBody, key: "multiple");
    question = new DateTimeQuestion(id, title, subtitle, conditionId, on,both,prior,multiple);
   // System.out.println(question);
    //both - date and time
    //prior - select date prior to current date
    //multiple - multiple date
    break;
```

Figure 26 Parse survey section 2

Within parsing survey function, TigerAware Android uses parsing question function. Above is the parsing question section. For each question in that list or in that survey, we get their question id, title, type and so on. Then based on what kind of information it contains, we encapsulate them into ResearchStack question format to generate actual questions. For example, if a question contains its type as DATE_TIME_QUESTION, TigerAware Android will generate a data and time question accordingly.

After we adding all questions into List<Question>, the last step is to launch surveys, and the following are the ERD and the work flow chart for the storage structure we mentioned above.

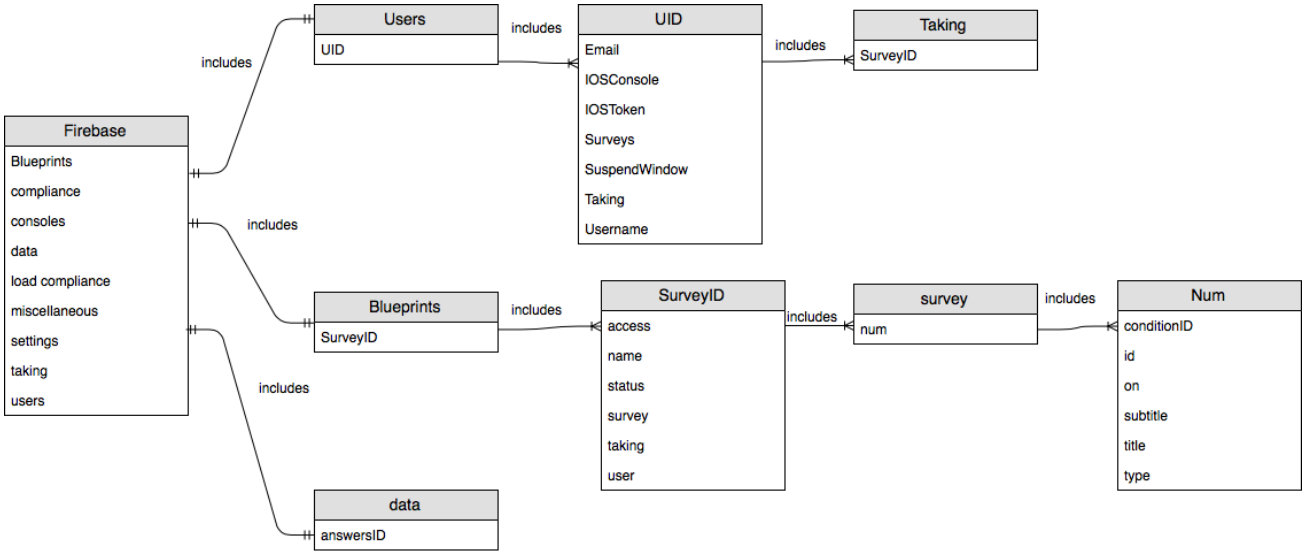


Figure 27 ERD for the storage structure

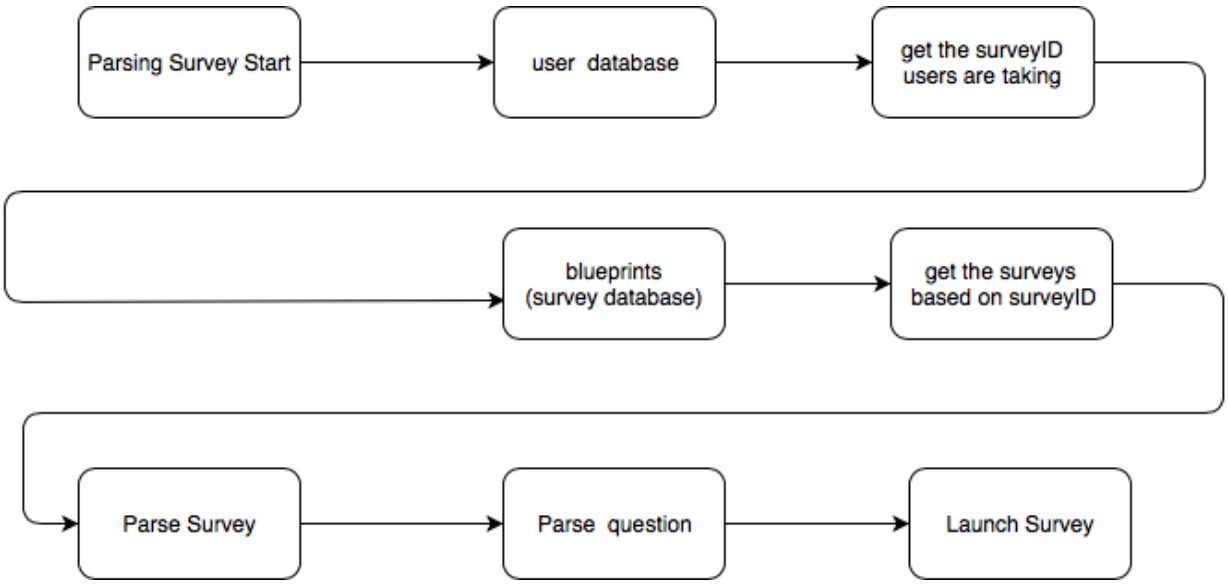


Figure 28 Flowcharts for the storage structure

3.2.4 Launch Survey

As we mentioned in 3.2.3, after parsing surveys, all question are stored in surveys in certain format and waiting for launching, and the following is the flow chart we mentioned above.

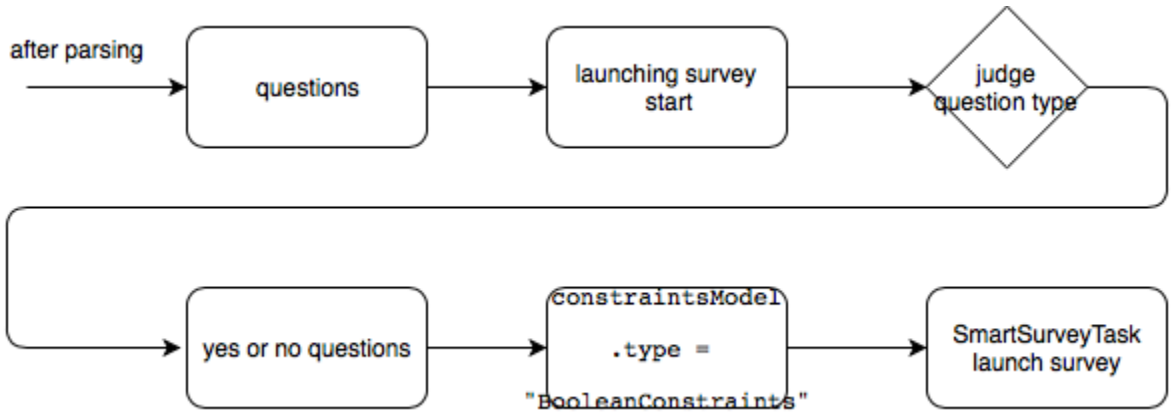


Figure 29 Flowcharts for launching survey

First, after parsing process, all questions are stored in surveys in certain format. Then as soon as we launch or a user click one of the survey in the survey list, the launching survey function will start. The first step of the function is to judge what kind of question it is, let us take yes or no question as example, if a question is regarded as a yes or no question, the launching survey function will set the constrainsModel.type as booleanConstraints, which is only a custom name of that type, in other words, we can just take it as a name tag. The last step of launching survey function is to use ReserchStack API and the SmartSurveyTask, a main function supported by one of its main core, ResearchStack Skin, and the booleanConstraints we set will be used in this SmartSurveyTask. In SmartSurveyTask, similarly, it will first divide different questions into different types based on their constraints. Then for different questions, SmartSurveyTask will define different answer format according to that type of question. The answer format and the question step are both defined and supported by the other core api of

ResearchStack, which is not in ResearchStack Skin but in ResearchStack BackBone. In this way, both ResearchStack BackBone and ResearchStack Skin are fully implemented and cooperated with each other.

3.2.5 Upload results

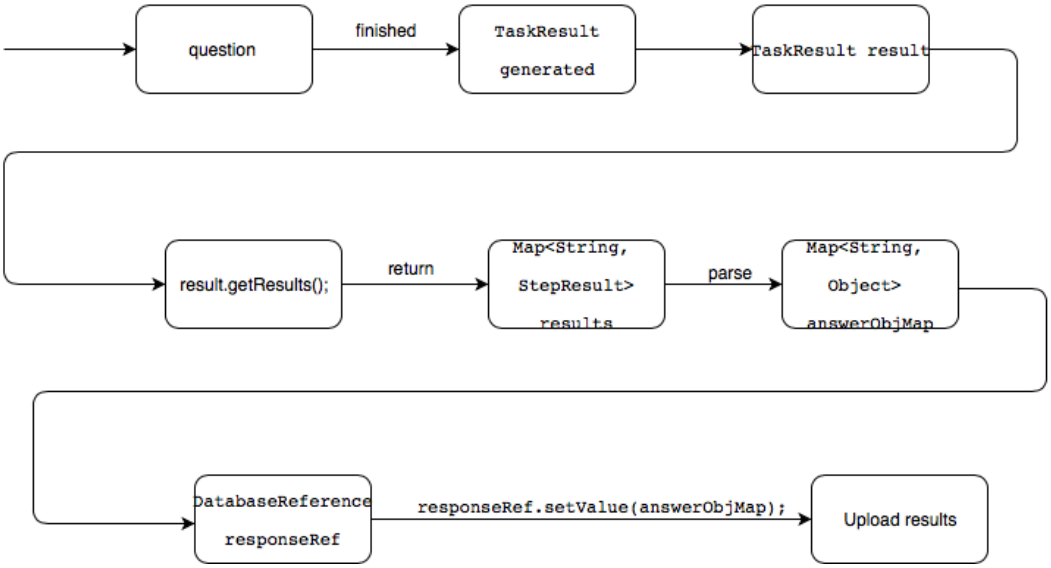
After a user finish one question, his answer will be uploaded into Firebase. In TigerAware Android, it uses TaskResult, which is supported by BackBone in ResearchStack, to collect all the answers.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    // 如果返回为OK时才处理结果
    SurveyUtil.inSurvey = false;
    SurveyUtil.delayedBool = true;
    if (resultCode == RESULT_OK) {
        TaskResult result = (TaskResult) data.getSerializableExtra(ViewTaskActivity.EXTRA_TASK_RESULT);

        MainActivity activity = (MainActivity) getActivity();
        DatabaseReference database = activity.getDatabase();
        DatabaseReference dataChild = database.child("data");
        DatabaseReference answersChild = dataChild.child(result.getIdentifier()).child("answers");
        // 创建Answer存储对象
        Map<String, Object> answerObjMap = new HashMap<>();
        // 存储结束的时间
        SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ssZ");
        String timeString = formatter.format(result.getEndDate());
        answerObjMap.put("timeStamp", timeString);
        // 存储用户id
        FirebaseAuth auth = FirebaseAuth.getInstance();
        answerObjMap.put("userID", auth.getCurrentUser().getUid());

        answerObjMap.put("AndroidVersionNum",1.352);

        // 获取答案
        Map<String, Object> answersMap = new HashMap<>();

        Map<String, StepResult> results = result.getResults();
```

Figure 30 On Activity Result

As we mentioned in BackBone introduction section, a TaskResult object is a result that contains all the step results generated from task. After one run of a task or ordered task (that is, class `Task` or class `OrderedTask`), it will generate its own TaskResult. In other words, a task result is typically generated by the framework as the task proceeds. When the task completes, the answer will be recorded in taskresult, which has a function called getResults. By calling that function, a hashmap will be generated. Hashmap will map the key which is a string, to an object, which is a StepResult class. Then we can parse the StepResult based on what kind of question it is. After parsing

The following is the flow charts for uploading results.



Figurer 31 Flow charts for uploading results

3.3 Site maps

Figure 32 First activity

This is the first activity comes into user, the login page. In this page, user can either login with existing account and password or create an account.

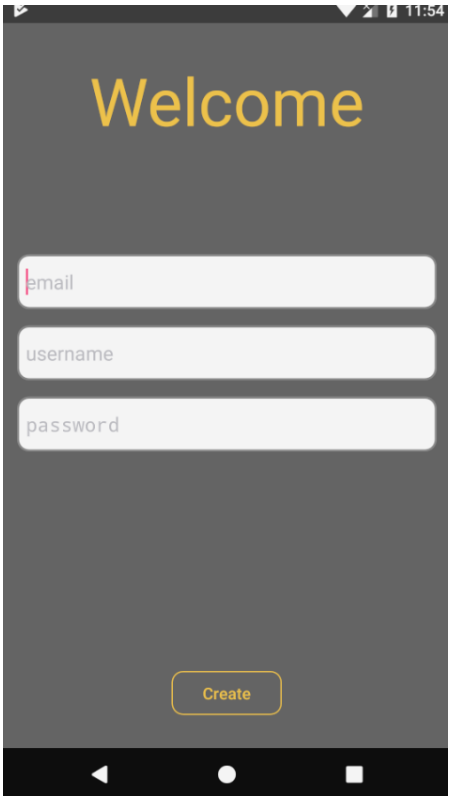


Figure 33 Registration page

Above is the registration page, if use do not have an account, TigerAware will lead user to this page to help them create an account with email, username, and according password.

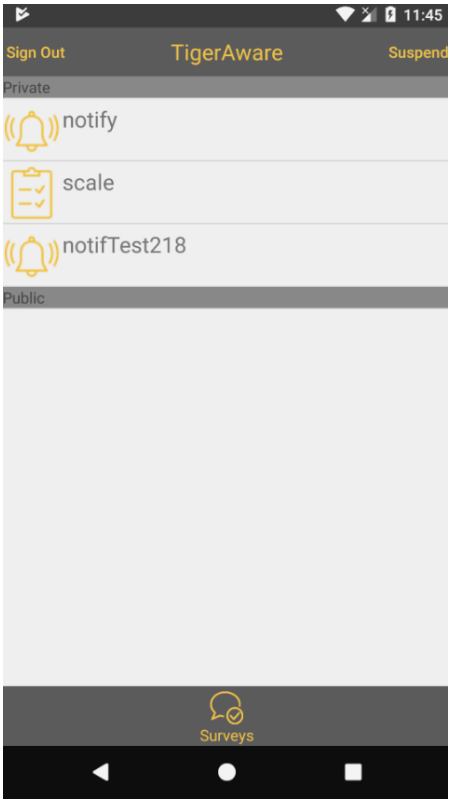


Figure 34 Main activity

This is the main activity, when people successfully login in, the main page with show up with the survey list, and two other buttons. One is the sign out button, which will lead user back to the first page for the security purpose. Another is the suspend window, in suspend window, user will not receive any notification.

Figure 35 Survey page

This is the survey page, including different kinds of questions, which is the core function of TigerAware Android. In the next chapter, we will talk about the exact question types, the answer format and how it implemented.

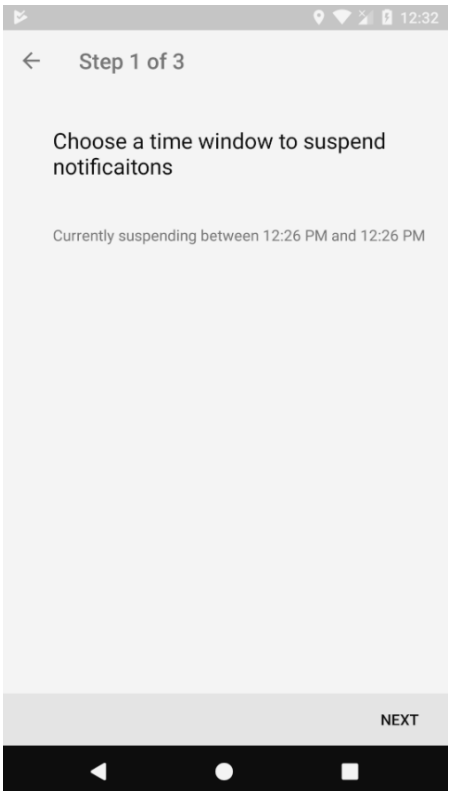


Figure 36 Suspend window page

This is the suspend window page, user can choose a time window to suspend notifications. In this suspend window, user will not receive any notification.

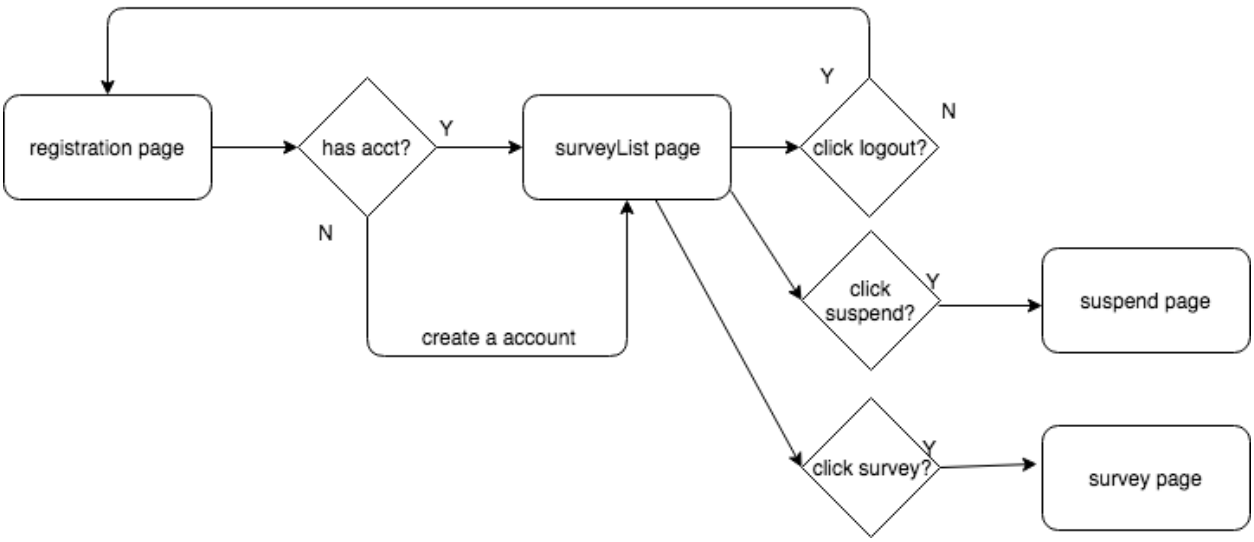


Figure 37 Site map

3.4 Functions and algorithms

In this chapter, we are going to introduce three main functions in TigerAware Android: questions, suspension, and notifications. Then, we will discuss how suspension and notification are implemented and the algorithms behind them.

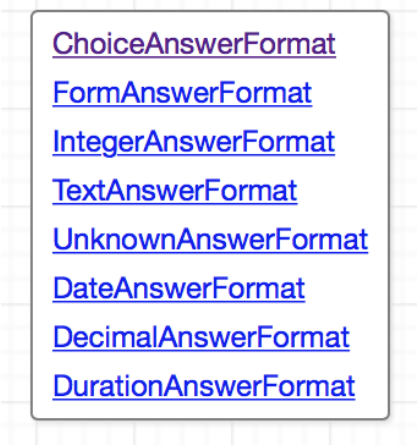
3.4.1 Question types

In TigerAware Android, most question types are supported by ResearchStack in order to have the similar functionality with TigerAware IOS, which is using ResearchKit. For those questions that are not supported by ResearchStack but are implemented in ResearchKit, ResearchStack offers a custom interface for user to design a new question type based on the research study purpose.

In the table below, we can see there are seven question types or question answers supported by ResearchStack, and the UnknownAnswerFormat is the custom interface for developers to design their own questions.

As for TigerAware, in total, TigerAware Android suppports twelve question types so far, they are: yes or no question, multiple choice question, date and time question, text slide question, text field question, time of day question, time interval question, scale question, continuous scale question, BACTrack Alcohol Sensor question, number type question, active location question, active gait question and active PSAT question.

Table 4 ResearchStack Supported Anwersformat



(1) Yes or no question

Definition: A yes or no question is a question whose expected answer is “yes” or “no”. Formally, they present an exclusive disjunction, a pair of alternatives of which only one is acceptable. For example, a yes or no question could be like that “Did you drink alcohol today?”

Implementation: Yes or no question is predefined in ResearchStack, as a basic question, TigerAware Android uses it directly with minor changes.

(2) Multiple choice question

Definition: A multiple choice question is a question whose expected answer could be only one choice, or one or more choices, defined by the researcher based on different study.

Implementation: For multiple choice question, TigerAware Android makes some slight changes. In ResearchStack, it has two different question types or answer formats, one is single choice answer format, which can only allow user to choose one and only one choice. Another is multiple choice answer format, which can allow user to choose one or more than one choices. However, for better

implementation, or a more convenient way, TigerAware Android combines these two questions into one. This is how it works, when researchers are scheduling surveys on the TigerAware dashboard for multiple choice question, they can also choose allow multiple choice or not, if they click yes, a boolean variable called allowMulti will be passed to TigerAware along with questions. Thus we defined a new constraint type called allowMulti, if allowMulti equals true, it will select multiple choice answer format otherwise it will select single choice answer format.

(3) Date and Time question

Definition: A date and time question is a question whose expected answer is “yes” or “no”. For example, an alcohol based on study could ask participants what was the last time they drink alcohol, and they need to specify what day associated with the time as well. The answer should be a calendar in which the user can select a day, then TigerAware will display a clock for user to select a time as well

Implementation: Date and time answer format is predefined in ResearchStack, which means the clock and the calendar are supported. However, For date and time question, TigerAware Android makes some slight changes. First, TigerAware Android changed the time format from 24 hours into 12 hours with AM/PM.

Second, TigerAware Android offers three more options for this question, just like how it works in multiple choice question we mentioned above, the researcher can disable or enable these three options in the dashboard, then three parameters will be passed to TigerAware Android with the question itself. One is “select both date and time”, if researcher enable this option, the answer would be date and time, otherwise the answer will only be a certain day. Another is “select past days”, if researcher enable this option, the participants could select past days instead of select the future days. The last one is “select

multiple days", if researcher enable this option, when participants finish one date and time, they could choose another date and time as well.

Future work: For current studies, in order to not confuse users and researchers, we manually disabled "select multiple days", and enabled "select both date and time" and "select past days". In order to fully use this question and make it consistent with what researchers selected and scheduled in the dashboard, we should not fix these options when we are ready to use.

(4) Text slide question

Definition: A text slide question is a question or an informative page to give user information, there is no answer required for user to input or type back to this question. Usually, the text slide question will only contain text content, but in TigerAware Android, we modified this question and made it compatible with image content.

Implementation: text slide question is predefined in ResearchStack, as a basic question, TigerAware Android uses it while adding image type content with text slide content to make this question more flexible and contain more information. For current study, we save image at local space and when there is a text slide question coming, as we mentioned above, there will be a "enable image" option. If user enable this option and schedule an image content as well as test slide, they will show together in the text slide question.

Future work: If we want to use image content in more general space, not like using few certain images for certain studies, we have to store all images in database and read these images with questions. A good way is to store images in Firebase storage and store the associated URL in the Firebase database.

(5) Text field question

Definition: A text field question is a question whose expected answer is a text field, which means users need to type their answer for these question. Usually the answer will be more general and not easy to predefine, or it is hard to be limited into A,B,C and D four options. For example, a question can be like “What did you have this morning?”

Implementation: Text field question is predefined in ResearchStack, as a basic question, TigerAware Android uses it directly with minor changes.

(6) Time of day question

Definition: A time of day question is a question whose expected answer is a timestamp. For example, a time of day question could be like “When was the last time you had alcohol today?” Similarly, to day and time question, one is asking for day and this question is asking for time.

Implementation: time of day is predefined in ResearchStack, as a basic question, TigerAware Android uses it directly with minor changes.

(7) Time interval question

Definition: A time interval question is a question whose expected answer is a period of time. The purpose of this question is ask for a time interval between two activity. For example, a time interval question could be like “How long do you relax between questions?”. As for the answer, the user can select hours from 0-23 and select minutes from 0-59, and the answer will combine them together.

Implementation: time interval is predefined in ResearchStack, as a basic question, TigerAware Android uses it directly with minor changes.

(8) Scale question

Definition: A scale question is a question whose expected answer is a degree. To achieve this question type, researcher will define a scale bar with numbers, for example, if a question is to determine whether users are happy or not, a scale bar can be defined from one to five, and one stands for not happy, five stands for very happy, and the user can drag the scale bar to the degree associated with their true situations.

Implementation: scale question is not support in ResearchStack Backbone, but it is supported in ResearchStack Skin. Before this question type, TigerAware Android was using ResearchStack Backbone, we can call it TigerAware Android 1.0. Then in order to add this question type, we changed TigerAware Android from 1.0 to 2.0 by refining the system structure and using ResearchStack Skin instead, and ResearchStack Backbone is still in use as the major component, which offering the answer format in ResearchStack Skin.

(9) Continuous scale question

Definition: A continuous scale question is a question whose expected answer is a degree. To achieve this question type, researcher will define a scale bar with numbers. Differently with scale question, a scale question can only start from one to up to eight degrees, and display all names with that degree. For example, if a question is to determine whether users are happy or not, a scale bar can be defined from one to three, and one stands for not happy, two means not sure yet, and one stands for very happy, and the user can drag the scale bar to the degree associated with their true situations. While in continuous question, the researchers can only define the minimum string and the maximum string, which is “not happy” and “very happy”. In addition, the continuous scale questions can hold more

degrees starting not only from one. For example, a continuous question can be from 100 to 300 and a minimum string with a maximum string.

Implementation: A continuous scale question is predefined in ResearchStack which sharing the same framework with scale question. We added the maximum string and minimum string and the maximum value and minimum value by reading them from firebase. We also changed the display string by adding the minimum value number into the display string number which starts from zero. For example, if we use scale question, and the question is from 11 to 15, the display number will be 1-5, but if we use continuous scale question, the display number will be 11-15 instead.

(10) Number type question

Definition: A number type question is a question whose expected answer is a number. To achieve this question type, researcher will define a number pad with numbers. The range of number is from zero to nine. The purpose of this question is to ask users their information which has a large range, and also, when researchers are scheduling this question type, they can also define a unit type associated with the number answer. For example, a question could be defined like this, what is your age, and the unit type could be years.

Implementation: a number type question is predefined in ResearchStack, as a basic question, TigerAware Android uses it directly with minor changes.

3.4.2 Notifications

In TigerAware Android, one major function is notification. Research studies usually want to make surveys scheduled at random time in order to get a general understanding of participants general lives, and they also want to schedule notifications at certain time to remind the participates to take their

survey in time. When researchers are scheduling surveys, they can also schedule both notifications associated with that survey, so that it makes the survey taking process more efficient, or some participants may miss some surveys without notifications, and as we mentioned above, there are two kinds of notifications mechanisms in TigerAware Android, random notifications and scheduled notifications.

(1) Random notifications

Definition:

Random notification plays a significant role in TigerAware Android notification system, when researchers are scheduling surveys, they can also schedule random notification with random notification start time, random notification end time, how many days it will last and how many random notification will be sent out during that time.

Mechanism：

Random notifications are also treat as questions, to see how it works, please refer to section 3.2.3 Parse Survey and section 3.2.4 Launch Survey in questions. Then after the question data is processed, if the question type equals random notification, it will generate a random timestamp between the start time and the end time.

In addition, in order to save memory and improve the working efficiency, every time the application is opened, it will only schedule notifications in the next two days in advance. To achieve this function, a few comparison is added into the make notification function.

Above all, in order to schedule notification every day till the last day, it will start scheduling notifications from the next day and increase the date by one till the last day. Then after the date data is received in the make notification function as well as the start time, end time and the last day, the following comparison mechanism will happen to keep every aspects in notifications working as expected.

(1) First, it will judge whether right now is before notification date, to ensure the notification is scheduled in the future.

(2) Second, it will judge whether the number of notification scheduled is less than five hundred, to make sure the notifications are not overloaded for the memory storage.

(3) Then it will make sure notification date is before two days after today, which means every time a participant open the application, TigerAware Android will only schedule two days notification in advance instead of scheduling all the notifications till the last day at one time. In this way, it will save space for the memory storage and improve the working efficiency for the notification system.

(4) Last but not least, it will also check whether the notification date and time are in suspend window. Suspend window is a window during which the notifications cannot be received, we will introduce suspend window in more detail in the following chapter.

Algorithms:

To ensure each random notifications are scheduled at least fifteen minutes from each other, there are two algorithms to solve this problem:

(1) The latest start time algorithm:

This the algorithm always calculates the latest start time for the first following notification. For example, if researcher want to schedule 5 random notifications from 8:15am to 10:15am, which means there are 4 times breaks of 15 mins. So the latest start time for the first notification is 10:15am - 4*15mins=9:15am, otherwise there would not be enough time for the left notifications. now we just random choose a time between start time 8:15am and the latest start time 9:15am as the first notification time. For example, if we choose 8:35am as the first notification, to guarantee 15 mins break, the new start time should be 8:35am+15mins=8:50am. Now we have new start time 8:50am ,end time 10:15am and 4 times notifications left. Then loop till all notifications are scheduled.

(2) The conditional random selection algorithm:

Similar to reservoir sampling, this algorithm lists all the possible timestamps to select in an array, then remove a 15 minute radius from that array. For example, there are 120 minutes from 8:15am to 10:15am, then it creates an array or list of sorts of all 120 digits and randomly select an index in that array. After that it would remove a 15-minute radius from that array. So if it chose 97 then it would remove 97 and all the numbers from 82-112. Then select another random index in that array, which would exclusively be full of valid numbers. To transfer it back to timestamp, it would then add that value to the start value to form time in hours and minutes format.

To ensure the 15 minutes' break, random notifications should be scheduled with a 14-minute radius (so select the number, then remove it + and - 14 minutes). The 14-minute radius is so selected to enable an exclusive radial selection of 15 minutes.

The algorithm will also check to see if the amount of notifications requested can be schedule (so 2 notifications for a 30 mins period, 4 notifications for a 60 minutes' period…)

Process and example:

a) Create an array of numbers from the lowest specified bound of minute to the highest specified bound in minutes
b) Select a random index in that array, the value of the specified number which the index points to is the value of the selected number
c) remove the 14-minute radius and the selected value. These removals must be made by value not index
d) append the random number to the low bound

Example:

Random Window: 12:00-12:45

1. array generated: 0,1,2,3….,45

2. select random index: i = 31 (30 selected)

…28,29,30,31,32…

3.remove 14 minute radius: 16-44 removed from array

4.append number to low bound 12:00 + 30 minutes = 12:30

5.repeat until all numbers selected

(2) Scheduled notifications

Definition:

Scheduled notification plays a significant role in TigerAware Android notification system, when researchers are scheduling surveys, they can also schedule scheduled notification with certain time and what day it will end.

Mechanism：

Scheduled notifications are the simplified version of random notifications. The difference is, it would not use the random selection algorithm and use the start time from the scheduled notification itself instead.

3.4.3 Suspension

Definition:

Suspension means, during suspend window, user will not receive any notification even though there is a notification scheduled at that time.

Mechanism：

First, users need to set up their own suspend window in TigerAware Android, which contains a start time and an end time, then when a new notification come in, it will compare weather the notification date is in suspend window then post that notification, which means the notification time should be after suspend window start time and before suspend window end time. Users are not supported to get any notification during the suspend window even though there is a notification scheduled at that time.

The problem is, for the situation above, the suspend window will repeat over and over again because it only compares the time but ignore the date. To make it work for one time, date need to be added from the backend and bound with time, which is the user input.

So the point of the new suspend is to only suspend 1 time. Currently suspend suspends at a specific time every day. So currently you would say I do not want to get a notification between 1PM and 3PM and you would then never receive a notification then forever. We want to have it so you do not receive the notification just between 1PM-3PM on the day you suspended (1 time).

The following is how the improved suspend window work for only one time suspension.

Algorithms:

(1) Adding suspend window date based on the end date

The main point of the changes is to make it such that each suspend window applies only one time before being automatically removed. The way that this should be implemented is by using the current date when the user sets the window in addition to the time. It should not need to add any additional input fields for the user to fill out; simply grab the date on the backend. Then, use that in combination with the user-input times to apply the time window to the earliest possible opportunity. There should be two cases:

a) The times entered by the user are both in the future. For example, say the time is 1pm and I set the suspend window start at 3pm and end at 5pm. In this scenario, It is supposed to do suspension on the current date between 3pm and 5pm. However, my suspend should apply *only* today from 3pm to 5pm, and not tomorrow.
b) The more complex case is if the user provides times such that only the end time is in the future. For example, consider the same scenario as above except I set the window times from 10am to 5pm. Although 10am has already passed, since 5pm is in the future we can still apply the window. This scenario should suspend from the current time until 5pm.

To achieve the scenarios above, the main point is to compare the end time with now, if end time is before now, suspend date is tomorrow otherwise suspend date is the same day when user set up suspend window.

It would add the date when user finish the suspend survey from the backend and store it separately like "start time", "end time". And when it is generating notification, it will not only compare the whether the notification time is in suspend window (previous version) but also compare the suspend date.

(2) A date detection algorithm for both start time and end time

To make suspension happen only one time, date need to be added. The difference with first algorithm is, this algorithm added two dates instead of one. In Firebase, under user/suspendWindow there is a windowStart and windowEnd that both store times. To execute suspend once and only once we used 2 new fields windowEndDate and windowStartDate. They store the date and time at which are supposed to suspend. The researchers wanted just the time input, so it still have to get only the time from the question and apply the dates to it. The applying the dates is the only tough part the rest is just reading and writing dates from the firebase and putting them in the right place.

This is how it works:when user input both start time and end time, grab date at that moment and store it as windowEndDate and windowStartDate. Then compare the start time and end time, if end time is before start time, windowEndDate equals windowEndDate plus one. Otherwise they are both today. And it should still suit the following scenarios.

(1) The times entered by the user are both in the future. For example, say the time is 1pm and I set the suspend window start at 3pm and end at 5pm. In this scenario, It is supposed to do suspension

on the current date between 3pm and 5pm. However, my suspend should apply *only* today from 3pm to 5pm, and not tomorrow.

(2) The more complex case is if the user provides times such that only the end time is in the future. For example, consider the same scenario as above except I set the window times from 10am to 5pm. Although 10am has already passed, since 5pm is in the future we can still apply the window. This scenario should suspend from the current time until 5pm.

In the performance chapter we will compare both algorithms in more detail for more scenarios.

4. SYSTEM PERFORMANCE EVALUATION

System performance is evaluated in the following three aspects: question types, notification and suspension.

Table 5 The list of functions and algorithms about System performance

Functions	Requirements	methods	results
Questions	Dynamic and custom questions	Dynamic and custom questions Vs original ReseachStack	Dynamic and custom questions design is better than just using ResearchStack and its fixed question types because it has more flexibility
Notifications	Schedule notifications with 15 minutes break	(1) the latest start time algorithm (2) the conditional random selection algorithm	We selected the conditional random selection algorithm which is more compatible with TigerAware dashboard design(strictly 15 minutes break)
Suspension	Schedule one time suspension	(1) date detection algorithm for both start time and end time (2)Adding suspend window date based on the end date	A Intelligent date detection algorithm for both start time and end time is better because it can avoid suspend over midnight problem

4.1 Different framework for question types

For question types, the improved tigeraware Android system has more flexibility than original researchStack. The original TigerAware android directly ResearchStack BackBone and its question types and countered a few problems. Such as slider bar question type is not supported in ResearchStack BackBone and we cannot modify questions. However, it is supported in ResearchStack Skin, then we download both ResearchStack Backbone and ResearchStack Skin to local and modify them to have more flexibility. In this way, the questions can dynamically change based on question design from the dashboard side. We also added custom questions such as BacTrack Alcohol Sensor question.

Table 6 The performance comparison along the framework changes of TigerAware

framework	Dynamic questions	Custom questions	Slider question
import Original ResearchStack Backbone and	no	no	no
import Original ResearchStack both Skin and Backbone	no	no	yes
download ResearchStack skin and backbone to local import and modify both	yes	yes	yes

4.2 Notifications performance evaluation

For notifications, there are two algorithms for Scheduling notifications with 15 minutes break. The following is the table about the performance about them.

Table 7 The performance comparison about different algorithms for notifications

algorithms	Advantages	Disadvantages
the latest start time algorithm	more efficient in its use of memory	Less random: after first notification is scheduled, the second notification cannot be scheduled before first notification.
the conditional random selection algorithm	More random:after first notification is scheduled, the second notification still can be scheduled before first notification	Use more memory

4.3 Suspension performance evaluation

For suspension, there are two algorithms to design a suspend window. The following is the table about the performance about the two algorithms.

Table 8 The performance comparison about different algorithms for suspension

algorithms	Advantages	Disadvantages
date detection algorithm for both start time and end time	Solve any suspend problem	Use More memory
Adding suspend window date based on the end date	Less memory Easy to achieve	It cannot solve the suspend over midnight problem

5. CONCLUSION AND FUTURE WORK

According to ResearchKit, a similar SDK called ResearchStack is used to develop a functionally equal application supported by Android, including kinds of questions, active tasks, suspend window and notifications.

For kinds of questions, TigerAware has experienced three stages: directly import ResearchStack Backbone, import ResearchStack Backbone and Skin, and download ResearchStack Backbone and skin then modify. The first stage can only support a few questions while the second can support more, such as slider questions. The final stage works the best because both ResearchStack BackBone and Skin are modified, and custom question type is created to support a variety of question types and active tasks for study use.

For random notifications, two conditional random selection algorithm are proposed to handle notifications with more flexibility, they are the latest start time algorithm, the conditional random selection algorithm. Eventually we chose the conditional random selection algorithm because it is more random based on the time sequence.

For suspend window, one time suspension is necessary to avoid repeating issue, and there are two algorithms to handle suspend window in different situations, such as suspend over midnight. One is based on end time and adding one date, another is to detect both start date and end date and the second algorithm is finally implemented because it has more compatibility.

In conclusion, by modifying both ResearchStack BackBone and Skin and adding random notification with 15 minutes break and suspension, the improved TigerAware Android System has the ability to adapt the existing functions on iOS into android with the same functionality, and develop the

new features together with TigerAware System to help some research study specifics, and all of these features and functions developed are marked with the mature development of TigerAware Android version 2.0.

In the future, as the changing of the database structure, the way to read the surveys from the Firebase need to be changed and the era of TigerAware Android version 3.0 is coming!

Table 9 The list of my contributions

	My contributions
Survey Questions	Multiple choice question, date and time question, picture in text slide question, time of day question, scale question, continuous scale question and so on
Notifications	Random notification in fifteen minutes break
Suspension	One time suspension
Others	UI improvements: text color, icon, bar color. Testing: debug and so on

6. REFERENCES

[1] Pew Research Center, "Mobile Fact Sheet," Jan 2018. [Online]. Available: http://www.pewinternet.org/fact-sheet/mobile/. [Accessed 2018 February 2018].

[2] Jayanth Kanugo, “TigerAware Dashboard: An Improved Survey Generation and Response Visualization Dashboard,” May 2018. [Online]. Avaliable: http://dslsrv1.rnet.missouri.edu/~shangy/ [accessed 2019 april 2019]

[3] William Morrison; Luke Guerdan; Jayanth Kanugo; Timothy Trull; Yi Shang, “TigerAware: An Innovative Mobile Survey and Sensor Data Collection and Analytics System,” June 2018 [Online]. in 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), 2018

[4] Jason Long. “ResearchStack,” [Online]. available: http://researchstack.org [accessed 2019 april 2019]

[5] ResearchStack “backbone documentation,” [online]. Avaliable: “http://researchstack.org/documentation/backbone/ [accessed 2019 april 2019]

[6] ResearchStack “backbone documentation [online]. Avaliable: “http://researchstack.org/documentation/skin/” [accessed 2019 april 2019]

[7] ResearchKit “ResearchKit” [online]. Avaliable: “http://researchkit.org/” [accessed 2019 april 2019]

[8] ResearchKit “Active Tasks” [online]. Avaliable: “http://researchkit.org/docs/docs/ActiveTasks/ActiveTasks.html” [accessed 2019 april 2019]