Group 1: Data Aggregator: Fitness Tracker

by

Andrew Lincoln, Andrew_Lincoln@student.uml.edu
Geoffrey Klapproth, Geoffrey_Klapproth@student.uml.edu
Stephen Demeule, Stephen_Demeule@student.uml.edu
Harrison Kelly, Harrison_Kelly@student.uml.edu
David Lordan, David_Lordan@student.uml.edu
Ian McGaunn, Ian_McGaunn@student.uml.edu

Table of Contents

1. Introduction

1.1. Document Description

This document is a detailed description of the our team's Android application that is meant to track motion. The document includes descriptions of the application's purpose and intended use, as well as descriptions of our design considerations and implementation.

1.1.1. Introduction

The purpose of this project is to create an Android application that will be able to store information about a user. This information will include: the user's weight, height, sex, date of birth, and name. The name for the application is Fitness Tracker and it is in the Alpha stage. Additionally, that application will also be able to transfer data from a sensor (such as an Android watch) that will act as a step counter. Once the data is collected, it is sent to a database in the Cloud to be stored.

This project must allow the user to register and log in to the Android application. The user management system will be handled locally because of the missing interaction with the Cloud group. After logging in, the user will then be allowed to enter and edit their weight, height, sex, date of birth, and name. The user interaction with the application should flow smoothly and the user should not be stuck and confused as to how they should proceed. Due to the limitation of not being able to have any interaction with the Sensor group, the step counter will not be implemented.

This document's intended audience is anyone who is interested in learning what this application does at a lower level. It is also meant for someone who wants to build on the application so that they can get an understanding of the architecture and design decisions that were put in place when the application was developed.

This document goes into detail for explaining the Android application we developed, Fitness Tracker. The document first describes an introduction of the system overview by describing its functionality and design. It next describes the design considerations that took place when the application was developed. These include assumptions that were made when developing the application, dependencies on other frameworks and applications, general constraints that had to be taken into consideration, the goals and guidelines we had when designing the application, and the development methods we used for developing this application. The next section describes the architectural strategies that we used for developing the Android application. Finally, we describe the system architecture as a whole and the policies and tactics that were used for development along with a detailed explanation of the system design.

1.1.2. System Overview

This application was designed with the intention of taking advantage of the sensory data the Bluetooth-connected smart watch provides to an Android device. Utilizing this hardware to monitor and prompt the user to make more active, healthy choices became the primary goal and motif driving development.

As mentioned in the introduction, a simplistic and minimal approach to encouraging healthy behavior is provided through use of this application. A user is capable of creating a personal profile upon their first launch of the application, where they simply need to fill in an e-mail address which is used as the primary log-in account information, enter a password for their account, and confirm the password by entering it once more. Any subsequent log-ins will prompt the user to sign in using their profile from a log-in screen. After signing in, the user can manage their profile, which includes managing personal information for application use and registering a smart watch to sync to for collecting data.

The application is organized through a series of activities that are easily accessible from one to the other. Each screen houses buttons, selectable text, or other forms of intuitive and easily accessible options to allow smooth and flowing user interaction with the device. Again, as the application is designed with an "ease-of-use" approach, most of the data collected through the sensor and other miscellaneous data is kept hidden from the user to avoid any confusion or complicated interactivity. In other words, the user will only see what is important and relevant to what them.

2. Design Considerations

.

This section will describe and outline several of the concerns and considerations that will need to be looked at before our final design. This includes our platform dependencies, constraints, goals, and method of development.

2.1. Assumptions and Dependencies

This Application requires an Android phone running Android 5.0 or greater (API level 21) with Bluetooth and WiFi connectivity. It is our hope that it is simple enough to be used by anyone, but it assumes that users have had exposure to typical mobile application layouts on the Android platform. Future versions of the application will further prioritize user experience and attempt to increase the visual appeal of the interface, but its current state is prioritized to include the primary structural components of the applications before making usability optimizations.

2.2. General Constraints

The difficulties we faced in the project were from both the project itself and forces beyond the project:

- Program will be limited distribution by making it solely for Android platform.
- Time restraints made development time cut short.
- Teams general unfamiliarity with Android development made progress difficult.
- Lacking third party to test device.
- Lack of back-end support from needed sensory group made data collection impossible.
- No cloud storage space meant local machine storage only.
- Unclear end goals staggered development.
- Currently there are no interfaces to any external sensors, making our ability to implement motion detection and analysis impossible.

2.3. Goals and Guidelines

- Allows the user to easily navigate around the application. This is necessary because an application that is difficult to work with is utilized less often. If an application is downloaded that is too difficult to learn, a user will rarely spend the time becoming familiar with its functionality and will likely opt for an alternative application that is more intuitive and easier to work with.
- Design the layout of the Android application in such a way to promote speed while still staying with the common Android application development guidelines. This will allow the application to quickly respond to the user's action and be recognizable as a common Android application.
- Allow for the user to be able to add a device (Android watch) to the application seamlessly and have it pair without any issues. This will prevent users from getting frustrated if the device will not pair with the application easily.
- Allow for the user to easily edit the information that they first entered into the application upon first installing it. This will allow users to update their weight, height, sex, and mobility in case their information changes.

2.4. Development Methods

This application was primarily developed using the Scrum methodology, a type of agile software development. The application was developed over the course of two separate Scrum sprints. The results of the first sprint was the successful implementation of the user information screen. The second sprint resulted in the implementation of both the sensor management screen, and the user management screen.

No other development methods were considered as Scrum was seen by the entire team as the most appropriate approach to meet our project's goals. The Scrum method seemed to offer the most efficient use of our time and allowed for quick, flexible development. As this application is primarily based on user interaction, usability tests would be a crucial indicator of our success. The Scrum method allows us to make several iterations of our application that could theoretically each be tested for usability. The results of these tests could then be easily incorporated into future versions. Again, this makes the Scrum method a perfect means by which our team can meet its goals.

A full description of the Scrum methodology can be found at the following link:
https://www.scrum.org/

3. Architectural Strategies

Our architectural strategies were largely governed by our dependence on the Android ecosystem. Using the Android framework constrains us to using the Java interfaces to Androids native system components and the software design patterns enforced by Android's class hierarchy.

Additionally, we attempted to use sane software design practices that would allow us to later link our program to the other subsystems that are necessary for its operation, principally some sensor hardware and a cloud system.

Our user interface strategy was to design for what the user would expect out of a mobile application for fitness tracking. Since design paradigms for this sort of program are pretty well established, especially for mobile devices, we tried our best to emulate existing solutions. Each of our main views uses a logical layout and should theoretically be intuitive and usable for established users of Android applications.

4. System Architecture

The program is to receive data from an outside sensor - the smart watch - and will then take the data and store it into a local drive. Users are able to sign into the application. This requires a user name and a password for security measures. Here they are taken to the main screen where they are capable to viewing their activities (sent in from the sensor) and enter in personal data that the sensor is not capable of recording (diet weight and other things). This data, if capable, is then sent off to cloud storage. From this main screen the user can also tell the app which sensors to follow and remove sensors that the user no longer wishes to use.

5. Policies and Tactics

Throughout the project, we had to make many decisions on developmental resources. One of the most important factors was deciding which IDE would be standard for implementation, which we opted for IntelliJ/Android Studio. This was chosen in order to try to minimize the amount of issues or other work required for importing and compiling any changes to the code before pushing to a repository hosting the approved version.

The only specific coding practice that was used was the writing of detailed code documentation. This includes descriptions of variables and their use, as well as descriptions of functions that are used and their purpose. This documentation will allow for anyone working on the project in the future to quickly learn and understand the code's organization and purpose. As this is a type of application that is best developed incrementally through several different versions, it is critical that this documentation be updated so that developers working on new versions will be aware of updated features and their implementation.

Testing the software became a primary concern during development because of two major reasons. The first was due to the lack of group members who had access to either a smart watch or an Android device. This severely limited who could test the code and commits and to at what point those who could test were available. The second was the lack of third-party testers participating. Having those outside of development provide feedback through testing the application would provide much better context and specification on what needs change or what to add in terms of usability or functionality.

Following the Android app design guidelines, the code is organized into a series of sub components that make up the viewable screens and the logic behind them. For each of the screens, or *activities*, there is an associated XML file that details how the activity looks, as well as XML files for their associated settings menus. For each of these activities the logic that manipulates them is placed in associated Java file.

Section 7 (System Installation and Execution), which is seen below, provides a detailed description of how to download and run our application, either on an emulator or on an Android device.

6. Detailed System Design

The following section will provide brief descriptions of our application's main components. As our application is currently only a simple user interface, our three components are the different views provided to the user. The three views that are described are the account management view, user information view, and the sensor management view.

For each component we have provided a brief classification description, a definition, a brief description of the component's responsibilities, design constraints, the view composition and the view's uses and interactions.

6.1 - Account Management

- **Classification**
  User Interface View.

- **Definition**
  The purpose of this component is to provide a means by which users can create new accounts or sign in to existing accounts. This is to ensure that the data being gathered is uploaded to a corresponding cloud account where it may be viewed by the user or the health care provider.

- **Responsibilities**
  As one of our requirements is to allow users to be able to create and manage accounts, we needed to provide an interface for such a functionality. This view provides that functionality, allowing users to both sign up for a new account as well as log in to an existing account.

- **Constraints**
  This view, along with all other user interface views, must be able to quickly respond to a user's actions within a reasonable amount time. Other constraints consist of proper user input, such as a valid email address and password. The interface will only accept email and password strings of a reasonable number of characters. The view must also be able to interface with a database of user names and passwords to confirm a user's credentials and to log in to the proper account. As the current version of our application does not yet have any connection to a database of account information, this constraint is not currently applicable nor testable. However, the view must be set up in such a way to allow for this functionality in future versions.

- **Composition**
  The user management interface consists of a sign-in screen and a sign-up screen. The sign-in screen is a very simple form made up of an email field and a password field.

A screenshot of the sign in view can be seen in figure 1 below:

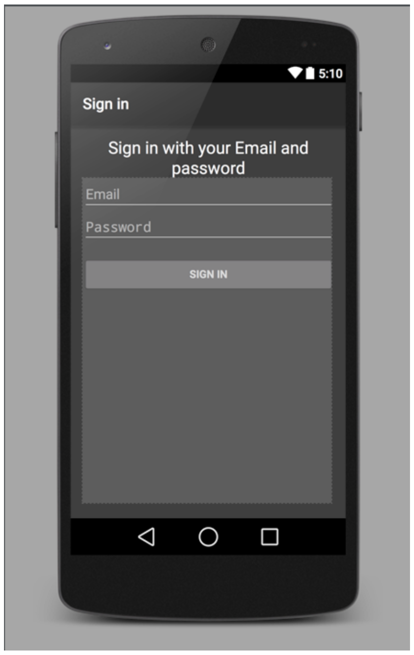


figure 1.

Similarly, the account creation view contains a very simple form with text-fields for the new user's email address, their desired password, and confirmation of their new password. A screenshot of the account creation view can be seen in figure 2 below:

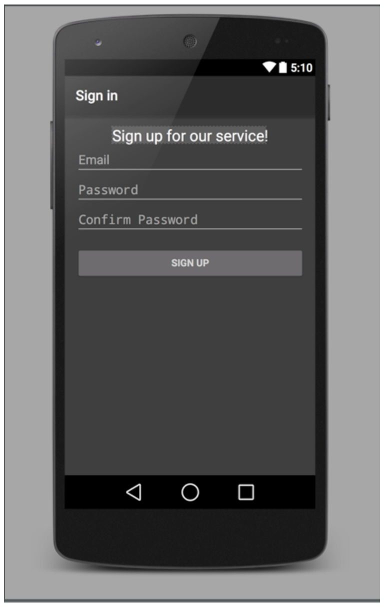


figure 2.

- **Uses/Interactions**
  This component is meant to interact with users to allow for the creation of accounts and/or logging into currently existing accounts. Presumably, this would be done in conjunction with a database of user names and passwords, which is not accessible in the current version of the application. This database would then fetch all of the user's personal information and would then route collected data to the appropriate Cloud service.

- **Resources**
  The following is a list of files that are required by this view:
  1. activity_login.xml
  2. activity_signup.xml

- **Processing**
  The following is a list of files that are used to generate this view's behavior:
  1. LoginActivity.java
  2. SignupActivity.java

6.2 - User Information

- **Classification**
  User Interface View.

- **Definition**
  This view allows a user to enter their personal medical information, such as name, date of birth, weight, height, and gender.

- **Responsibilities**
  The user information view provides a means by which users can edit and store their personal information. This information would they ideally be used in conjunction with movement analysis and would additionally provide information to their health care provider. This could also possibly be used by a cloud service to do large-scale analytics on a large section of users to find correlations between a user's activity and their physical state.

  Users can easily edit or view their personal information whenever they wish.

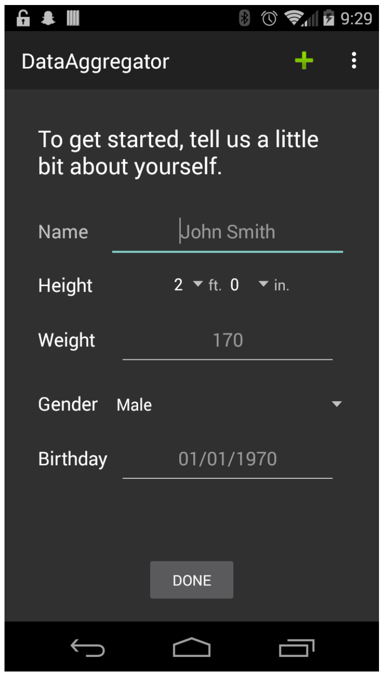
- **Constraints**
  Again, like any normal user interface component the view must respond to user actions quickly. As this view was built using standard Android practices, this should not be an issue. User information must be constrained to

appropriate data and the interface allows for this. For example, the user's height is input as a number feet and inches rather than simply a string, which could be any value, ensuring the input is a value that makes sense. The same is true for the date of birth input field as well as the weight input field.

As future versions of our application must connect to a Cloud service, the user data that is saved from this screen must stored in an appropriate data structure that may then be sent to the Cloud, or, sent to another local software subcomponent that has yet to be implemented.

- **Composition**
  Similarly to the other views, the user information screen consists of a simple form with fields for a user's name, height, weight, gender, and date of birth. The fields use appropriate Android widgets to select text, numbers, or whatever is needed for a particular field.

  A screenshot of the view can be seen in figure 3 below:



To get started, tell us a little bit about yourself.

Name	John Smith
Height	2 ft. 0 in.
Weight	170
Gender	Male
Birthday	01/01/1970

figure 3

- **Uses/Interactions**
  Ideally, the user information screen should send its current user information to a Cloud storage system where it can be viewed by the user or their health care provider. However, this current version does not have any interface to a Cloud database so this functionality is not yet implemented.

Currently all information is received by this view comes directly from the user and is stored locally on the device.

- **Resources**
  The following is a list of files that are required by this view:
  1. activity_main.xml
  2. menu_main.xml

- **Processing**
  The following is a list of files that are used to generate this view's behavior:
  1. MainActivity.java

6.3 - Sensor Management

- **Classification**
  User Interface View.

- **Definition**
  The sensor management screen will allow users to add and remove connections from their Android smart phone to various motion sensing devices.

- **Responsibilities**
  This component will be responsible for allowing the user to enter information about motion senors that may then be paired with their phone, presumably via Bluetooth. As no actual communication between our application and any sensors were ever established, this component does not have any functionality beyond simply being a UI skeleton.

- **Constraints**
  This view is currently only a UI skeleton with very little actual functionality. The current constraints are very simple and very typical UI considerations, such as quick responses to user actions and user input validation. Only the device ID and device name text fields will require input validation as the only other field is a simple set of radio button selectors.

  Were the view to interface with sensors there would need to be a way to detect nearby sensors and ensure their compatibility with our application. As our UI skeleton is based on there only being three possible sensors to add, any sensor detected would need to be one of the available three.

- **Composition**

This ‘add sensor’ view consists of a simple form, allowing a user to enter a device identifier, a device name, and a device type. The currently selectable device types are a Moto 360, an ASUS ZenWatch, or an LG G Watch R.

A screenshot of the ‘add sensor’ view can be seen in figure 4 below:

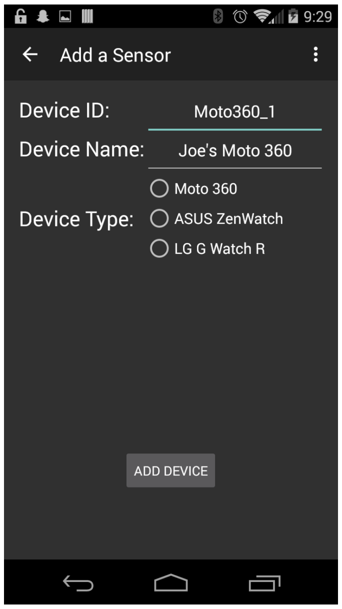


figure 4

The ‘remove sensor’ screen consists of a simple drop down menu that displays all currently added sensors. The user may then select the appropriate sensor they wish to remove.

A screenshot of the ‘remove sensor’ screen can be seen in figure 5 on the next page:

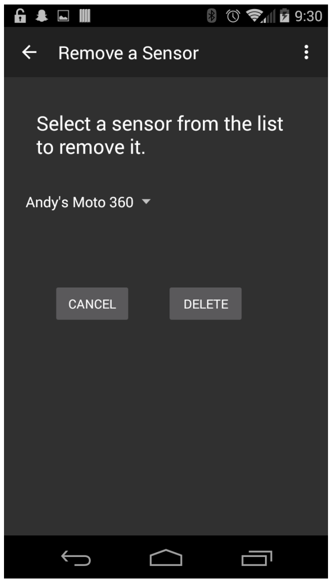


figure 5

- **Uses/Interactions**
  This interface is intended to be used with motion detection sensors and as such would of course need to interact with them. The only interaction with sensors that would be needed is detection of sensor proximity and checking for compatibility. Currently all input for this view comes directly from the user.

- **Resources**
  The following is a list of files that are required by this view:
  1. activity_add_asensor.xml
  2. menu_add_asensor.xml
  3. activity_remove_asensor.xml
  4. menu_remove_asensor.xml

- **Processing**
  The following is a list of files that are used to generate this view's behavior:
  1. AddSensorActivity.java
  2. RemoveSensorActivity.java,

7. System Installation and Execution

Disclaimer: You must have the lastest JDK package installed, must have Android Studio or IntelliJ (since both are supported by the same corporation, Android Studio is just a free and more simplistic IDE) with the SDK packages, and must have a working emulator or Android device capable of running in debug/developer mode to install.

In order to run the current version of the application, a pull request must be made for the code on Github through the repository. Once access has been granted and a local copy is available, import or open the project in Android Studio and it will automatically convert any SDK paths to point to the installation on the local machine. From there, run the project and select either the emulator or the connected Android device (in developer mode). The application will automatically start once it has been loaded onto the device and should function with user input. After this point, it is safe to disconnect the Android device (if an emulator was not used for running the project) without impacting performance or ending the application. It will also be available through the app screen.

8. Glossary

An ordered list of defined terms and concepts used throughout the document.

1. Android - This is a mobile operation system that is developed by Google. It is available on a wide variety of mobile phones and tablets.

2. Bluetooth - This is a wireless technology standard for exchanging data over short distances, usually between mobile devices or other short-range devices.

3. Cloud Computing - A general term for the storage and downloading of data and/or services via the Internet.

4. IDE (Integrated Development Environment) - a software application that provides interactive and comprehensive tools for software development. These consist of a source code editor, build automation tools, and a debugger. Android Studio is the primary IDE referred to in this document.

5. Motion Sensor - These are electronic devices that have the ability to detect both small and large movements. This may included in several mobile devices, such as a smart-watch, or a specialized device that is meant only to detect a wearers movement. Implemented with an accelerometer.

6. SDK (Software Development Kit) - a packaged set of software tools which allows the creation of applications in a certain platform or framework. Android SDK is the one referred to in this document.

7. User Interface - A graphical view of information and forms that a user may interact with. The user interactions will then trigger internal events to create an appropriate action.

9. Bibliography

"Android Developers." *Android Developers*. N.p., n.d. Web. 07 May 2015.

Meier, Reto. *Professional Android 4 Application Development*. Indianapolis, IN.:

Wiley/Wrox, 2012. Print.

"Improving the Profession of Software Development." Scrum.org. N.p., n.d. Web. 09 May 2015.