CS428 Final Documentation

Diabetes Tracker

Project Description
Process
Requirements & Specifications
    Yuchen and Wei :
    Mike and Tommy:
    Qi and Yiming:
    Yiyu and Yuelong:
Architechure & Design
    Framework
        Firebase:
        Espresso:
    Architecture and Design
        Mike & Tommy:
        Qi & Yiming:
            Tutorial Page
            Login System
            Step Counter
            Calories Tracking System
        Yuchen & Wei :
            Data Collection
            User inputs indicators and view as chart
            Diagnositic user's health condition
            User set his/her monitor plan and reminder
        Yiyu & Yuelong:
Reflections and Lessons Learned
    Yiyu:
    Yuelong:
    Michael:
    Qi:
    Yiming:
    Yuchen:
    Wei:
    Tommy:

1. Project Description

Our goal was to try to design an Android app which acts as a friendly alarm, record keeper and advisor for people who are suffering from diabetes. This mobile app keeps track of users' daily stats(i.e. the food they eat, the total amount of calories they consume, and the number of steps they take). If the user has a bad lifestyle, the app will alert the user and give some proper advice. Also, this app will provide knowledge and an interface so that patients(clients) have the "ability" to manage a healthy life. Last but not least, it will keep track of the data related to health indicators (blood sugar, blood pressure and weight), and give useful suggestions.

Because it is an android app, we used android studio as our IDE. Java was our main programming language. We used git as our version control tool, and for the database, we made use of Firebase.

2. Process

We used eXtreme ProgrammingXP for our software development this semester, also known as XP. XP has several advantages including fast planning, short release cycles, frequent testing and code refactoring. We began our initial stage of XP by first creating user stories to pinpoint important functionalities and allocate time units for each story. Afterwards, we finalized the weekly iterations and coding pairs for the changes and releases.

As much of the project was a learning process, the team experienced difficulties following the process test-driven development, meaning that tests were often written after the implementation phase. Also, a limitation of Android Studio's memory management disallowed us from running all of the test classes at once as it would run out of memory. This was resolved by running each test class individually.

We used git to allow for collaborative development between all of the pairs. Each pair had their own repository branch to work with and was merged by our team leaders Qian and Wei. Since the limitation of git needed manual merging with the same file but different versions, we minimized the tasks by having the group leaders to merge all at once instead of each pair members merging on their own.

3. Requirements & Specifications

Yuchen and Wei：

In this project, our pair focused on the diagnosis part and user indicator part. First, we collected the medical data from medical center, doctors and online papers. These data would be used as a criteria for diagnosis. Next, we developed an diagnosis algorithm for user to check their health condition after inputting his/her personal information such as waistline, age and gender. In addition, we wanted user to be able to see useful suggestions based on the diagnosis result. Then, we wanted the user to input indicators such as weight, blood sugar and blood pressure. User would be able to keep track of their indicator records as chart. Last but not least, we wanted to help user set up the monitor plan to check the user's indicator level. For example, if the user had a blood pressure level of 140.0 mmHg over 3 days, we would set up an alert window to let user know he/her was in a bad health condition and should follow the suggestions genereated by the diagnosis score.

Mike and Tommy:

Our first goal for this project was to find an easy to use external database and set up a plan for how data should be stored hierarchically in that database. Next, we wanted a user profile page and another page to allow the user to edit his or her personal information. Then, we wanted the user to be able to enter optional information for more advanced diagnostics. Along with that optional information we decided that, when registering, the user should have to input basic information that is required for the most basic diagnostics. Finally, we wanted to improve the user experience by restricting the types of information entered - such as using radio buttons for booleans and a number pad for integer and decimal types.

Qi and Yiming:

we implemented the login system, calories tracking system, step counter and tutorial page. Users can register, login, and reset their passwords through the login system which uses Firebase's authentication. The calories tracking system allows users to log their daily calories and sugar consumed to the app. And they can view a list food items they logged and a chart indicating their calories consumption history. Next, users have a step counter once they enter the “Exercise Mode”. The app will record how many steps users finished during that exercise session. Finally, a tutorial page is shown when the app is first installed on the device. It succintly introduces the main features of the app with tinted and transparent overlays.

Yiyu and Yuelong:

Our first goal was to create an expandable list to display diagnosis result and suggested diets when our app finished the diagnosis. Next we wanted to create a suggested diet page to provide users more information of suggested diets including a list of food, its pictures, glycemic index and calories. Then we wanted to link the diet page with two more pages that users were able to choose a set of suggested food and save them. Also users were allowed to retrieve all food list they saved sorted by the date. If users did not save any food list, users will see nothing when they check the saved food list.

4. Architechure & Design

4.1. Framework

Firebase:

We chose the Firebase framework as our database management for its relative ease of use and well documented API. This allowed our group to easily input and manage information for the users using simple Java methods instead of complicated SQL commands. Firebase has been used by many other Android apps in the past and this allowed us to find help when stuck on problems with great ease. The Firebase database is interacted with when changing important information but when not changing anything, a local copy is made to reduce the internet connectivity requirements of the app.

Espresso:

Android Espresso was the ideal framework for us to work on our advanced GUI tests. The Espresso framework imitates and automates the gestures via code implementations. As testers, we can set up a distinct testing flow and individualized testing parameters for each test we designed without worrying about intermediate affects on each other becasue each test will be tested separately. Our group implemented Espresso testing in most of our activities and views. By using Espresso, we eliminated redundant information entry and customized each test to do their respective goals without creating numerous dependencies.

4.2. Architecture and Design

Overall, our application included several major components: the user profile, food consumption log, health Indicator log, diagnosis and suggestions, food list and meal plan, monitor plan, and notification system.

Mike & Tommy:

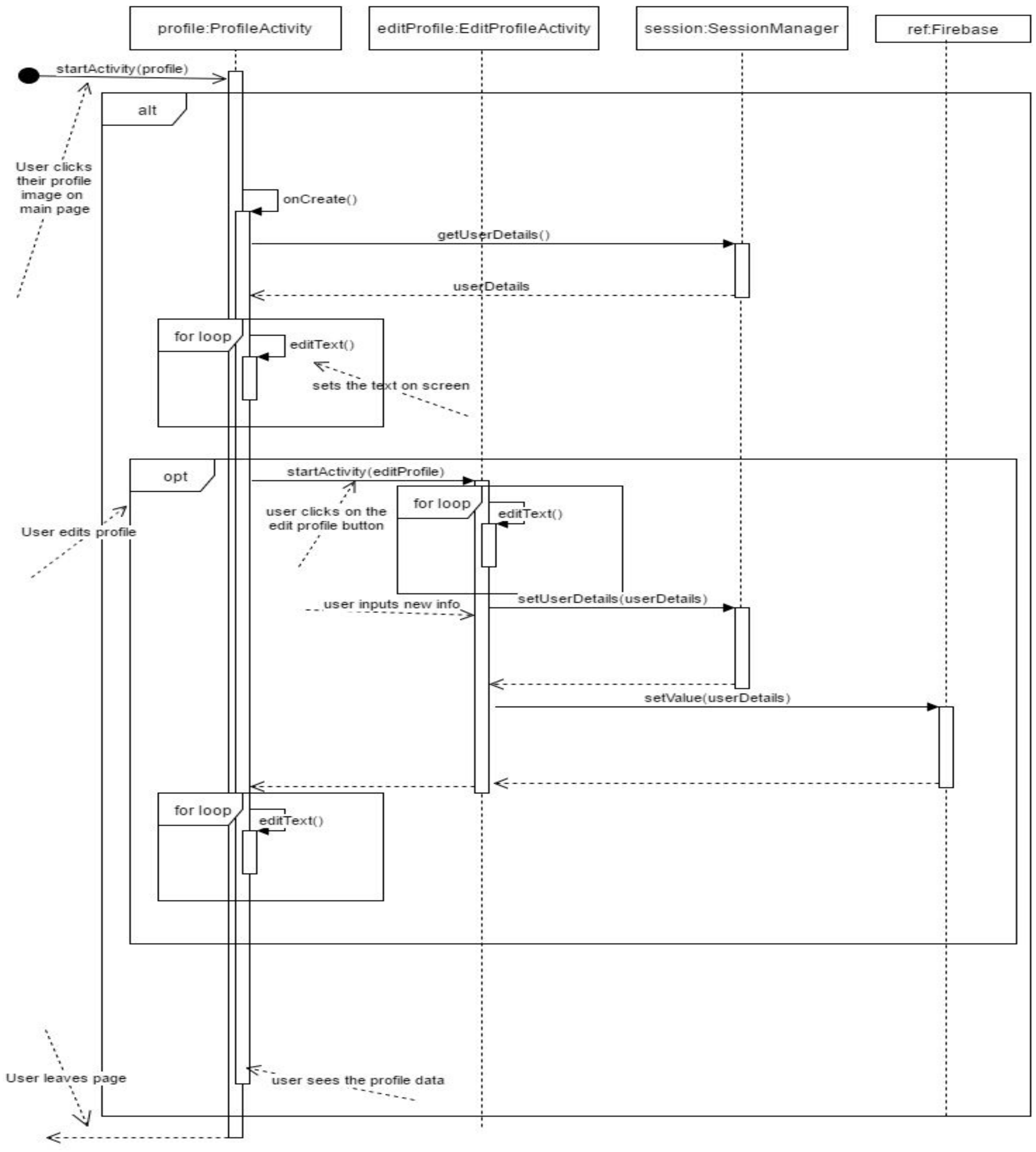

Fig. 1. Mike and Tommy’s Sequence Diagram

In order to accomplish our first goal of integrating the Firebase database into our app, we establish a connection with the database when logging in or changing information. This is done by creating an instance of the Firebase class which acts the communicator between the app and the database. Firebase made it easier to access and change information than classic SQL because everything is interacted with in the JSON format. The Firebase class has methods that may be called to allow us to access “children” which are JSONs contained within the main JSON, change the value stored in a child, and read the value stored in a child. To reduce the internet connectivity requirements of the app, we create a local copy stored in the `SessionManager` class for reading in the app. In doing so, we no longer need access to the database everytime we need access to the information for the user. It is only upon logging in or making changes to the information - which must be mirrored in the database - that we must have a connection. The `SessionManager` is then accessed by many of the other activities in the app to retrieve the information, such as: `ProfileActivity`, `EditProfileActivity`, and `RegistrationActivity`.

For the user profile, we created a page that is linked to from the home activity, `MainActivity`. By tapping on the logo - a plus sign - the user is brought to the `ProfileActivity`. On this screen, the user can see two columns of information, the left being the name of the piece of information and the right being the value that is associated with that user. The user information is grabbed in the form of a `HashMap<String, Object>` called `userDetails` from the `SessionManager`. Also, there is a button on the top right that leads to the `EditProfileActivity` in which the user is able to change their personal information. The `EditProfileActivity` has a slightly different layout than the `ProfileActivity`, in that, instead of two columns, it is laid out linearly to make it easier for the user to input and modify information. This the current information is grabbed from the `SessionManager` in the same way as in `ProfileActivity`. When new information is input or modified, the app will write this information to the local copy and to the Firebase database. Using our `Restriction` class, the app will check to make sure user inputs are valid, and display error messages next to incorrect fields and stop the user from submitting if all inputs are not valid. Furthermore, for boolean values, we used radio buttons to remove the possibility of invalid inputs.

In our user profile page, some of the information fields are required, while others are optional. The required fields are requested during registration, and the user cannot leave these fields blank while editing his or her profile. By adding checks to all the optional variables, if the user leaves a field blank or checks “Prefer not to answer” for booleans, these fields will be cleared from our `SessionManager` and Firebase database, then displayed as “N/A” on the user’s profile page.

Qi & Yiming:

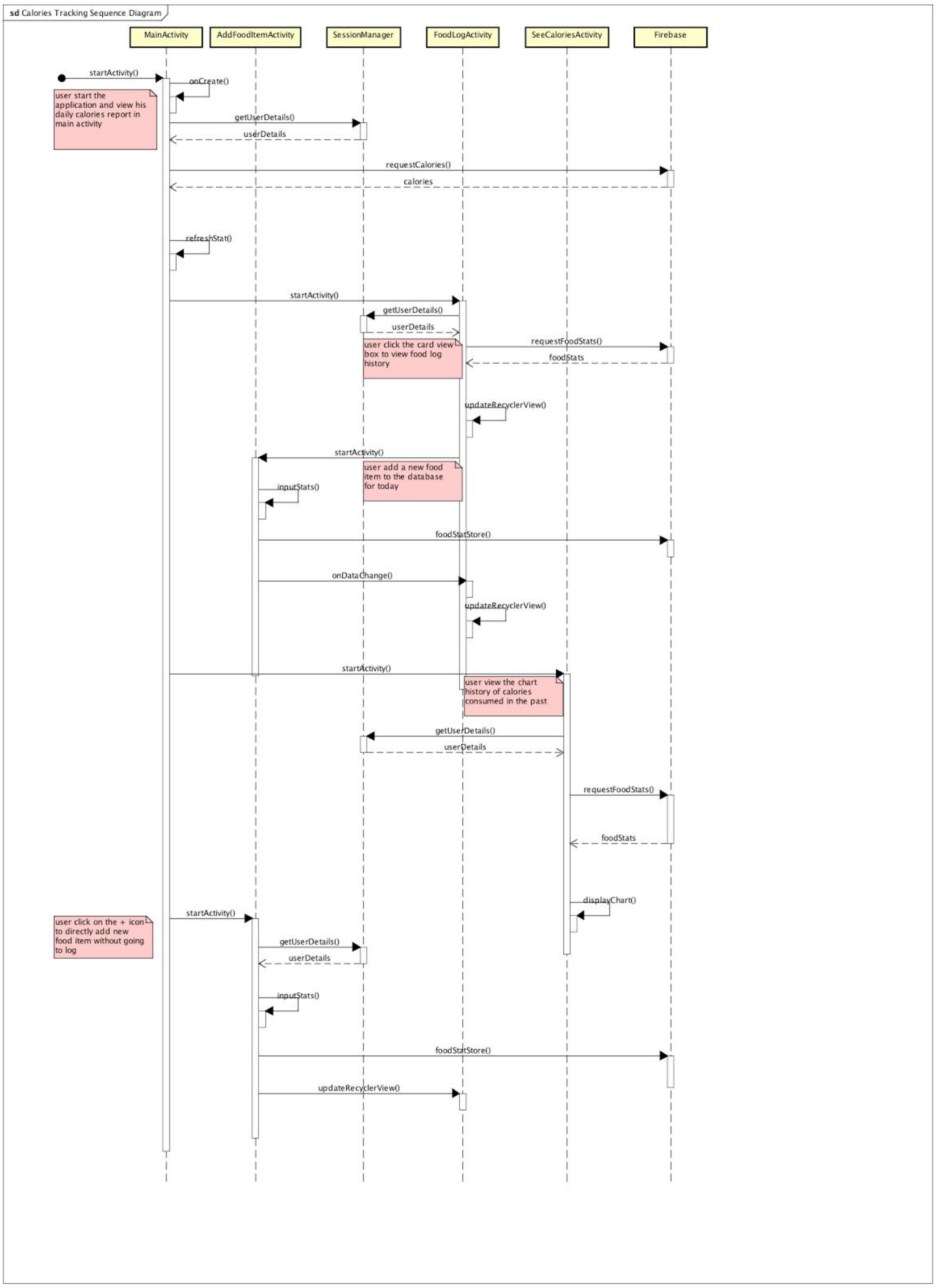

Fig. 2. Qi and Yiming’s Sequence Diagram

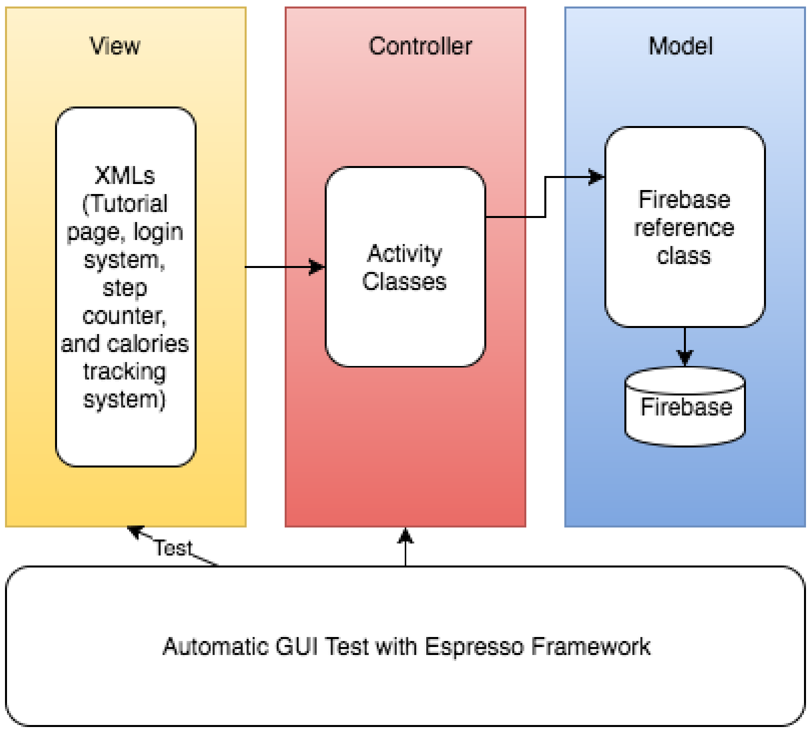
View
XMLs (Tutorial page, login system, step counter, and calories tracking system)
Controller
Activity Classes
Model
Firebase reference class
Firebase
Test
Automatic GUI Test with Espresso Framework

Fig. 3. Qi and Yiming's Top Level Architecture

Tutorial Page

The tutorial page displays instructional overlays when the user installs the app on the device for the first time and then starts the app. The instructional overlays make use of a popular library called `ShowcaseView`. The library can create an overlay to highlight and showcase a part of the screen. The lastest version of the library provides no interface for showing a series of overlays. Therefore, we created two classes to manage showcases: `ShowcaseManager` and `ShowcaseParam` in the `helpers` package.

The `ShowcaseManager` class contains an array of ShowcaseParam and an integer nextIndex. `public void showShowCaseView(ShowcaseParam showcaseParam)` is the function that shows one `ShowcaseParam`.

This function

```
public void onShowcaseViewDidHide(ShowcaseView showcaseView) {
        // When this showcaseView is dismissed, show the next
showcase
        showNext();
}
```

ensures that if the user click OK button to dismiss an overlay, the next overlay shows up.

The `ShowcaseParam` class encapsulates the following information: the activity on which the ShowcaseView is shown, the View that needs a highlight, tutorial title, tutorial content, a unique ID for a ShowcaseView, and the ShowcaseView to show. This class is used for simple parameter passing.

To show the tutorial in the `MainActivity`, we create a list of `ShowcaseParam` objects which are the tutorials to be shown. Then we call `showTutorial()` in `MainActivity`.

Login System

The Login system utilizes Firebase's Authentication. To connect to Firebase, we establish a Firebase reference using `new Firebase("OurAppFirebaseURL").`

The Login system consists of `LoginActivity`, `RegistrationActivity`, and `ForgetPasswordActivity`. After the splash screen is shown, `LoginActivity` is started. It works as a second entry point of the application. Inside `LoginActivity`, we use `session.isLoggedIn()` to check user status. We direct the user to home screen is he is logged in. Otherwise, the user is prompted to sign in.

`RegistrationActivity` required the user to provide email, password, age, BMI, blood pressure, gender, and family history.

If a user forgets his password, he can reset his password on Forget Password page. The page is controlled by `ForgetPasswordActivity`. Firebase provides a function `resetPassword`. Once the users submit a password reset request, they will receive an email from Firebase with a temporary password.

Step Counter

We used a step counter sensor available for Android 4.4+. Unsupported devices will not have the step counter shown when the users go to the step counter page. To have a clean start on counting, we used `SharedPreferences` to store last recorded steps.

The step counter records, displays and refreshes each time our user enters the `StepCounterActivity`. We've utilized android `SensorEventListener` to track steps each time `StepCounterActivity` was activated.

Calories Tracking System

The calories tracking system was designed to update in a real time scenario with a firebase query each time when user accesses the food log page or calories intake page. On the main activity page, the user can have the option to add, view calories chart or view food log via three buttons in our generated card view. If the user choose to add a new food, by clicking on the add button, user will be directed to the `AddFoodItemActivity.`Each food item will contain three fields: food name, calories and sugar in their respective units. Upon a successful information retrieval from `SessionManager,` user email and other information will be used to query our firebase for storage. When the food was successfully incorporated into our database, user will be directed to our `FoodLogActivity.`In FoodLogActivity, the database will be queried again for newest updates as well as a history of all the food consumptions from the first usage of our app. If the user wishes to add food again while in `FoodLogActivity`, he or she can use the entry point at the bottom right fab icon;they will be redirected to the `AddFoodItemActivity`.

For calories consumption history, the user can only enter via the entry point in our `MainActivity` by cliking on the icon in our generated card view. The `SeeCaloriesActivity` will query our firebase using the information stored in `SessionManager` and display the history of calories consumptions per each day. Since this activity was read-only, user will only be able to scroll left or right to view the data from the database.

Yuchen & Wei：

Data Collection

In this user story, we collected data from medical center, papers, and doctors

We selected low-GI, medium-GI and high-GI food for meal plan's future usage. We collected several most updated diagram/mechanism for evaluating a agent's risk of getting diabetes.

We also looked into multiple complications and all kinds of inducements of diabetes such as BMI, waistline, age, family history, sedentary lifestyle. Indicators like blood sugar, blood pressure were the most critical data and we needed to monitor them in our app. We also used these data in a dynamic way. For example, a sedentary job plus less or equal then 60 minutes exercise per day, or less then 30 minutes erercise per day was considered as sedentary lifestyle.

User inputs indicators and view as chart

In this user story, we allowed user to input indicators. There were three indicators for user to input: weight, blood sugar, blood pressure. User would be able to view his/her input history in a separate page and would be able to update his indicator data for the current date. In addiction, the user would be able to see the history data as a chart. There were three separate charts corresponding to the type of indicator.

We first created an indicator class to store an indicator instance which contains weight, blood sugar and blood pressure for each user. In the input process we checked invalid inputs. For instance, a person who had a weight of 630kg was considered impossible. These data would be sent to the firebase and stored. Next, we created an indicatorlog class which contained an indicator object and the date when the indiactors were input. With these two helper class, we created an activity called indicatorlog activity. After the user added the indicator data or the user clicked the data card in the main page, we would nagivate the user to see his/her history data. In this process, we first created an recyclerview to present these data. Next, we created an recycleradapter instance to store the date, weight, blood sugar and blood pressure we got from the firebase. Recycleradapter would get all the data in the firebase. Thus, user would be able to see the data.

For the chart part, we got the data in the same form. In addition, we created three line charts for corresponding indicator. Also we created three buttons for user to select which chart they wanted to see.

This user story would be connected with setmonitorplanActivity. We would allow user to set the monitor plan to check their indicators. For example, if the user had over 140 mmhg blood sugar for 3 days, we would give them an alert to tell the user to be aware of the problem.

```
 public Class Indicator() {
   double bloodSugar, bloodPressure, weight;
}
public Class IndicatorLog() {
   Indicator indicator
   Date date
}
public static class MessageViewHolder extends RecyclerView.ViewHolder {
   public MessageViewHolder(View v) {}
}
FirebaseRecyclerAdapter<IndicatorItemLog, MessageViewHolder> adapter =
new FirebaseRecyclerAdapter<IndicatorItemLog, MessageViewHolder>
(IndicatorItemLog.class, R.layout.recycler_list_indicator_section,
               MessageViewHolder.class,mRef) {
           @Override
           protected void populateViewHolder(MessageViewHolder
messageViewHolder, IndicatorItemLog d, int i) {}
};
mRecyclerView.setAdapter(adapter);


public Class SeeIndicatorActivity(){ // see the indicator chart }
public Class IndicatorLogActivity(){ // see the indicator history data }
public Class AddIndicatorActivity(){ // add indicator }
```

Diagnositic user’s health condition

In this user story, we allowed user to see their health condition after inputing their personal information. User would be able to know what kind of health condition that he/her was in and would receive suggestions about how to start a good livestyle.

}

We first created a user class which would make a user instance and store all the personal information which the user inputed in the profile page. Based on these information, we implemented an algorithm to calcualate the user's score, which would be the result of diagnosis. We created calculate function for data we got. The calcluation was based on the data chart we collected in the previous user story "Data collection". After we got all these separate score, we added them together and got our final score for that user. Based on the score, we gave them different suggestions. For instance, if the user got a score of 26, we would consider the user was healthy. However, if the user had a score of 40, we would suggest the user to make a diet and do more exercise. These functions were all implemented in the user class.

This design connected the user proflie and user diagnositic. Also, with diagnosis information, we would be able to develop the function that user could take a look at different kinds of food and select the food they want as a diet meal.

```
public Class User() {

   user info…

   public int calcScore() { return score;}

   public String generateSuggestion() { return suggestion; }

}

public Class DiagnosisActivity(){

      // generate score for user and give user suggestions

}
```

User set his/her monitor plan and reminder

In this user story we need to help user monitor their health indicators. First, user could set which indicator they want to see on the orange card-view on the front page. Also, they could get warning is long as the indicator they choose (can be both blood sugar and blood pressure) is out of a normal boundary for several days (user can also pick this length). We get the monitor setting by DataSnapshot and use the variable "toDisplay" to decide which indicator to be shown on the card view. Also, We build two classes Monitor and MonitorPressure to keep track of the warning messages, which contain detachWarning() methods to decide whether to generate warning or not.

```
public class MonitorPressure {

   public int count=3;

   public Queue<Double> bloodPressureQueue;

   public boolean warning;

}
```

For the reminder, if the user hasn't input any health indicator today, we would leave a message to him as a android notification. We use notificationCompatBuilder to create the notification. We set its autoCancel attribute to be true so that it would dismiss after user click on it. It guides user to the AddIndicatorActivity when clicking, so that user can quickly add their health indicator. Then we try to get the blood pressure data from that date, using Datasnapshot. If we do find a invalid one, we remove it by:

```
mNotificationManager.cancel(id);
```

setbacks and solving:

About initial setting:

we give a initial (default) setting: {period:3, display: blood sugar, monitor{blood sugar}}

In test cases, we could change it to: {period:1, display: weight monitor{blood sugar,blood pressure}}

Asynchronous problem with firebase:

do that in a nested listener functions:

```
someRef.addValueEventListener(new ValueEventListener() {

   @Override

   public void onDataChange(DataSnapshot dataSnapshot) {

       someRef.addValueEventListener(new ValueEventListener() {

       @Override

      public void onDataChange(DataSnapshot dataSnapshot) {

      }

}}}
```

Multi-layer problem of reminder:

```
if (alertDialog!=null) { alertDialog.dismiss(); }
```

within the internal “onDataChange” function so that the dialog would not show up for multiple times.

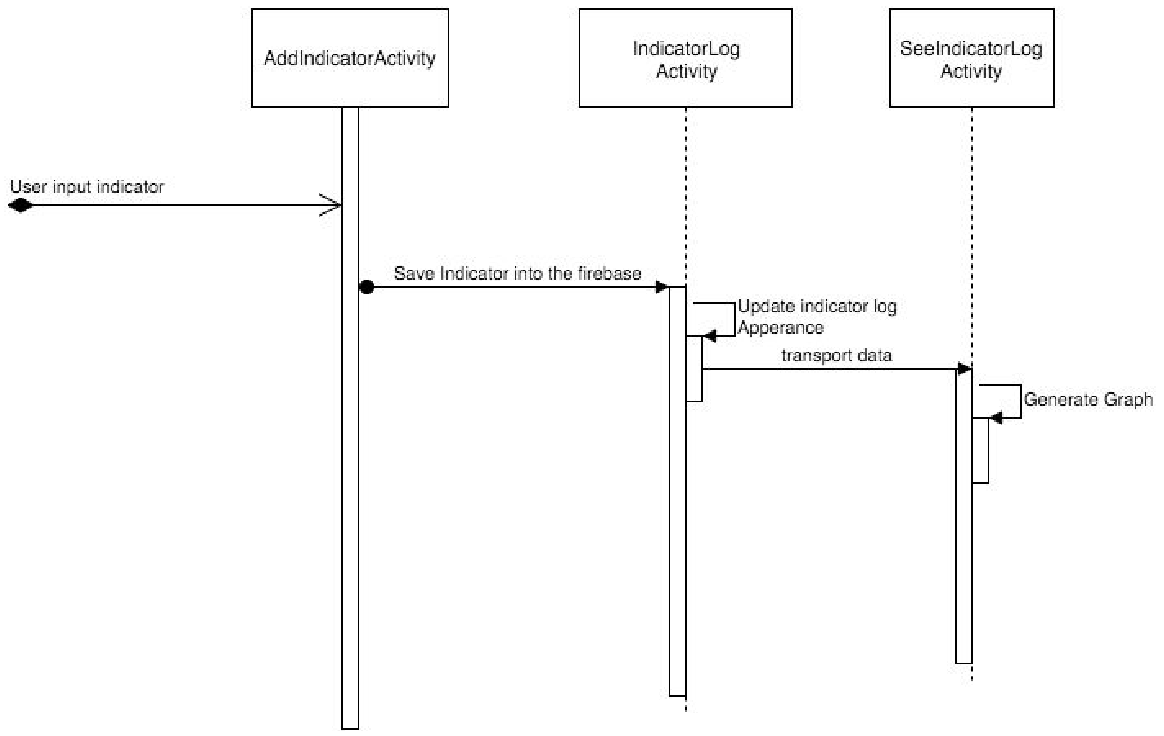

Fig. 4. Yuchen and Wei's Sequence Diagram 1

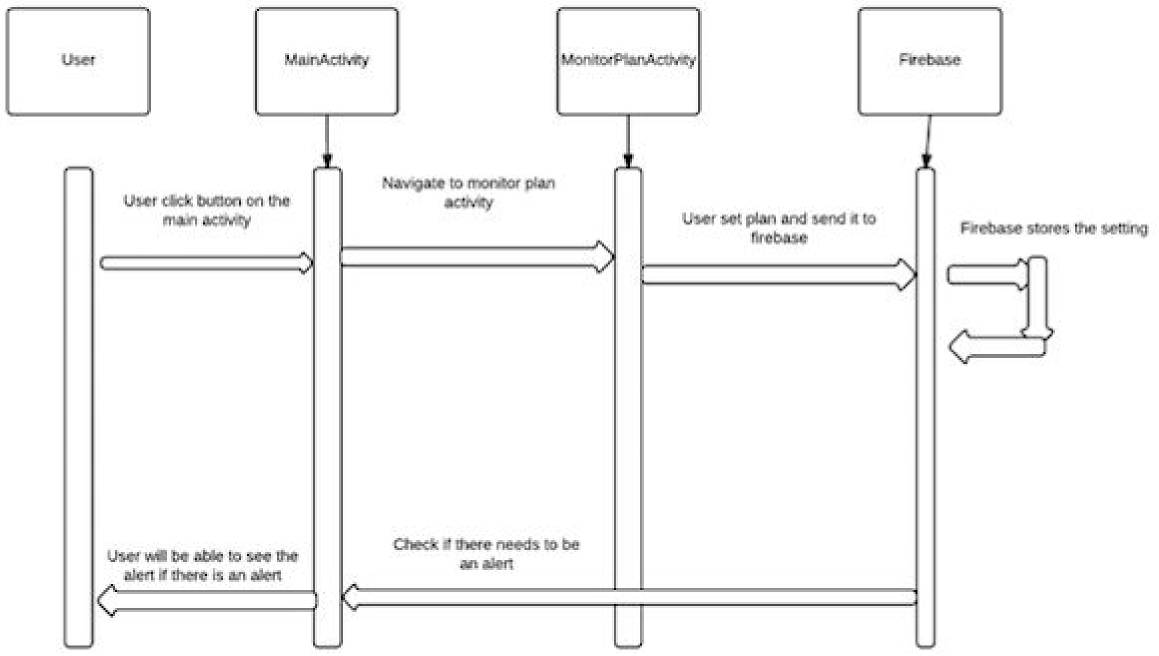
User
MainActivity
MonitorPlanActivity
Firebase
User click button on the main activity
Navigate to monitor plan activity
User set plan and send it to firebase
Firebase stores the setting
Check if there needs to be an alert
User will be able to see the alert if there is an alert

Fig. 5. Yuchen and Wei's Sequence Diagram 2

Yiyu & Yuelong:

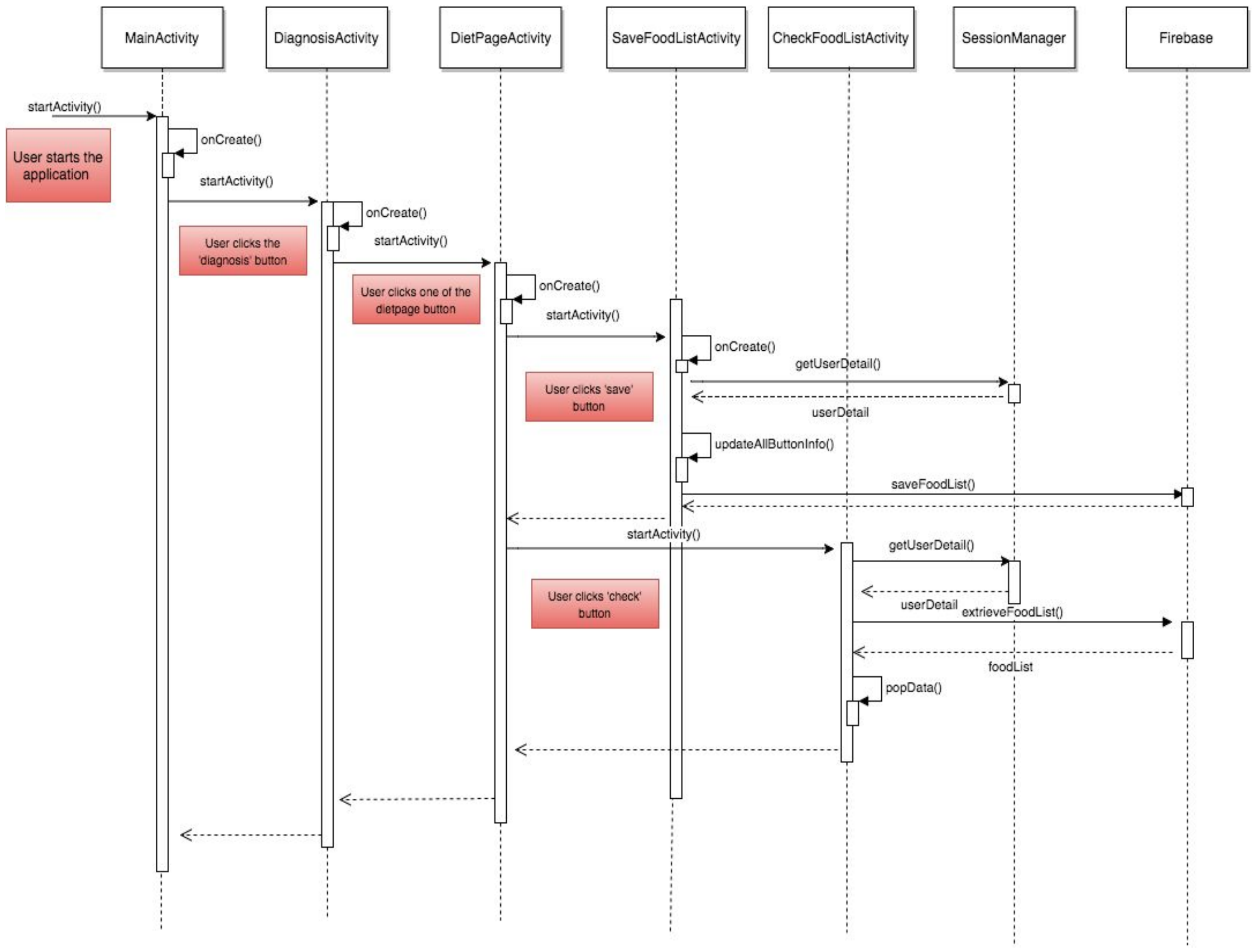

Fig. 6. Yiyu and Yuelong's Sequence Diagram

In order to accomplish our first goal of preparing an expandalbe list to display the diagnosis results and suggested diets, we created an adapter class `ExpandableListAdapter` to render the expandable list view. This `ExpandableListAdapter` class was extended from `BaseExpandableListAdapter`. `ExpandableListAdapter` class overrided most of its parent's functions and had three new private member variables, activity context, string list of headers, and list of string list of items corresponding to each header. When `DiagnosisActivity` had managed to prepare list of headers and lists of item list, it would construct an object of `ExpandableListAdapter` with its context, and the prepared data. Then `ExpandableListAdapter` would help render the prepared data as an expandable list. Thus in order to display the diagnosis results and suggested diets, we just created a string list to hold header information, and a list of string list to hold item information under each header.

The second goal was to create a suggested diet page to provide users more detailed information about the suggested diet. We had three suggested diet pages `DietLowGIActivity, DietMediumGIActivity, DietHighGIActivity,` corresponding to three cases of low, medium, high glycemic indices. In each suggested diet page, users were able to click “save” and “check” buttons to go to another two pages. Also, the suggested diet page displayed the food list as an expadable list. Each header would show the name of a kind of food while each item in the list under that header would show its picture, glycemic index, and calories. We used `ExpandableListAdapterWithImages`, a modified version of `ExpandableListAdapter` to render the food list so that it could render images. `ExpandableListAdapterWithImages` is done by adding one more member variable, a list of image objects.

The third goal was to allow users to save a customized list of food from a corresponding suggested diet page. In doing so, we created three pages, `SaveLowGIDietActivity, SaveMediumGIDietActivity, SaveHighGIDietActivity,` linked from “save” button of corresponding diet page. In each page to save the food list, we used the design of Yes/No radio buttons for each food so that only one of Yes/No button will be clicked. In order to save the food list into firebase, we first retrieved user information by calling `userDetails` from `SessionManager.` Then we used the users’ email to connect to firebase. Next we defined a listener for “submit” button so that if users clicked the button, each Yes/No button would be checked to generate a string that contains the foods chosen. Finally we stored the string that holds the information of selected food into firebase by a helper function `saveFoodList` in `FoodList` class. After storing the food list into firebase, users would be directed to the suggested diet page again.

The last goal was to allow users to check all food lists they saved in history. If users had not saved one, they would see empty page. We created `checkSavedDietActivity` class to render the page. It first retrieved user information by calling `userDetails` from `SessionManager.` Then it connected to firbase and got firebase object by users’ email. Finally it would wait for an event listener that retrieved the data from firbase. After that, it prepared the retrieved data to construct an `ExpandableListAdapter` object and used `ExpandableListAdapter` to render the food list.

5. Reflections and Lessons Learned

Yiyu:

It was the first time that I learned and built an Android application. It was instructive and interesting than I expected. Using Java in development made me feel great, but testing in Android was a little frustrating to me. Overall I was happy about learning a lot of knowledge about mobile development. About the development process, each team member contributed a lot to the project. However, it would be better if we could have better design on user stories so that they are interesting and meaningful to implement. I was happy about the experience of this project.

Yuelong:

This is the first time I work on Android. It is interesting to learn how the activities interact with each other through the Intent object and the manifest file, save information across activities with the SharedPreference object, and how to define the activity layouts through the layout xml files. I also learned to use firebase to save data across each different log in sessions. This is a valuable experience to code in a new platform and learn new frameworks and libraries. I also learned a lot about code commenting and refactoring because they could make it a lot easier to share your code with others. It has been a great experience overall to work with my team.

Michael:

It was a unique experience to work on a project that I was not particularly interested in. However, it was interesting to learn about Android development and frameworks. It was valuable to also learn about new strategies for code such as the Interface in Java; it was necessary to learn this because Java does not support passing in functions as parameters as other languages do. Overall I am proud of the work that I contributed to this project. The team as a whole seemed to have troubles sticking exactly to the process of eXtreme Programming, particularly writing tests before writing the implementation, as was the case in CS427, but I believe the end product to be satisfactory considering the sheer amount of learning that preceded each step of this project.

Qi:

Through this project, I learned android development for the first time. I learned many libraries, frameworks, and Google’s material design. Getting myself familiar with so many libraries and design guidelines is very beneficial for me whenever I want to start developing another android application. Experience I gained in CS427 was very helpful for this project. We were following the process of Extreme Programming. We did pair programming. The process was much more organized than it was in CS427. However, there are still a lot to be improved. Testing is in our schedule but it was not done quite well. We always had unexpected problems with unstable testing. Some user stories seem unconnected so that the app might not be what it was supposed to be. Overall, the experience was pleasant and the app conforms to the requirements.

Yiming:

This project helped me to further understand the importance of programming practices such as XP when developing application that demands team work and coordination. Throughout our project we've had many problems that required absolute coordination to solve. Luckily, with the practice of XP, we communicated within the team effectively with comprehensive documentation, in depth communication and coordinated planning for each iterations. We designed the user stories, executed and revised with speed so that we can move on. For the problems that occurred during each week, we always revised them with solutions and refactoring to make sure our code stays coherent and simple to read. In conclusion, the experience using XP was definitely rewarding. I learned to code in a group more effectively and swiftly.

Yuchen:

I have learned a lot from this project. The very first techinical stuff I experienced was android development. At first, Wei and I didn't have experience on android development. However, in order to push ourselves to learn new skills, we wanted to start an android project. In these early iterations, I spent most of time learning android development. Although I didn't make large progress in our project, I found myself familiar with basic android development. With my solid Java programming skills, I and Wei implemented several important features such as Diagnosis, add indicators and monitor indicator level. In addition to android development, I also learned a lot about refactoring, writing advanced test and team working skills which seemed to be the most important. Overall, I was very happy to work with my teammates and I charished this experience of working on android development.

Wei:

After doing this project, I learned a great deal about leadership and technical stuffs. I got more familiar with the android develop. Equipped with fluent java skill, I learned a lot about multiple java features and the utility/usage of firebase. A most setback I faced was the asynchronous problem related to “get” option from our database. Fortunately, I finally get hint from my node.js coding and solve that problem. Besides problem solving, a great thing I learned is refactoring and writing advanced tests. I learned the importance of these good coding styles and habits. As one of the team leader, I also learn a lot about teamwork and leadership. To conclude, We build the product we desired and I am happy to work with my talented teammates.

Tommy:

This project was a good learning experience, and was my first time doing Android programming. It was interesting learning how to build an app and experiencing how to build different layouts and activities. I also learned a lot about dealing with GUI testing, learning how to solve various problems along the way. It was also a good experience working with a large team again, and following the XP process was quite helpful in our organization and development. Overall, it was a good experience working together with the team and learning a lot along the way.