Musical Heart Rate Adjuster

Software Engineering 14:332:452 – Group #12
https://github.com/revan/HeartRateAdjuster

Kenny Bambridge, Jonathan Chang, Samani Gikandi,
Tae-Min Kim, Nikhil Shenoy, Revan Sopher

May 4, 2014

Contents

1 Customer Statement of Requirements/Project Proposal **7**
- 1.1 Problem . . . 7
  - 1.1.1 More Specifically . . . 7
  - 1.1.2 Background . . . 9
  - 1.1.3 Devices and Specifications . . . 9
- 1.2 Solution . . . 9
  - 1.2.1 Music . . . 10
  - 1.2.2 Database . . . 11
  - 1.2.3 System Architecture Diagram . . . 11
  - 1.2.4 Product Usage . . . 12
  - 1.2.5 Product Ownership (tentative) . . . 13
- 1.3 Glossary of Terms . . . 14

2 System Requirements **15**
- 2.1 Functional Requirements . . . 15
- 2.2 Non-Functional Requirements . . . 17
- 2.3 On-Screen Appearance Requirements . . . 18
- 2.4 Organization . . . 25
  - 2.4.1 Stakeholders . . . 25
  - 2.4.2 Actors and Goals . . . 27
- 2.5 Use Cases . . . 27
  - 2.5.1 Casual Description of Use Cases . . . 28
  - 2.5.2 Traceability Matrix . . . 28
  - 2.5.3 Fully-Dressed Description of Use Cases . . . 30
  - 2.5.4 Use Case Diagram . . . 42
  - 2.5.5 System Sequence Diagrams . . . 43
  - 2.5.6 System Operation Contracts . . . 48
  - 2.5.7 Mathematical Model . . . 49

3 User Interface Specification **50**
- 3.1 Preliminary Design . . . 50
  - 3.1.1 Use Case UC-1: Log Data . . . 50
  - 3.1.2 Use Case UC-2: Set Target Heart Rate . . . 51
  - 3.1.3 Use Case UC-3: Skip Track . . . 54

3.1.4 Use Case UC-4: Toggle Playback . . . 56
3.1.5 Use Case UC-5: Display Statistics . . . 58
3.1.6 Use Case UC-6: Get Heart Rate . . . 60
3.1.7 Use Case UC-7: Provide Music Data . . . 61
3.1.8 Use Case UC-8: Set Resting Heart Rate . . . 62
3.1.9 Use Case UC-9: Calculate Peak Heart Rate . . . 63
3.2 Effort Estimation . . . 65

4 Domain Model **68**
4.1 Concept Definitions . . . 68
4.2 Association Definitions . . . 69
4.3 Attribute Definitions . . . 70
4.4 Traceability Matrix . . . 71

5 Plan of Work **72**
5.1 Gantt Chart . . . 73
5.2 Product Ownership . . . 73
5.3 Breakdown of Responsibilities . . . 74

6 Interaction Diagrams **75**
6.1 Alternate Scenarios . . . 79
6.2 Design Patterns . . . 80
6.3 Assignment of Responsibilities . . . 80

7 Class Diagram and Interface Specification **82**
7.1 Class Diagram . . . 83
7.1.1 Class Diagram for Data Management . . . 83
7.2 Data Types and Operation Signatures . . . 85
7.3 Traceability Matrix . . . 90
7.4 Design Patterns . . . 91
7.5 Object Constraint Language . . . 92

8 System Architecture and System Design **94**
8.1 Architectural Styles . . . 94
8.2 Identifying Subsystems . . . 95
8.3 Mapping Subsystems to Hardware . . . 96
8.4 Persistent Data Storage . . . 96
8.5 Network Protocol . . . 96
8.6 Global Control Flow . . . 96
8.6.1 Execution Orderness . . . 96
8.6.2 Time Dependency . . . 96
8.6.3 Concurrency . . . 97

8.7 Hardware Requirements 97

9 Algorithms and Data Structures 98
9.1 Algorithms 98
9.1.1 Pattern Recognition 98
9.1.2 Danger Detection 99
9.2 Data Structures 100

10 User Interface Design and Implementation 102

11 Design of Tests 103

12 Project Management and Plan of Work 108
12.1 Merging Contributions 108
12.2 Project Coordination and Progress Report 108
12.3 Plan of Work 109
12.4 Breakdown of Responsibilities 109
12.5 History of Work, Current Status, and Future Work 112
12.6 Key Accomplishments 112
12.7 Possible Future Directions 112

Team Profile

Nikhil Shenoy C++, Python

Revan Sopher Android programming, web programming, Java, Python

Tae-Min Kim Java, C++, Python

Samani Gikandi Java, C, Ruby, Network programming, Device driver/firmware programming

Kenny Bambridge IOS programming, web programming, Java, Python

Jonathan Chang documentation, organization, C++

Summary of Changes

Based on the feedback we received from Professor Marsic and the TAs during our previous demo, we decided that a few changes were in order. As Professor Marsic mentioned, it, would be helpful to the user if we could suggest a beneficial target heart rate instead of having them blindly guess and randomly adjust the slider. Also, Professor Marsic pointed

out that there is the potential for the user to overexert himself during a workout. By Therefore, we ended up adding two new features which are also included below in our itemized list and described with further detail.

Obviously, our first demo was a more "bare-bones" presentation of the features we had implemented so far. Between the first and second demo, we have worked first and foremost, on integrating the data subsystem, the UI subsystem, and the chest strap subsystem. Now, we can directly access the graphs and data from the menu off of the home screen. Also, the graphs are "prettier," and there are more variations other than heart rate vs. time that the user can choose from. Suffice it to say, we were able to implement the suggestions given to us, as well as integrate the pieces that we promised.

To view the exact changes made, feel free to visit our project webpage: `https://github.com/revan/HeartRateAdjuster` where each revision is automatically tracked by GitHub.

Itemized List of Changes

Integration of Subsystems The UI can interact with the data subsystem directly, which is necessary for viewing the graphs of user exercise and song statistics.

Critical Condition Alert If the user's heart rate falls outside of a "safe" range, the system alerts the user accordingly. (The algorithm used to detect a critical condition is explained in the algorithms section under "Danger Detection" and cited in reference 17.)

Automatically Calculate Target Heart Rate (The algorithm we used in the critical condition calculation and the target heart rate calculation are described in the algorithms section under "Pattern Recognition" and cited in reference 18.)

UI Refinementsi As demonstrated in the second demo, our main screen has had a few minor changes from the preliminary design section. For example, there additions such as the use case where a user needs to be alerted about a dangerous heart condition, or if a user chooses to set or calculate his heart rate.

screenshots of UI and data subsystems A few screenshots of the current UI and data subsystems have been included to show the changes visually. Sometimes images are easier to comprehend than words.

Improved database/more, better graphs Originally, our SQLite database was complete "barebones," and although the normal database functions such as "add record," worked, it was visually unappealing. Now, instead of having a single line graph, we have a few other options for users to select such as Genre or Artist. The graphs update continuously, and users can zoom in and out of the graphs and drag them around on the screen.

Design Patterns A section that discusses the major design patterns used in our project has also been included.

Object Constraint Language A section on OCL has been added which includes the invariants, preconditions, and postconditions of all the important classes used in our code.

Because of the successful integration, and addition of important features, everything works a lot "cleaner" than compared to the first demo!

1 Customer Statement of Requirements/Project Proposal

1.1 Problem

There seems to be a growing concern over the bevy of health-related issues that society faces: cancer, obesity, heart diseases. This is evidenced by the estimated $25.9 billion that consumers spent on fitness membership in 2013 or the government's seemingly carte blanche spending on "perfecting" the healthcare.gov website. While it is impossible to completely eliminate health problems, we focus on a small, albeit interesting subset of the health industry - personal health monitoring. Just like "an apple a day keeps the doctor away," our project seeks to maintain the personal health of an individual, keeping him in the best physical shape possible, and reducing the risk of health problems.

1.1.1 More Specifically

Lack of education about proper fitness is a widespread problem. Many people in the country would like to exercise and stay in shape, but only a small subset of those people know how to monitor their health in a way that allows them to stay fit. There are several methods out there which people can use to get the proper information; tools such as fitness blogs, the President's Council on Fitness, Sports, and Nutrition, and the classic visit to the doctor's office are all excellent examples. However, many people don't know about those methods or choose not to utilize them, and they do their body a disservice by performing exercises that could be detrimental to their health. The Internet is littered with articles such as "9 Exercises You're Doing Wrong" and "The 7 Fitness Myths You Need to Know". With information like this readily available to exercisers, it can be hard to find correct information. And even if one does find correct information, he must check to see if that information applies to a person with his body shape and size. The general problem of finding correct exercise information is that there is no set standard; there is no "one size fits all" set of guidelines which one can follow to have an effective workout. Everybody's body responds differently to different exercises, so the best that the medical community can do is to provide a set of recommendations for people of the most average body type. While this set of recommendations is good in the general, they will never tailor to the needs of one's body and workout. Finding the correct exercise information for one's

body type is quite a difficult problem, and it will continue to be a problem until a solution is provided to track each person's exercise routine.

Of all the different metrics for measuring the quality of one's fitness, heart rate is the most important factor in determining whether a workout was effective. Monitoring one's heart rate is useful because it determines whether the exerciser is performing his exercise safely as well as successfully. Experts recommend that one's target heart rate during exercise should be between 60-85% percent of the maximum heart rate, and that anything higher than 85% increases cardiovascular and orthopedic risk to the exerciser. Naturally, the target heart rate varies for people of different ages, so one should always take this into account before starting a fitness regimen. Also, the frequency of exercise before the new regimen should be considered. If one has not exercised frequently before starting the new regimen, then he should start exercising at a rate that is towards the lower end of the target heart rate zone and then gradually increase his activity once his body gets accustomed to the exercise. Heart rate is a significant, if not the most important, factor in determining whether a workout was done correctly and effectively, and it must be monitored closely in order to prevent injury.

Unfortunately, there are people who don't know how to correctly monitor their heart rate, and they mistakenly create a certain fitness plan based on wrong information and end up not optimizing their workout. They go to the gym, run on the treadmill at a light pace, and consider that enough to maintain their health. They do not check their heart rate and make sure they are in the safe region of activity. This critical lack of measurement affects the entire workout. For an exercise to be effective, one must maintain a heart rate that is within the target range for an extended period of time. If not, the exerciser either puts himself at risk of injury or completes a workout that does very little to improve his fitness. Some use exhaustion and soreness after a workout as a judge of an effective workout. Although these methods do give an indication as to how effective the exercise was, they do not provide an insightful and accurate description of one's health. As a result, these people continue bad habits and routines that hinder their progress to stay fit; in fact, they may not be even making progress.

A solution to the problem of uninformed exercise must have three main components; it must include all relevant medical data such as heart rate information, create a fitness plan that fits relatively well to the client's body, and provide the client with feedback about the effectiveness of his workout. Once all these components come together, the client will be able to correctly monitor his health during exercise and get the most out of his workout.

1.1.2 Background

A healthy lifestyle depends upon a plethora of factors including environment, nutrition, socialization, and mental stability. However, we identified physical fitness and sleep as the two key factors to leading a healthy lifestyle. Their importance cannot be overstated.

Physical fitness or exercise fortifies the body, allowing one to stay in shape, avoid injuries, develop confidence, become stronger, and sleep better. Sufficient physical activity can reduce the risk of such symptoms as stress, depression, diabetes, high blood pressure, osteoporosis, and obesity.

Meanwhile, sleep is critical to the mind. It refreshes the brain, helps with daily functioning, uplifts one's mood and emotional well being, increases productivity, and improves learning and memory. "Good" sleep can lower the probability of contracting the following: heart disease, kidney disease, high blood pressure, diabetes, and stroke.

1.1.3 Devices and Specifications

Heart Rate monitor:
Uses Bluetooth or ANT+ to connect to smartphone

Smartphone:
Needs to be running Android 4.3+
Needs to have radio supporting Bluetooth 4.0+

1.2 Solution

It has been well documented that exercise and sleep both hold a significant impact on heart rate[14-15]. However from experience, we believe that the link between exercise and sleep and heart rate holds true for the converse as well. One of the targets of a good workout is an increased heart rate. On the other hand, high-quality sleep entails a decreasing heart rate.

Our proposed solution is designed to affect people's health by providing limited control to their heart rates. Our Musical Heart Rate Adjuster is targeted to operate in two areas where it can be the most effective - workouts and sleep - which in turn offer the aforementioned health benefits. We do not plan on adjusting heart rate with the intent of skipping the rigors of exercise or the process of falling asleep; on the contrary, we wish to adjust

heart rate to induce better quality workouts and sleep.

Our plan is composed of a few steps. First, we intend to increase the effectiveness of workouts by matching heart rate to an appropriate selection and tempo of music. This music can be adjusted accordingly to stimulate heart rates to reach a desired intensity of exercise. The music, which will be discussed later, performs the task of simulating workout difficulty. As an added benefit, studies have shown exercising while listening to music to provide many benefits, such as increased motivation and endurance, distraction from otherwise unbearable stress, and increased heart rate, among others.

Then, we seek to improve the quality of sleep by finding soothing music to gradually slow down a user's heart rate. In this instance, we use music as an instrument to aid users in falling asleep more quickly, and hopefully improve the performance of their rest. Listening to right music can also improve the quality of sleep; for instance, music by classical composer Mozart has been shown to increase health factors such as relaxation and mental stimulation.

1.2.1 Music

We utilize music to affect heart rate in two ways. In addition to identifying and playing music with speeds in the same vicinity as heartbeat, we also wish to be able to adjust the tempo of the music. A simple compound microscope has both a coarse adjustment knob as well as a fine adjustment knob. Our song library will organize songs into different categories, acting as a coarse adjuster for heart rates. Meanwhile, to add a little fine-tuning to adjust the heart rates, we will either write or find an existing application for an audio tempo changer. Given current heart rate, and subsequently, current music tempo, we will continually adjust the music tempo while measuring for changes in heart rate. This will occur until we hit the specified target heart rate, give or a take a few BPM. Thus, if there is no difference in heart rate, either the targeted heart rate has been reached - otherwise, the music tempo has not been adjusted enough.

We are interested in analyzing the magnitude of the effect of our music application on heart rate and finding a rough correlation based on the data that the MOTOACTV provides. All parties should remember, however, that correlation is not causation. While we take the assumption that the general public will react to music in similar ways (music with a slower tempo will decrease heart rate while music with a higher tempo will increase heart rate), it is difficult to know how every individual will react to the same music and can never be 100 percent accurate.

This will probably take some experimentation with test subjects in several situations such as rest, running, weight-lifting, and playing basketball. Time-permitting, we will also

find the ability of music to slow down heart rate and affect sleep by analyzing sleep monitor graphs. As a side experiment, we could measure the effect of several well-known classical songs on sleep quality.

Finally, we will be able to develop an algorithm for ranking the songs that induce the best performance. Even better, we could potentially toy around with machine learning to have our algorithm improve after more and more data sets. With the application of machine learning, each user's individual MOTOACTV device may correct itself in the case that a specific user does not follow the general trend as stated previously (a user's heart rate might increase from slow music rather than fast music). This way, our MOTOACTV will be able to increase both exercise and sleep performance through our own custom music player application, located on and loaded by the device. This application will utilize the user's music library stored locally on the device's memory.

1.2.2 Database

Users will want to monitor their personal health status, so our project will allow the user to view his workout data directly on his phone. This eliminates the inconvenience of having the user log in to a personal account on a website to view his data, because everything he needs will be on the phone itself. All the data collected from the workout will be stored locally on the phone, and the system will perform the necessary database calls to retrieve that data. That data will be processed and formatted into different graphs that will display the correlation between music and heart rate.

1.2.3 System Architecture Diagram

This diagram highlights our system architecture: Our heart rate monitor senses the user's BPM and transmits the data to the Android phone via Bluetooth as requested by the app. The phone then uploads the data to the server and database which processes the data. The system is then able to select the appropriate songs, and then display suitable graphs once the workout is completed.

Figure 1.1: System design

1.2.4 Product Usage

- The heart rate monitor should only be worn while it is in use - only while the user is exercising. While it is safe to wear the heart rate monitor during other times, there will be no benefit unless the application is currently running.

- Users may choose to use the Musical Heart Rate Adjuster while not sleeping or exercising if they wish to adjust their heart rate for alternate reasons (possibly for playing video games or preparing for an exam).
- The user will run the android application, and then input a target heart rate. The software will then choose a song based on your current heart rate and begin to either raise or lower it. Once the target heart rate is obtained within a certain tolerance, the software will work to maintain this heart rate rather than increasing/decreasing it.
- Music will be selected from the user's own personal music library (which should be stored on the flash memory of the Android device) to either increase or decrease the user's heart-rate. Music will be played by our software.
- The software will select and play music according to the user's current heart rate in real-time as it receives information from the connected heart rate monitor.
- Music will be delivered through the headphone jack on the Android device or through any bluetooth device.
- Receive information on the songs that are listened to in relation to their usage of the Android device. (What songs were listened to, which songs were the most effective at changing their heart rate, etc.)

1.2.5 Product Ownership (tentative)

Our team will be divided into three smaller sub-teams of two individuals each, the pairings listed below. Each sub-team will be responsible for music, hardware, or web and provide a brief description of their work on a shared Google drive folder. They will also include the necessary UML diagrams and charts. Every week (or bi-week) we will meet together for 1-3 hours during the timeframe determined by When2meet. During the meeting, we will have a specific agenda that primarily involves the week's progress and upcoming deliverable. Our discussion will probably be centered along the following questions: 1) What did you work on this past week? 2) What do you plan on working on next week? 3) Are there any changes that need to be made to the project? Every week, a different team member will take the lead for the next deliverable to ensure that everything is on time.

- Kenny and Samani will develop a system to select or modify a track based on requested BPM. If possible they will incorporate machine learning into the system.
- Jonathan and Nikhil will work on a database that receives, stores, and processes the data from the Android device. They will also be responsible for creating the graphs that measure different metrics of the workout.

- Revan and Tae-min will program the Android application and work on interfacing with the heart rate monitor.

1.3 Glossary of Terms

Electrocardiography (ECG) ECG is an interpretation of the electrical activity of the heart over a period of time as measured across the thorax or chest. This interpretation is produced by attaching electrodes to the surface of the skin. This is generally used to measure the heart's electrical conduction system by picking up electrical impulses generated by the polarization and depolarization of cardiac tissue.

Beats per Minute (BPM) BPM is the amount of times that the heart beats given one minute of time.

Resting Heart Rate The resting heart rate is the heart rate measured while the subject is both awake and inactive, not having performed physical activity prior. This resting heart rate, measured in bpm, is the initial value that the user should have before using our device to raise or lower their heart rate.

Database Databases are a place to store information. In our case, this is where we will store and process important data received from our health devices, allowing our system to simply act as a pleasant interface for the user.

Target Heart Rate The target heart rate is the heart rate which the user wishes to achieve. This will be lower than the recorded resting heart rate if the user is attempting to sleep, and higher than the recorded resting heart rate if the user is planning to work out. The user's maximum heart rate is based on how old the user is (220 minus the user's age), and the recommended target heart rate while exercising is between 50 and 85 percent, depending on how active the user normally is. While sleeping, people's heart rates generally drop approximately 8 percent from their resting heart rate, so the user's target heart rate should be approximately [(heart rate before sleeping)*0.92]

Smartphone Smartphones are mobile phones which contain features that are more advanced than basic mobile phones. In our case, any Android device which has the capability to use Bluetooth will suffice to interact with the sensors which will be put on the body.

Heart Rate Monitor A device which is able to monitor the user's heart rate. In our experiment we will be using a third party heart rate monitor (worn as a chest strap) which has sensors that are connected to the skin along with the MOTOACTV watch. The chest strap will record the heart rate while the watch will display the user's current heart rate in real time.

2 System Requirements

Based upon our consumer needs, we derived a list of requirements for our system to possess. For features that must be implemented by the system, we state that "The user shall," whereas for features that are preferred, but not "mandatory," we state that "The user should." For each requirement, we assign an identifier in the form of REQ-x, as well as a priority weight from 1 to 5. A higher priority weight indicates that the corresponding requirement is more essential to the success of the project, and more critical to fulfilling the customer's needs.

2.1 Functional Requirements

Identifier	Priority	Description
REQ-1	5	The system shall log user BPM data using the Heart Rate Monitor sensor during active periods.
REQ-2	4	The system shall allow user to select a target heart rate on the Android application.
REQ-3	3	The system shall determine a song to play based on whether the target heart rate is greater than or less than the resting heart rate.
REQ-4	5	The system shall play the designated song through either headphones or Bluetooth speakers to adjust user heart rate.
REQ-5	3	The system shall store the BPM data of each song in the database.
REQ-6	2	The system shall at the very least, output graphs relating BPM versus song speed.

REQ-7	1	The system should adjust the tempo of the song to attempt to match the user's BPM and stop when within a defined range.
REQ-8	1	The system should allow the user some control when they use the "Display Statistics" feature. That is, they should be able to customize the details of how the data is displayed (type of graph or specific categories of data).
REQ-9	1	The system should rank the songs that induce the best performance and use machine learning to improve the song selection algorithm.
REQ-10	1	The user should be able to change the current song if he is unsatisfied with it.
REQ-11	1	The user should be able to view his current heart rate as long as the chest strap is recording that information.
REQ-12	1	The user should be able to pause the current track if he needs to interrupt his activity for some reason
REQ-17	1	The system should accept input from the user and calculate a recommended peak heart rate
REQ-18	1	The system should be able to set an appropriate resting heart rate

Our functional requirements spell out the behavior of our system and reaction to user input. Our system is composed of several aspects such as the heart rate monitor, android device, server and database. These requirements describe some of the interactions between these components and the effects that the system as a whole produces. The images in the appearance requirements section later on provide more insight on the requirements and functionality of our system.

For our system to be able to accomplish any of its goals, it must first be able to record the relevant BPM data. Therefore, our REQ-1 is of utmost important. There is, however, an important scenario we must consider. If the the heart rate sensor is removed (accidentally or intentionally) while the user is active, any later data collected and song played may be skewed. Thus, the time in between active periods is irrelevant and will have no effect on the software.

In regards to music playback, it is desirable for our system to do the data processing and song section, to reduce the burden on the user. Again, after collecting the BPM data and storing it in our database, our system will use a pre-determined algorithm to analyze song tempo and bpm correlation to determine song selection (REQ-3 & REQ-5). As for physical playback, the choice of whether to use headphones or speakers will not have any effect on the performance of the system. The choice is simply the user's preference (REQ-4).

To safeguard against mistakes, and prevent negative side-effects, if the system makes an incorrect decision, there will be no negative consequences on the user's health. It should be able to re-adjust once it realizes that the song's tempo does not match the user's current and target heart rates (REQ-7). For REQ-9, this ranking system will be completely local and only relevant to the user of the system. This is just an optional improvement to our system to enhance the user's experience.

In order for an uninformed user to safely select a target heart rate, the system should have a mechanism to calculate a recommended target heart rate based on the user's age and level of activity (REQ-17), maximizing the safety of the system. Along with recommending a target heart rate, the system will be able to set the user's resting heart rate based on their current heart rate at the beginning of the workout (REQ-18), also contributing to the safety of the system.

2.2 Non-Functional Requirements

Require-ment	Priority Weight	Description
REQ-13	5	The Android interface shall have a minimal number of navigation menus; the user should not need more than three taps to find the information he needs
REQ-14	5	The user shall not be able to directly modify any data in the database. All data must be programmatically gathered and processed
REQ-15	3	The user should wear the device only when the user wishes to alter their heart-rate; the device will not provide useful information if it is worn when the user does not plan to increase or decrease their heart-rate.
REQ-16	3	The Android application should be intuitive and simple to use.

Meanwhile, our non-functional requirements are more descriptive than practical, listing the qualities of our system. These requirements are based on the term FURPS+, which includes functionality, usability, reliability, and performance.

2.3 On-Screen Appearance Requirements

This section contains mockups of the Android application's user interface. Although the arrangement and display is subject to change, these images contain all the essential information that needs to be conveyed to the user, as well as all the necessary inputs.
The inputs used while exercising, such as the BPM sliders and the music controls, take up a large amount of screen space to facilitate active use. Information display, such as the current track and BPM, is placed unobtrusively around the input methods. The configuration settings are hidden in a drop-down menu, as per the Android design standard.

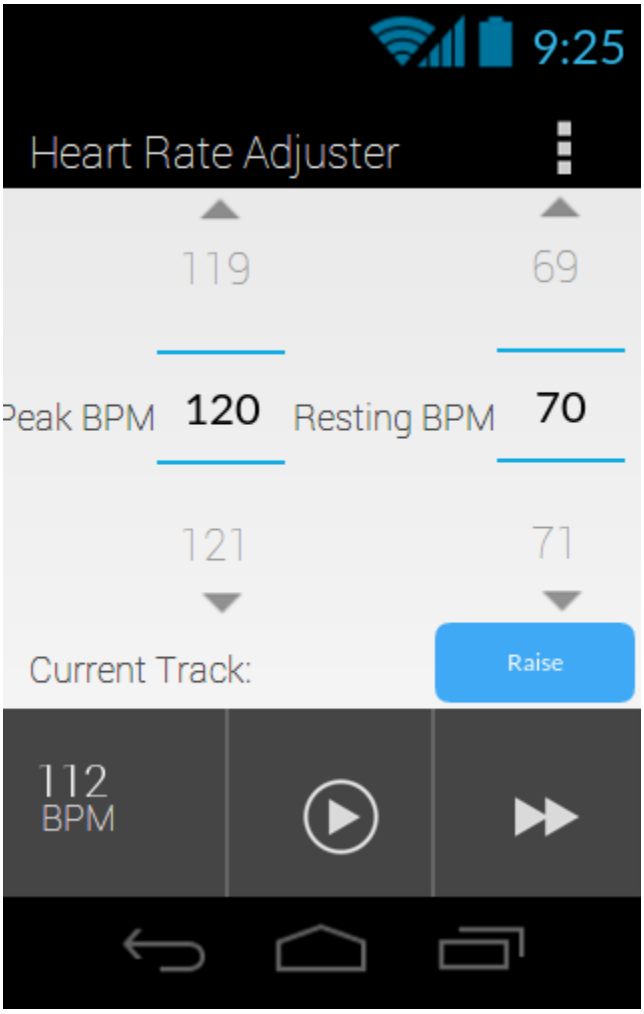


Figure 2.1: The main screen of the app provides a menu button, selectors for Target Peak and Resting BPM, a display of the current track, a display of the current BPM, and the option to Play/Pause and Skip the current track.

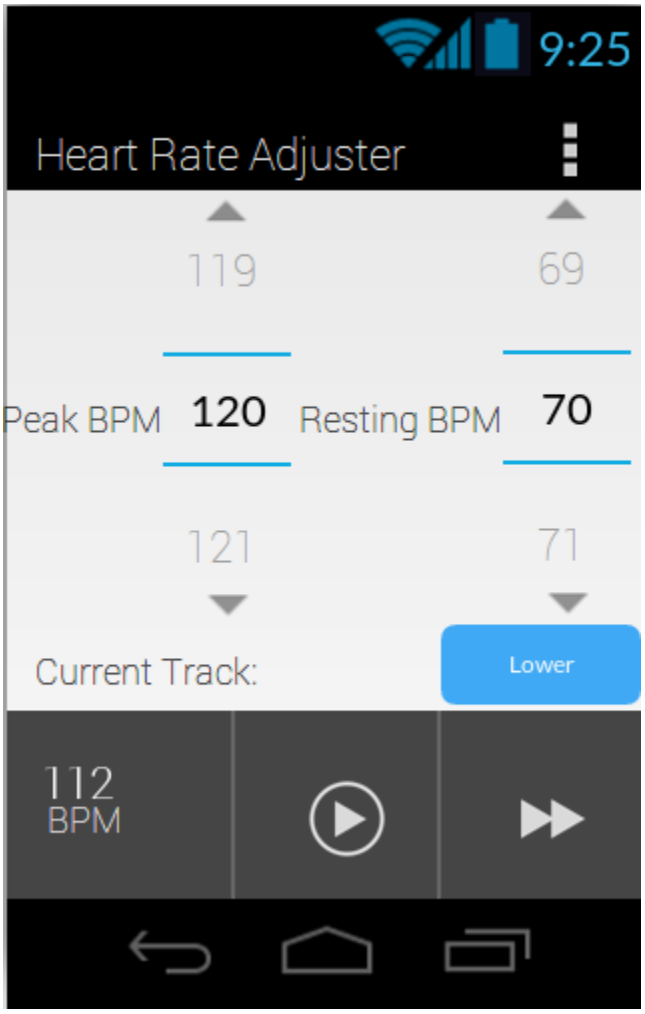


Figure 2.2: Pressing the "Raise/Lower" button toggles between attempting to raise or lower the BPM.

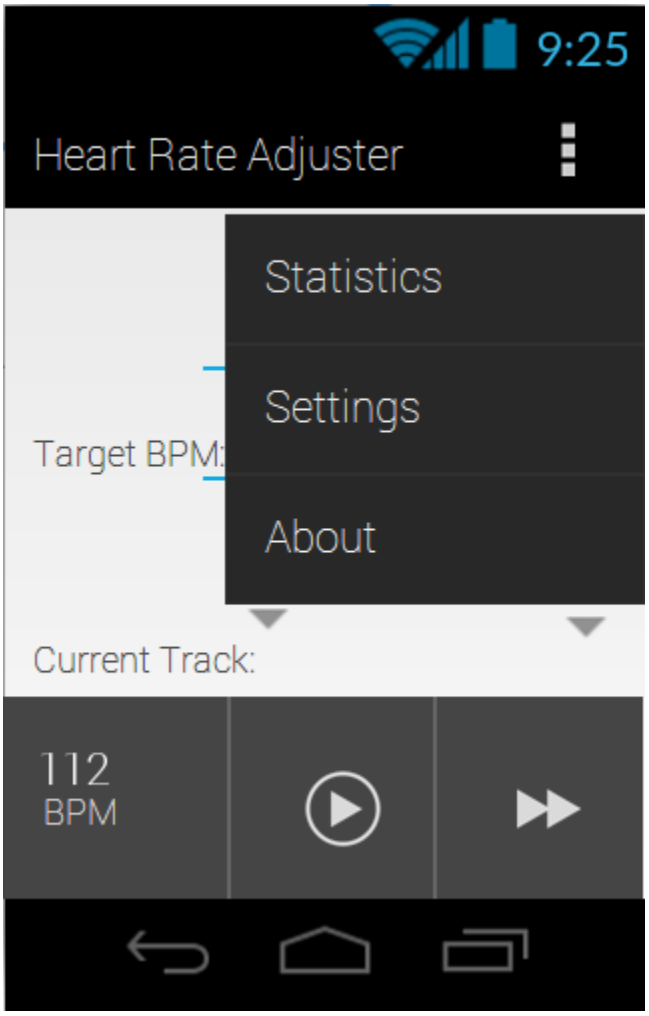


Figure 2.3: Pressing the menu button opens the context menu, providing the option to view statistics, edit settings, and view information about the app.

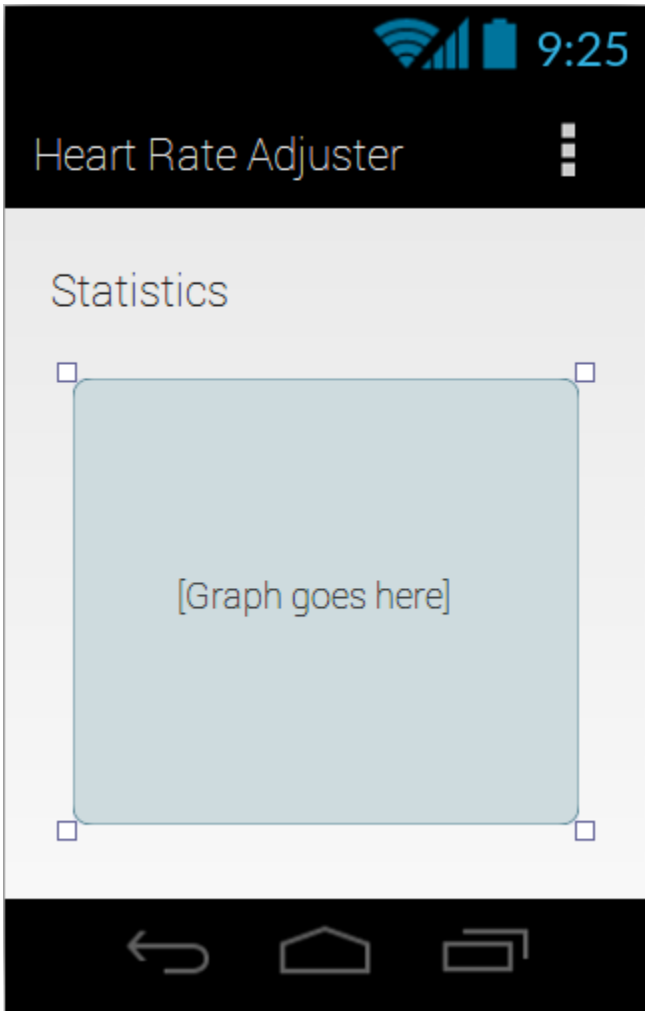


Figure 2.4: Selecting the Statistics option from the context menu displays a graph of user heart rate and song transitions.

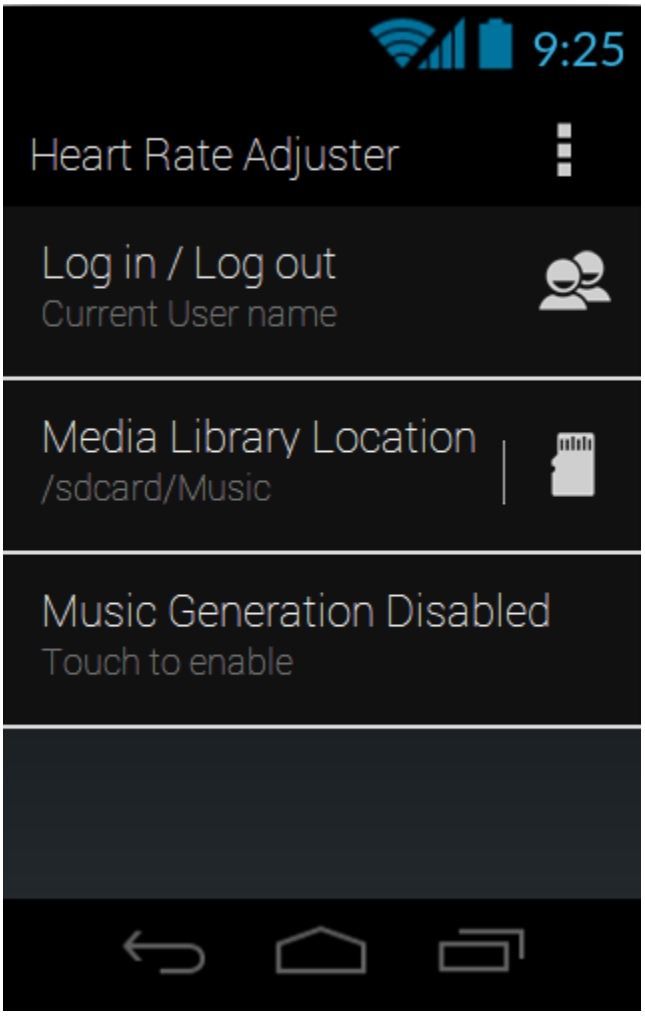


Figure 2.5: The settings page allows the user to open a menu to Log in, to edit the location of the media library (this is done via the OS's directory selection), and configure additional parameters such as music generation (if there is time to implement this feature).

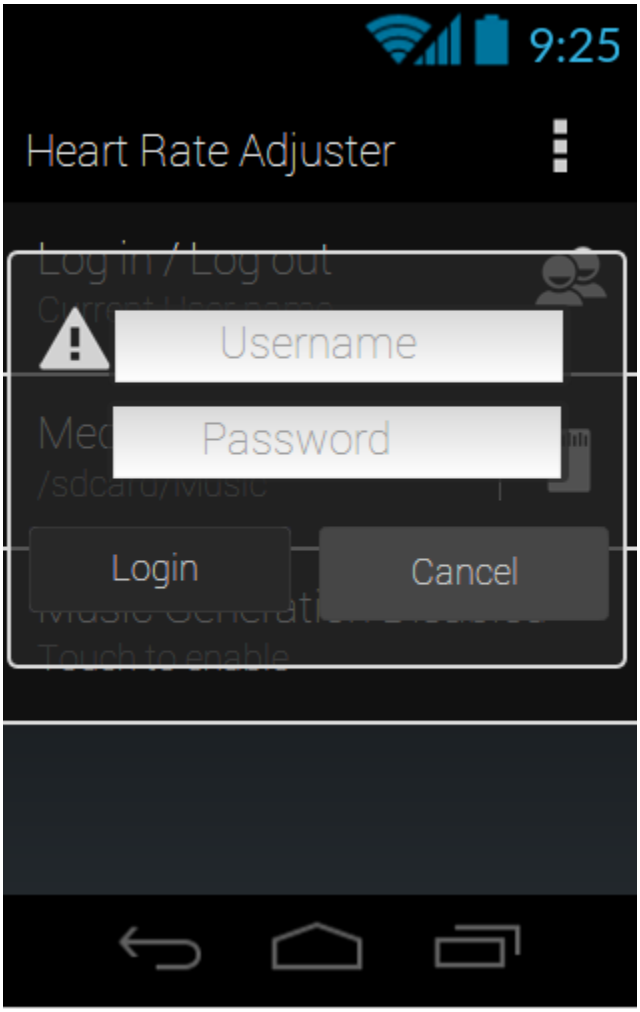


Figure 2.6: Selecting the Log in option prompts the user for their credentials.

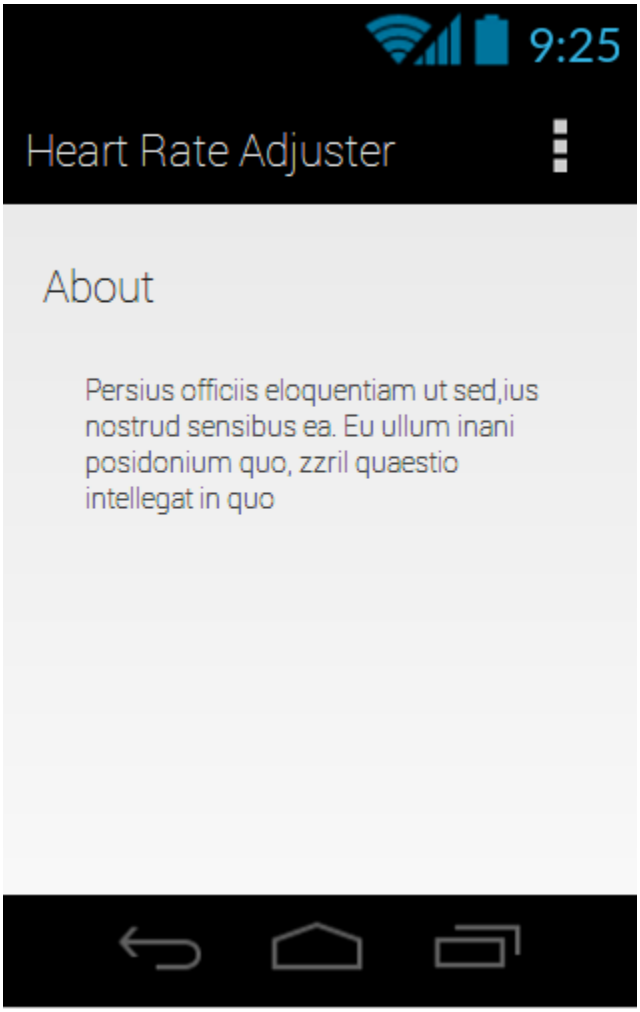


Figure 2.7: Selecting the About option from the context menu provides a description of the application.

2.4 Organization

2.4.1 Stakeholders

Stakeholders include individuals and organizations which are interested in the completion and use of a given product. The amount of stakeholders and different types of stakeholders relies on the versatility and ease-of-use of the product in question. Due to this software's very simple interface and design, stakeholders may include users of all ages and multiple types of organizations who are interested in obtaining easier sleep or a more energetic workout. Examples of potential stakeholders include:

1. Individuals who are interested in maintaining their health personally without out-

side help. With the many functions of the application, users have the capability of maintaining their health without the need to consult other people. People who are introverts or do not have easy access to another person who is able to easily analyze the individual's personal health would be very interested in this application. After running this application through their workout or sleep, users can easily consult the graphs which are produced rather than consulting a personal trainer or doctor about their health.

2. Organizations that specialize in helping people fall asleep. Rather than having to prescribe pills to every customer who has trouble sleeping, they will have the option to suggest this product to the customer for minor cases. While prescribing pills may tend to have slightly more dangerous side-effects, our product does not introduce any chemicals to the body which may potentially cause harm to the consumer. Organizations who are interested in a cleaner alternative to help people with their sleeping problems would be stakeholders for this product.

3. Organizations that specialize in promoting exercise and personal health. Not only does this product help those who are trying to sleep, but also those who wish to be more fit. While personal trainers may know how to help the customers and be great motivators, organizations may be interested in helping a larger pool of customers without having to increase the amount of hands that they have working. With this product organizations may grant customers the option of being self-sufficient, helping to increase self-esteem, as well as a great motivator as the application works to increase the user's heart rate allowing them to push onward and burn calories easily.

4. Organizations interested in monitoring and researching people's health. While there are many users who are able to use the product's graphs and understand how their health and workout are, there are many users who still prefer the assistance of outside sources. This product may also be used by these outside sources to help them collect extra data on an individual's health. Rather than having the customer come to their location and run a couple tests in a single day, the organization will have the ability to provide this product to the customer and collect more regular data to understand the customer's day-to-day life rather than a couple of tests run at their office.

More specifically, this product may see stakeholders in:

- Personal Trainers
- Athletes
- Coaches
- Doctors

- Researchers
- Pharmacies
- Therapists
- General population

2.4.2 Actors and Goals

Actors can be defined as are people or devices that will directly interact with the product, and can also be loosely labeled as either "initiators" or "participators". These actors will have a specific goal with the given product, which is what the actors are attempting to achieve by interacting with the system. Actors and their respective goals are:

Actor	*Actor's Goal*
User(initiator)	To increase heart rate for exercising
User(initiator)	To decrease heart rate for sleeping
User(initiator)	To analyze health information from given graphs
Chest Strap (initiator)	To alert the user of an abnormal heart rate
Chest strap (participator)	To monitor the user's heart rate

This product is one which only requires the interaction of one human actor, the user of the product. While there is the potential for other humans to interact with the user's health information which is produced, only the user himself is considered an actor. The headband and chest strap are participating actors that are worn by the user to monitor information and relay the information via Bluetooth back to the smartphone which is running the application. The one exception is that our chest strap (used interchangeably with heart-rate sensor) may be an initiating actor and notify the user if his/her heart rate is abnormally high or low. In this case, the user would be the participating actor.

2.5 Use Cases

Use Cases are specific tasks that are created together by the designer and the client to simulate what the client wants out of his software solution. They are meant to describe the main features of the project such that the designer can easily address the needs of the client and create a product around those needs. Below is a casual description of the use cases for the reader to get a general idea of how the software should be used. Later, fully described use cases are shown for additional insight into the different cases.

2.5.1 Casual Description of Use Cases

Use Case	Action	Description
UC-1	logData	The system will log heart rate and music metadata.
UC-2	setTarget-HeartRate	The user can change the heart rate that the system is targeting.
UC-3	skipTrack	The user can elect to skip the current song. The system will begin playing a different song.
UC-4	toggle-Playback	The user can toggle the system between playing and not playing music.
UC-5	dis-playStatis-tics	The user can request the statistics about the current workout. This can be performed while the workout is in progress or after the workout has been completed.
UC-6	getH-eartRate	The user can view his current heart rate. This can be used when the user does not want to see all of the statistics from the workout and just wants his heart rate.
UC-7	alertUser	The heart-rate sensor detects an abnormally high or low heart rate and notifies the user at once.
UC-8	setResting	The user can set their resting heart rate to be their current heart rate
UC-9	calculate-Peak	The user can have the system calculate a recommended target heart rate

2.5.2 Traceability Matrix

The Traceability Matrix allows the reader to cross the functional and non-functional requirements described earlier with the use cases. This demonstrates which use cases fulfill each requirement, and the total priority weight of each use case will determine which cases are the most important. If an X is present at any point in the the column for a Use Case, then the corresponding requirement's priority weight must be added to the sum. The remaining Xs in the column are similarly considered, and the total priority weight for the Use Case is listed at the bottom of the column.

	Priority Weight	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9
REQ-1	5	X			X	X	X	X		X
REQ-2	4	X	X						X	
REQ-3	3	X	X	X						
REQ-4	5	X		X						
REQ-5	3	X				X		X		
REQ-6	2					X				
REQ-7	1	X	X	X						
REQ-8	1					X				
REQ-9	1					X				
REQ-10	1			X						
REQ-11	1						X	X		
REQ-12	1				X			X		
REQ-13	5	X	X	X	X	X	X	X		
REQ-14	5	X	X			X				
REQ-15	3	X	X	X	X	X	X	X	X	X
REQ-16	3	X	X	X	X	X	X	X	X	X
REQ-17	1									X
REQ-18	1				29				X	
Total Weight		37	37	21	17	25	19	21	11	12

2.5.3 Fully-Dressed Description of Use Cases

Use Case UC-1: logData

Related Requirements: REQ-1, REQ-2, REQ-3, REQ-4, REQ-5, REQ-7, REQ-13, REQ-14, REQ-15, REQ-16

Initiating Actor: User Interface

Participating Actor: Data Manager

Actor's Goal: Begin logging data about the user's heart rate and about the song currently being played.

Preconditions:

- The user has not begun his workout
- The user is wearing the device correctly; the chest strap is securely fastened to the user's chest near the solar plexus, and the musical heart rate application is open to the main screen on the Android device.

Postconditions:

- The system starts recording the initial heart rate, initial time stamp, and music data from the workout, if not already recording.
- The data is stored internally on the Android device in an SQLite database.

Flow of Events for Main Success Scenario:

- → User Interface calls the storeInitialState() function in the Data Manager to record the initial heart rate and time stamp of the user.
- The Data Manager calls the storeCurrentHeartRate() while there is no stop signal from the User Interface. This retrieves the current heart rate from the chest strap, combines it with the time stamp, and stores it in the database.
- → UI sends the stop signal, and the Data Manager stops recording data in the database.
- ← Data Manager sends signal back to the User Interface to indicate that the recording has stopped successfully.

Flow of Events for Alternate Success Scenario (Start Error):

- → User Interface calls the storeInitialState() function in the Data Manager to record the initial heart rate and time stamp of the user
- Chest strap reports an error in measurement. Sends signal to Data Manager about invalid data.
- ← Data Manager returns signal to the User Interface that the data was unable to be retrieved and that the data logging has not begun.

Flow of Events for Alternate Success Scenario (Error During Data Logging):

- → User Interface calls the storeInitialState() function in the Data Manager to record the initial heart rate and time stamp of the user.
- → Data Manager starts recording data in the database.
- ← Chest strap reports at least 10 successive errors in measurement. Sends signal to Data Manager about invalid data.
- ← Data Manager stops recording data. Sends signal that invalid data was received from the chest strap, but some data was recorded.

This use case describes how the system will begin storing data. The User Interface, via the commands entered by the user, will initiate the data storage by calling the storeInitialState() function in the Data Manager. The Data Manager will then retrieve the initial state of the system and current time stamp of the system and then store it in the database. Then the Data Manager will loop into the storeCurrentHeartRate() call until it receives a stop signal from the User Interface. Once this signal is received, the Data Manager will stop the loop and send a signal back saying that the data logging has stopped successfully. Two error scenarios could occur during this use case; the initial retrieval of the user's state could be unsuccessful, or a particular retrieval during the data logging could be invalid. To address the first case, the Data Manager will check for a signal from the chest strap to make sure that it is ready to transmit data and that the reading the strap picks up is correct. If the Data Manager receives a low signal, then it will send a signal to the User Interface that the storage of the initial state did not succeed. If the chest strap records some invalid data during the data storage, then it will send a signal to the Data Manager that an invalid value was recorded. The Data Manager will keep a counter of how many successive invalid entries were received. If the number of consecutive invalid entries crosses 10, then the Data Manager will send a signal to the User Interface that the chest strap is

recording invalid values. Thus, this use case accounts for the success of the main scenario and reactions to the two error scenarios.

Use Case UC-2: setTargetHeartRate

Related Requirements: REQ-2, REQ-3, REQ-7, REQ-13, REQ-14, REQ-15, REQ-16

Initiating Actor: User

Actor's Goal: To change the heart rate that the system is targeting

Preconditions:

- The system displays the selection menu for heart rate

Postconditions:

- The system updates the target heart rate used for music selection

Flow of Events for Main Success Scenario:

→ User selects "Raise" option on main UI

→ System sets current value of "Raise" selector as current target

OR

→ User selects "Lower" option on main UI

→ System sets current value of "Lower" selector as current target

OR

→ User modifies current selector value on main UI

→ System sets new value of selector as current target

In this use case, the user can modify the heart rate targeted by the music selection algorithm. This can be achieved by modifying one of the UI selectors, or by toggling the direction (raise or lower). For this reason, this could be split into several use cases, but since the functionality is the same we consolidate into one.

Use Case UC-3: skipTrack

Related Requirements: REQ-3, REQ-4, REQ-7, REQ-10, REQ-13, REQ-16

Initiating Actor: User

Actor's Goal: To play a different song.

Preconditions:

- The system is currently playing music.

Postconditions:

- A different song is being played at the same rate at which the previous song was playing

Flow of Events for Main Success Scenario:

- → User selects the "Skip Track" button.
- → Mobile interface requests a new song from the Music Selector
- Music Selector retrieves new track from file system while maintaining the current rate of workout.
- → Music Selector passes the new song to the Music Player
- ← Music Player begins playing the new track

The skipTrack case is one of the conveniences for the user. If the user does not like the song he is currently listening to, he can select a button on the Android device to advance to a new song. The Mobile Interface will request a different song from the Music Selector. The Music Selector will choose a song that will be adjusted to match the path that the algorithm has set out to reach the target heart rate. The songs will be selected from the user's music library which has already been loaded onto the device. In the case that the device does not contain another song which matches the current song's bpm/tempo to switch to, the device will select a song from the next highest/lowest level to reach the target heart rate (a faster song if heart rate is to be increased, a slower song if heart rate is to be decreased). Although the song may be out of range for the user's current heart rate, there will be no negative effects of using a song which is only slightly lower or slightly higher. The device will not choose a song that is very far out of the current range.

Use Case UC-4: togglePlayback

Related Requirements: REQ-1, REQ-12, REQ-13, REQ-16

Initiating Actor: User

Actor's Goal: Toggle the playback of music (pause or play).

Preconditions:

- The system is currently working.

Postconditions:

- If the system was already playing a track, the track will stop. If the system was not already playing a track, it will play the current one.

Flow of Events for Main Success Scenario:

- → User selects "Pause" option on mobile interface.
- → Mobile interface tells the Music Selector to hold its current state and the Music Player to stop playing music.
- ← Mobile interface displays a play button so that the user can resume the workout.

OR

- → User selects "Play" option on mobile interface.
- → Mobile interface tells the Music Selector to continue its paused state and the Music Player to continue playing music.
- ← Mobile interface displays a pause button so that the user can pause the workout.

The togglePlayback case is another straightforward, convenience-based use case. If the user needs to interrupt the workout for some reason and needs to stop the music, then all the user has to do is press the pause button on the device. To resume the music, he must press the button again, which will now be a play button. The system will make sure that this function is working properly. If no music is currently being played, it is considered to be paused and may be resumed. If music is being played, it is considered to be resumed and may be paused. The system will know whether music is playing or not. The heart

rate monitor shall also be paused/resumed as the music is. If it is not already recording, and should be, it will start recording (refer to postconditions for UC1, UC2).

Use Case UC-5: displayStatistics

Related Requirements: REQ-1, REQ-5, REQ-6, REQ-8, REQ-9, REQ-13, REQ-14, REQ-16

Initiating Actor: User Interface

Participating Actors: Data Manager, Data Assembler, Graph Container

Actor's Goal: Return graphs about the user's workout.

Preconditions:

- The system is no longer playing music.
- The system is no longer loggin data.
- The user is no longer working out.
- The User Interface has completed error checking on the user's request for graphs.

Postconditions:

- The User Interface will receive graphs of workout data that it requested through the Data Manager

Flow of Events for Main Success Scenario:

- → User Interface makes call(s) to any or all of the following functions: getArtistVsBPM(), getGenreVsBPM(), getTempoVsBPM(), getHRVsTime(), or getPlaylist().
- → The Data Manager will then make calls to the appropriate âĂIJassembleâĂİ function.
- The assemble function will retrieve the data from the database, package it as either an ordered pair of doubles or an ordered pair of a string and a double (for the music sections).
- → The Data Assembler will then be passed to the Graph Container, which will extract the data from the Data Assembler and graph the data that it contains. These graphs will be stored as an array in the Graph Container.

Use Case UC-6: getHeartRate

Related Requirements: REQ-1, REQ-11, REQ-13, REQ-15, REQ-16

Initiating Actor: User

Actor's Goal: View the current heart rate.

Preconditions:

- The device should already be monitoring the user's heart rate.

Postconditions:

- The current heart rate is displayed on the screen of the Android application.

Flow of Events for Main Success Scenario:

→ Mobile Interface requests current heart rate from chest strap.

← Chest Strap returns the current value of the heart rate.

← Mobile Interface displays the heart rate to the user.

The getHeartRate case is similar to the displayStatistics case, but it allows the user to see only his current heart rate. The full analysis provided by getStatistics may not be necessary at times, and this case allows the user to easily see his heart rate during the exercise. Once a second, the system requests the current heart rate from the chest strap. The chest strap then returns the heart rate, and the User Interface displays to the screen. For this function to work, the chest strap must be strapped firmly to the chest in the region of the heart. If not, the Chest Strap will be unable to record the current heart rate correctly. Also, the chest strap should not be moved or tampered with in any way while the device is recording the current heart rate. If no data is received from the chest strap, the system will present the user with a message saying that the chest strap is not properly fastened.

Use Case UC-7: alertUser

Related Requirements: REQ-1, REQ-5, REQ-11, REQ-12, REQ-13, REQ-15, REQ-16

Initiating Actor: Chest Strap

Participating Actors: Human User, Data Manager

Actor's Goal: Alert the user when an abnormal heart rate is detected.

Preconditions:

- The user is currently in the midst of a workout session.
- The user is wearing the device correctly; the chest strap is securely fastened to the user's chest near the solar plexus, and the musical heart rate application is open to the main screen on the Android device.
- The system is functioning properly.

Postconditions:

- The User Interface displays a warning notification to the user.
- The User has the option of stopping the workout session or ignoring the notification completely.

Flow of Events for Main Success Scenario:

- → The Chest Strap continally gathers heart rate data and sends it to the Data Manager as long as there is no stop signal given.
- For every piece of data received, the Data Manager checks the heart rate using an algorithm described later on in the report to determine if it is in the appropriate range.
- → If the Data Manager detects that a heart rate is outside of a safe range (above the normal maximum or below the normal minimum), the Data Manager communicates sendAlert() to the UI.
- → The UI displays a warning notification to the user and advises the user to end his workout session.
- ← The user responds by stopping the workout session on the UI. (The UI sends a stop signal to the Data Manager which then discontinues logging data from the Chest Strap.)

Flow of Events for Alternate Success Scenario (Ignore Warning):

- → The UI displays the warning notification to the user, advising him to end his workout session.
- ←The user chooses to ignore the notification and continues his workout
- The Data Manager continues to log data from the Chest Strap.
- → If after 15 data points, the user's heart rate has not fallen into the acceptable range, the system automatically pauses.
- → The UI informs the user that the system has been paused because it is not safe to use, and advises the user to consult a physician.

This use case describes the unfortunate scenario where, during the course of a user's workout, his heart rate has become dangerously high or dangerously low. The system therefore needs to notify the user of his condition. The Data Manager, which records the information from the chest strap makes the detection, and communicates to the UI to display a warning message. Normally, a user would take the advice of the notification and stop his workout. However, the user may choose to ignore the message, and if after 15 seconds, his heart rate has not dropped into the normal range, the system automatically pauses. A second message is sent informing the user of the pause, and the user is advised

to see a health care provider. Our device is geared primarily towards the casual workout enthusiast, so prime athletes who can stand extreme heart conditions would not likely use this device. Meanwhile, if the user has a pre-existing heart condition where he might receive this warning, it would be best for him not to use our device. In any case, it is better to be safe and pause the system.

Use Case UC-8: setResting

Related Requirements: REQ-14, REQ-16, REQ-18

Initiating Actor: User Interface

Participating Actors: Human User

Actor's Goal: Set the resting heart rate of the current user

Preconditions:

- The user has not begun his workout
- The user is wearing the device correctly; the chest strap is securely fastened to the userâĂŹs chest near the solar plexus, and the musical heart rate application is open to the main screen on the Android device.
- The heart rate monitor is currently recording data
- The resting heart rate has not yet been set

Postconditions:

- The resting heart rate of the user will be set to the user's current heart rate
- The UI will show the user's current heart rate as the user's resting heart rate

Flow of Events for Main Success Scenario:

- → User Interface makes call(s) to any or all of the following functions: getCurrentHeartRate(), setRestingHeartRate()
- → The System records the user's current heart rate
- → The System sets the user's resting heart rate as the user's current heart rate.

This use case describes the scenario where the user is initiating his Heart Rate Adjuster to begin his workout. The user must press the "Set Resting" button on the application so that the Heart Rate Adjuster will be able to accurately determine the user's resting heart rate. Assuming that the user has not yet begun his workout, the user's current heart rate and resting heart rate will be the same. Due to this fact, we may set the resting heart rate in the system to the current recorded heart rate of the user.

Use Case UC-9: calculatePeak

Related Requirements: REQ-14, REQ-16, REQ-17

Initiating Actor: User Interface

Participating Actors: Human User

Actor's Goal: Recommend a target heart rate for the user

Preconditions:

- The user does not know what their target heart rate should be.
- The user knows their age and approximate level of activity ranging from light to intense.

Postconditions:

- The system will calculate and recommend a target heart rate for the user's workout

Flow of Events for Main Success Scenario:

- → The user presses the "Calculate" button on the main screen
- → The user enters his age and approximate level of activity
- → The system accepts the user's input and calculates a recommended target heart rate
- → The system will show the user what his recommended target heart rate is.

This use case is very useful for maintaining the user's safety. Since there will be many users who are unaware of what target heart rate they should enter for the system, this use case allows them to input minimal personal information and be given a recommended

target heart rate. By implementing this use case, the amount of people who enter dangerous target heart rates will be minimized. The system simply requires the target's age and level of activity and is able to determine a safe target heart rate for the user.

2.5.4 Use Case Diagram

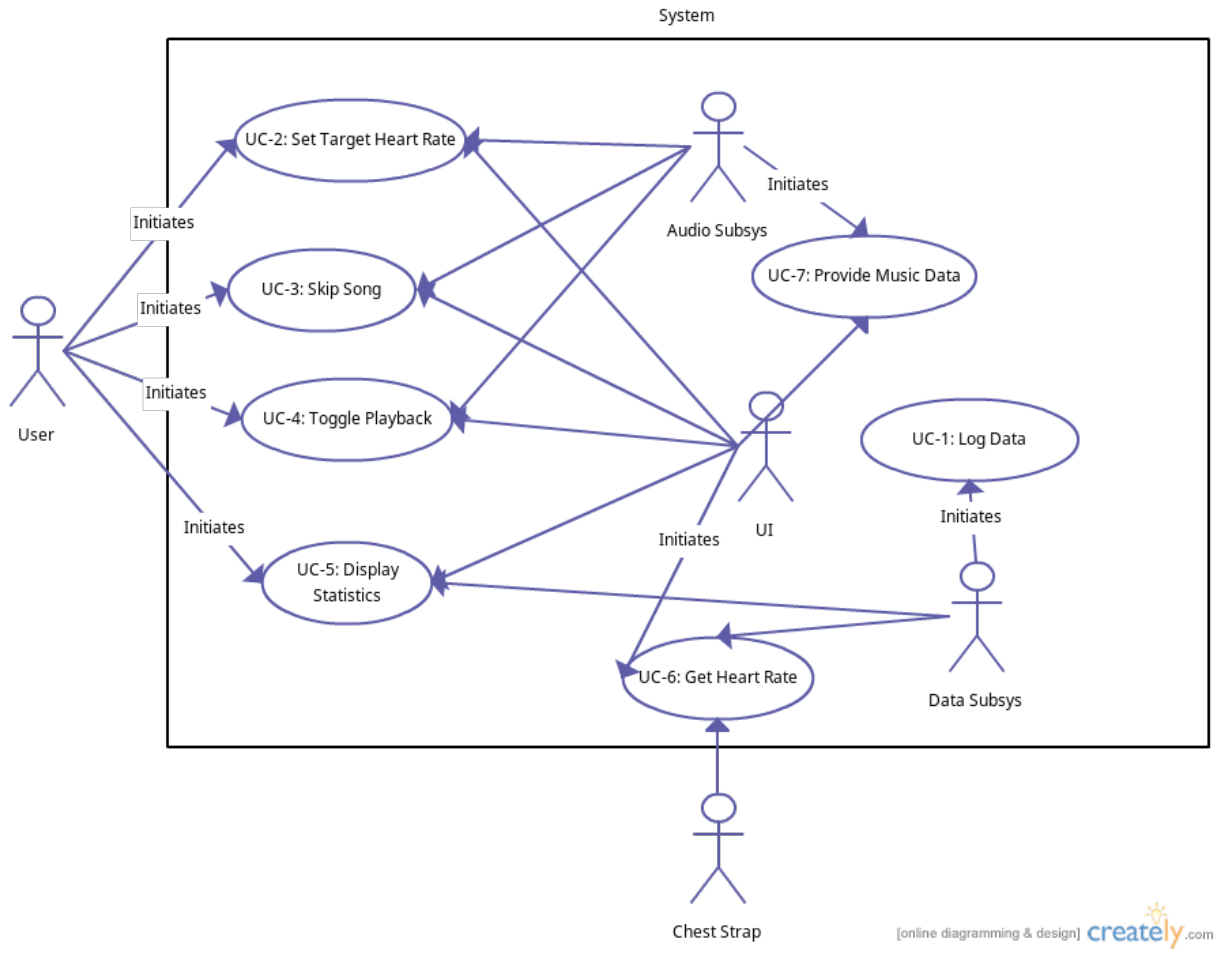
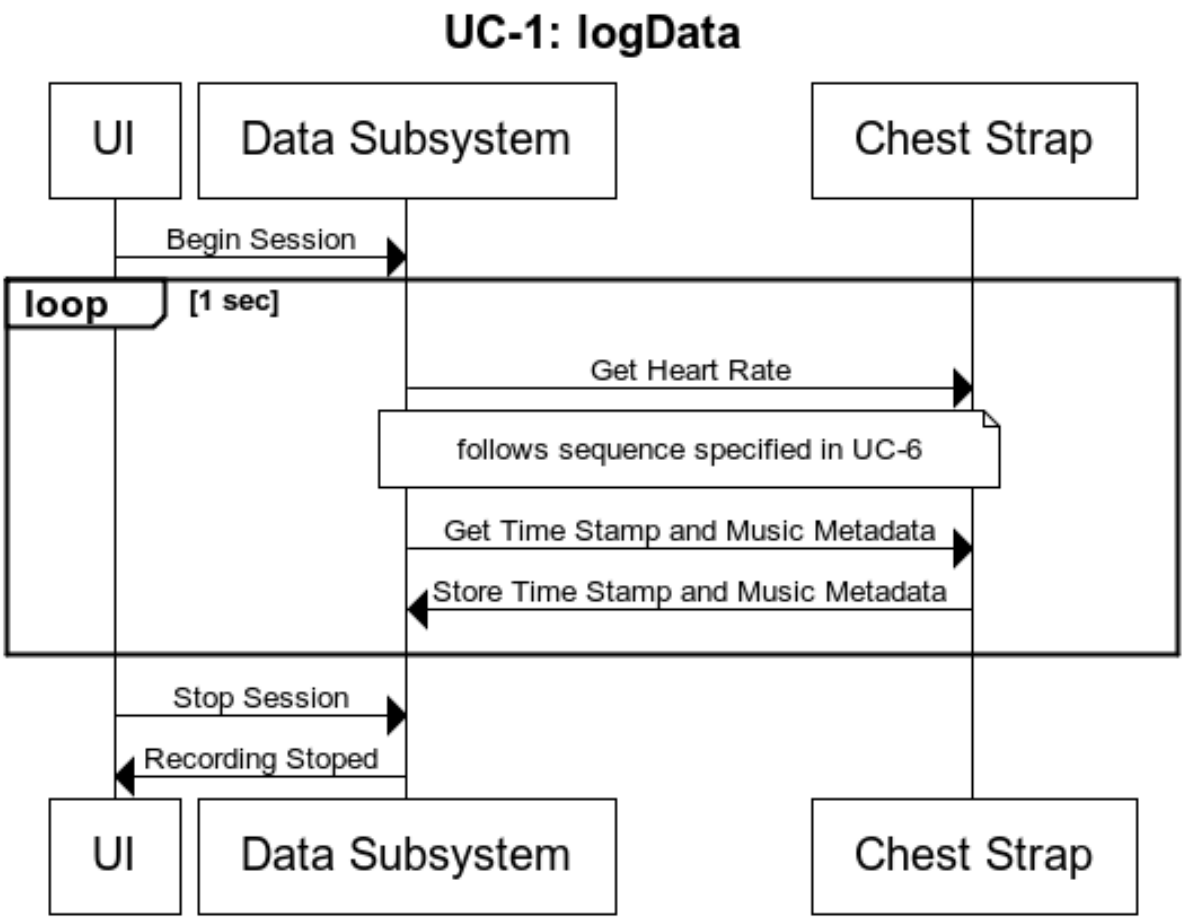
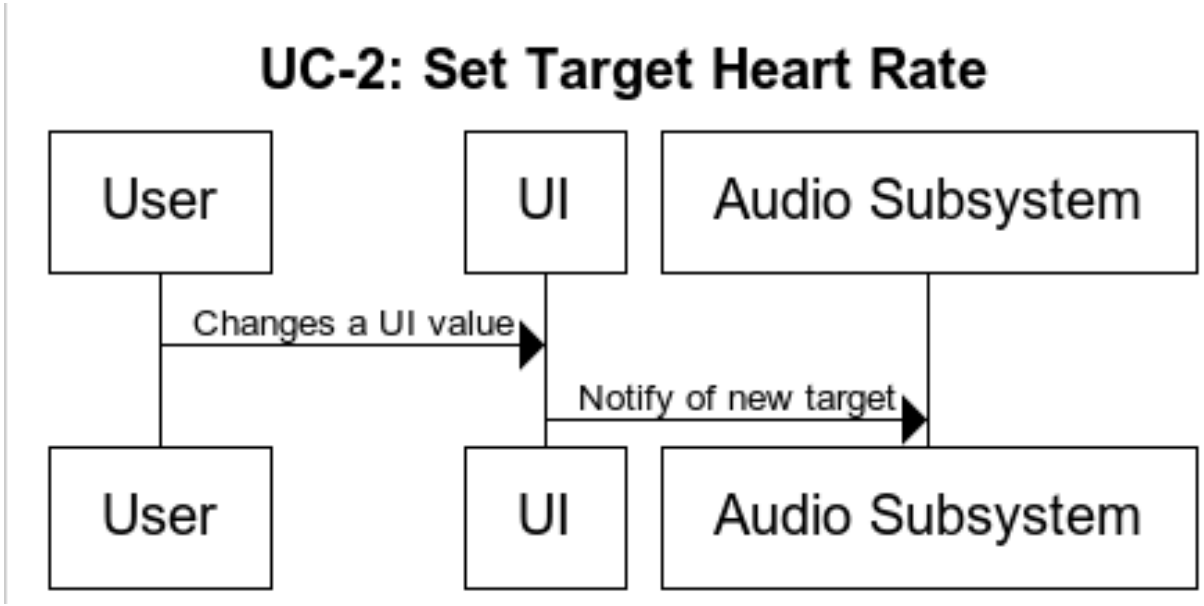
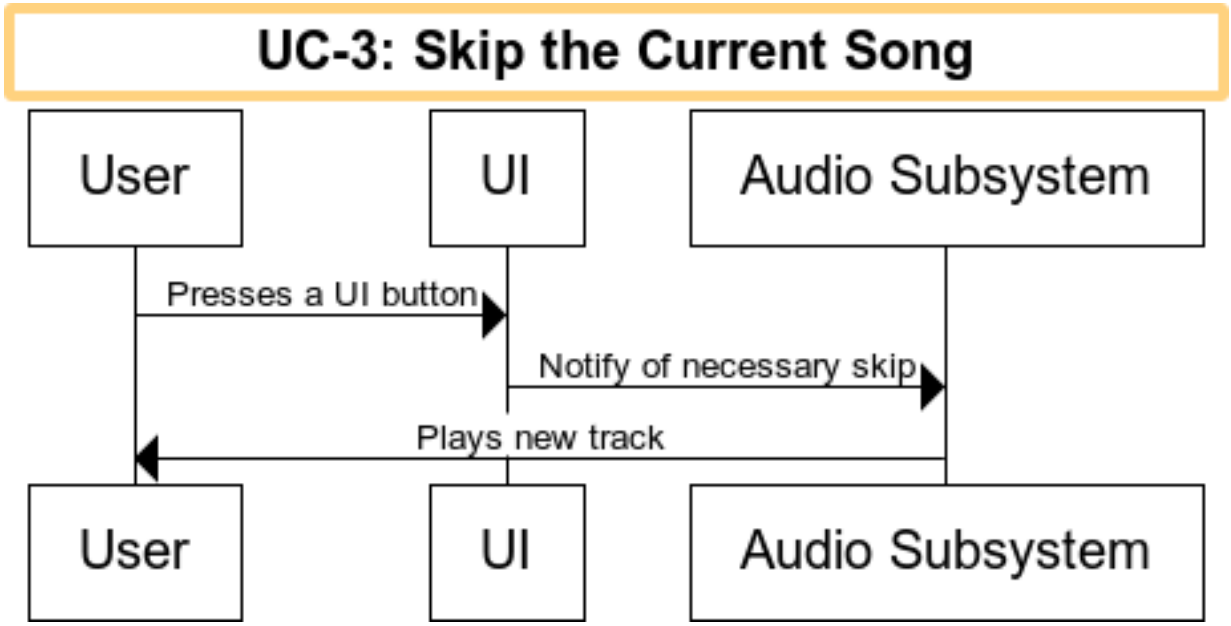


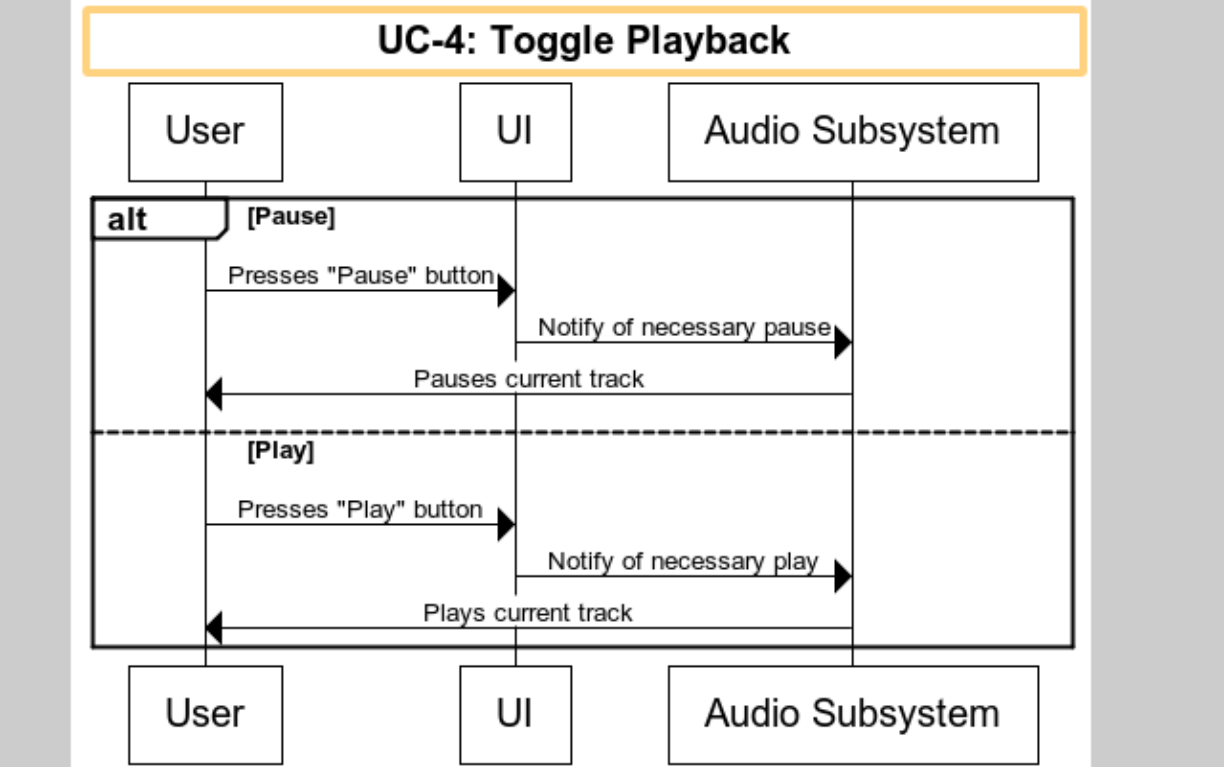
Figure 2.8: Arrows imply participation unless specified

2.5.5 System Sequence Diagrams

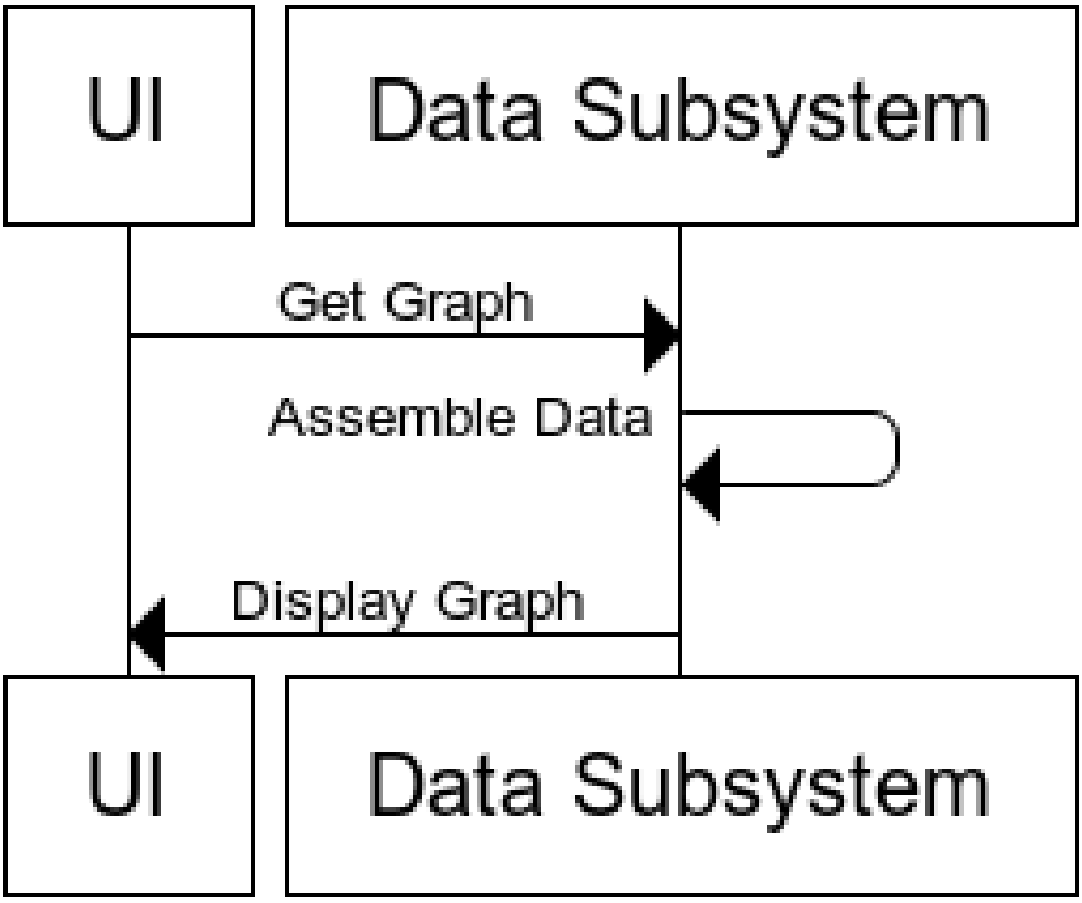



UC-2: Set Target Heart Rate
User
UI
Audio Subsystem
Changes a UI value
Notify of new target
User
UI
Audio Subsystem

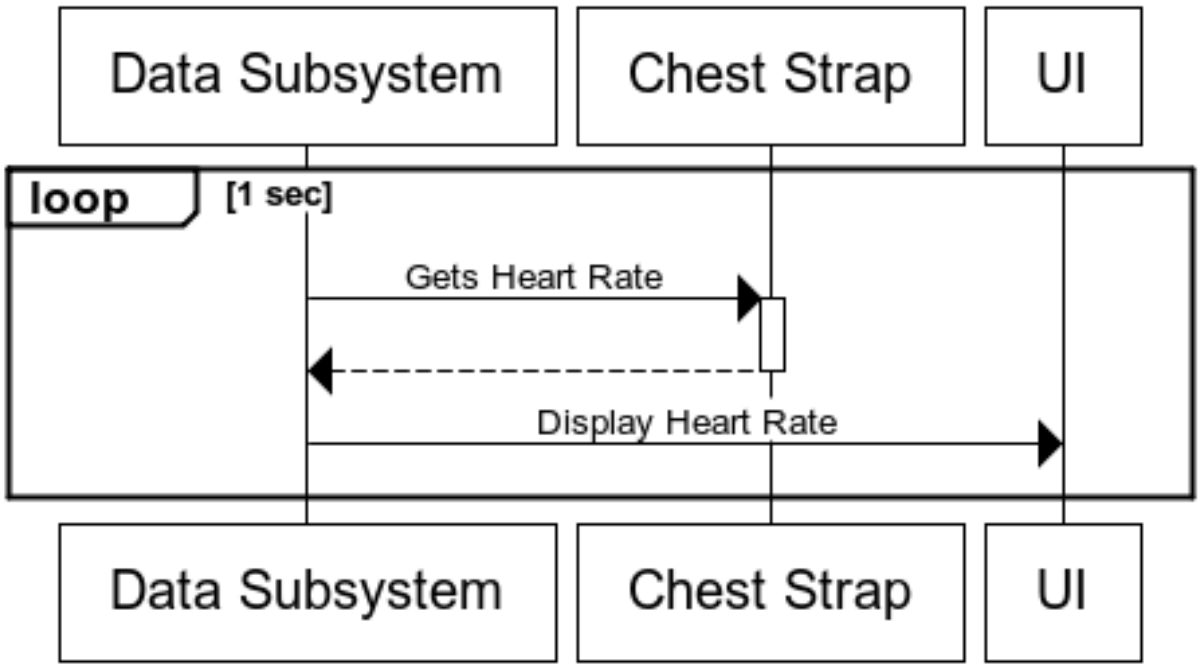

UC-3: Skip the Current Song
User
UI
Audio Subsystem
Presses a UI button
Notify of necessary skip
Plays new track
User
UI
Audio Subsystem

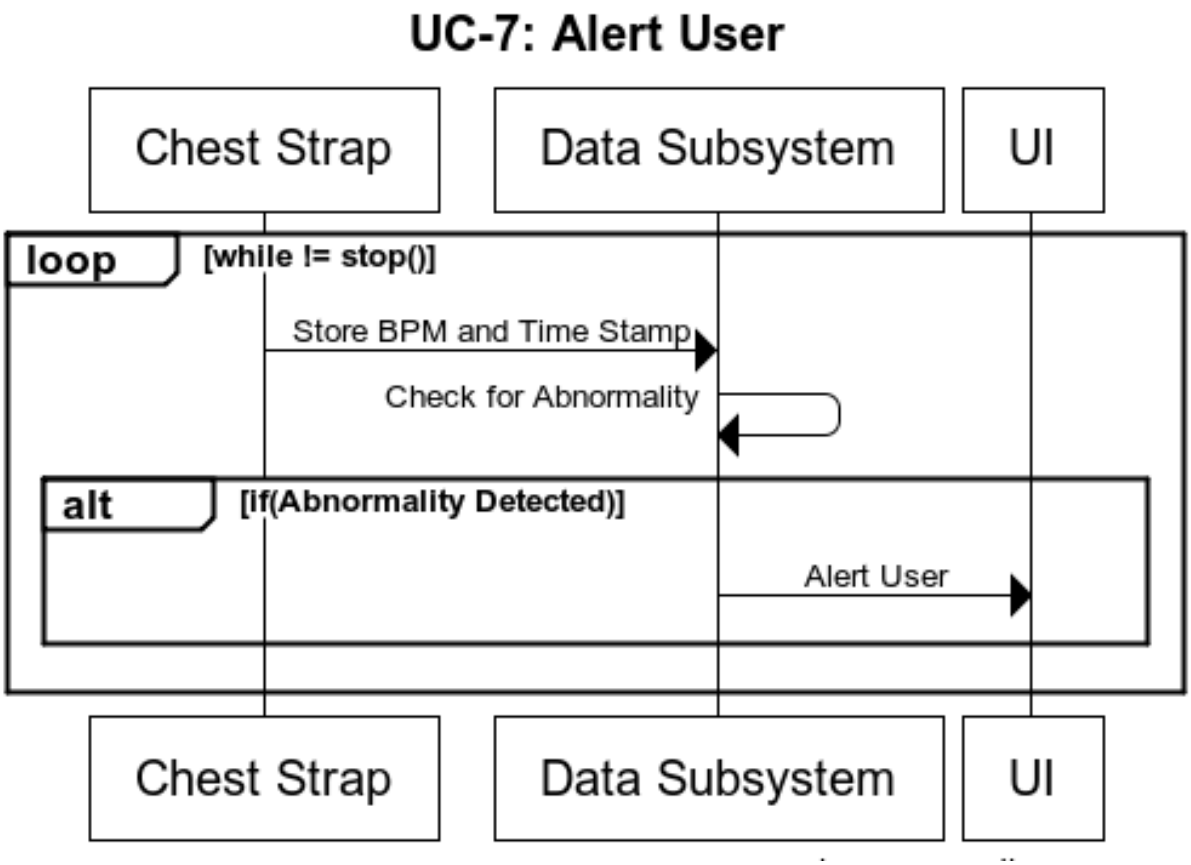

UC-4: Toggle Playback
User
UI
Audio Subsystem
alt
[Pause]
Presses "Pause" button
Notify of necessary pause
Pauses current track
[Play]
Presses "Play" button
Notify of necessary play
Plays current track
User
UI
Audio Subsystem

UC-5: displayStatistics



UC-6: Get Heart Rate





2.5.6 System Operation Contracts

OC-1: Enter Target Heart-rate

- **Precondition:** The application is open to the main screen and prompts the user for input.
- **Postcondition:** The system saves the input heart rate and will use it in selecting a song.

OC-2: Select Function("Skip Song")

- **Precondition:** The device is playing a song which needs to be changed.
- **Postcondition:** A different song is being played at the same rate at which the previous song was playing.

OC-3: Select Function("Toggle Playback")

- **Precondition:** The system is currently playing a song

- **Postcondition:** If the system was playing a song, it stops playing the song and recording the data from the workout. If the system is paused, toggling the playback will cause the system to start playing a song and recording data from the workout.

OC-4: Select Function("Display Statistics")

- **Precondition:** The user has either finished his workout or is in the middle of his workout and would like to see his statistics.
- **Postcondition:** The device retrieves the data from the databases, organizes it, and presents it to the user in the form of charts and tables.

OC-5: Select Function("Display Heart Rate")

- **Precondition:** The device should already be monitoring the user's heart rate.
- **Postcondition:** The current heart rate is displayed on the screen of the Android application.

2.5.7 Mathematical Model

The selection of which track to play requires a mathematical model. At its simplest, this consists of selecting the track with the closest BPM, that is to say minimizing the difference in BPM:

$$min(\mid target_{BPM} - track_{BPM} \mid) \tag{2.1}$$

If time permits, this simple model can be replaced with a more complex model incorporating Machine Learning to learn which tracks are more effective than others at changing pulse.

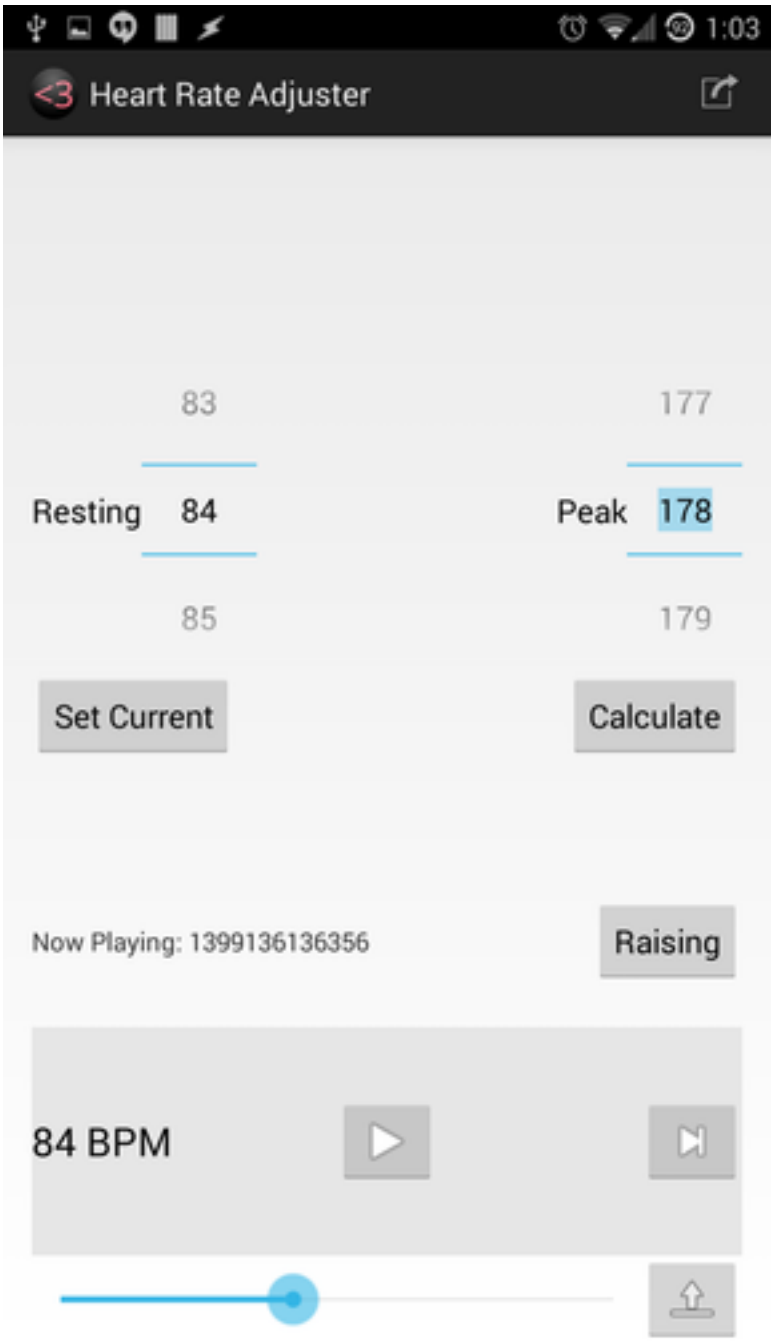
3 User Interface Specification

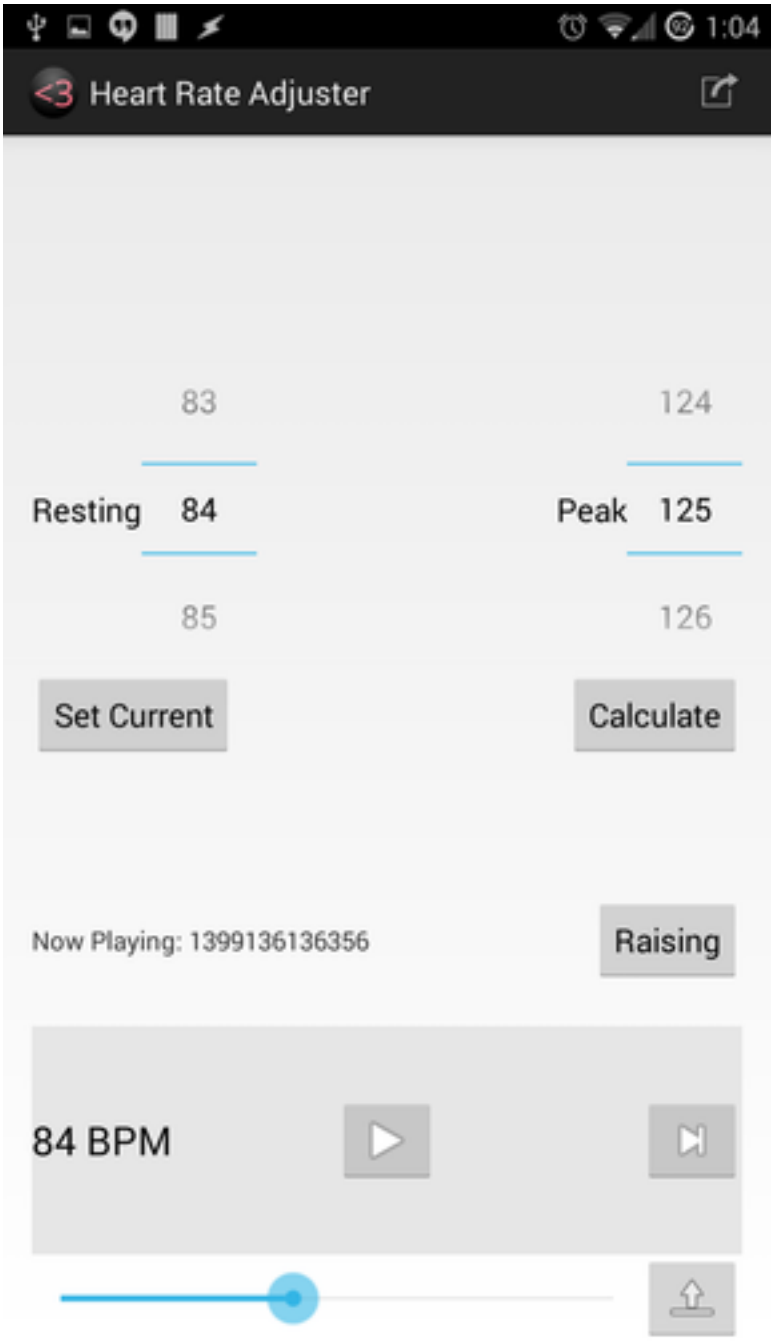
3.1 Preliminary Design

3.1.1 Use Case UC-1: Log Data

This Use Case doesn't have a User Interface component.

3.1.2 Use Case UC-2: Set Target Heart Rate



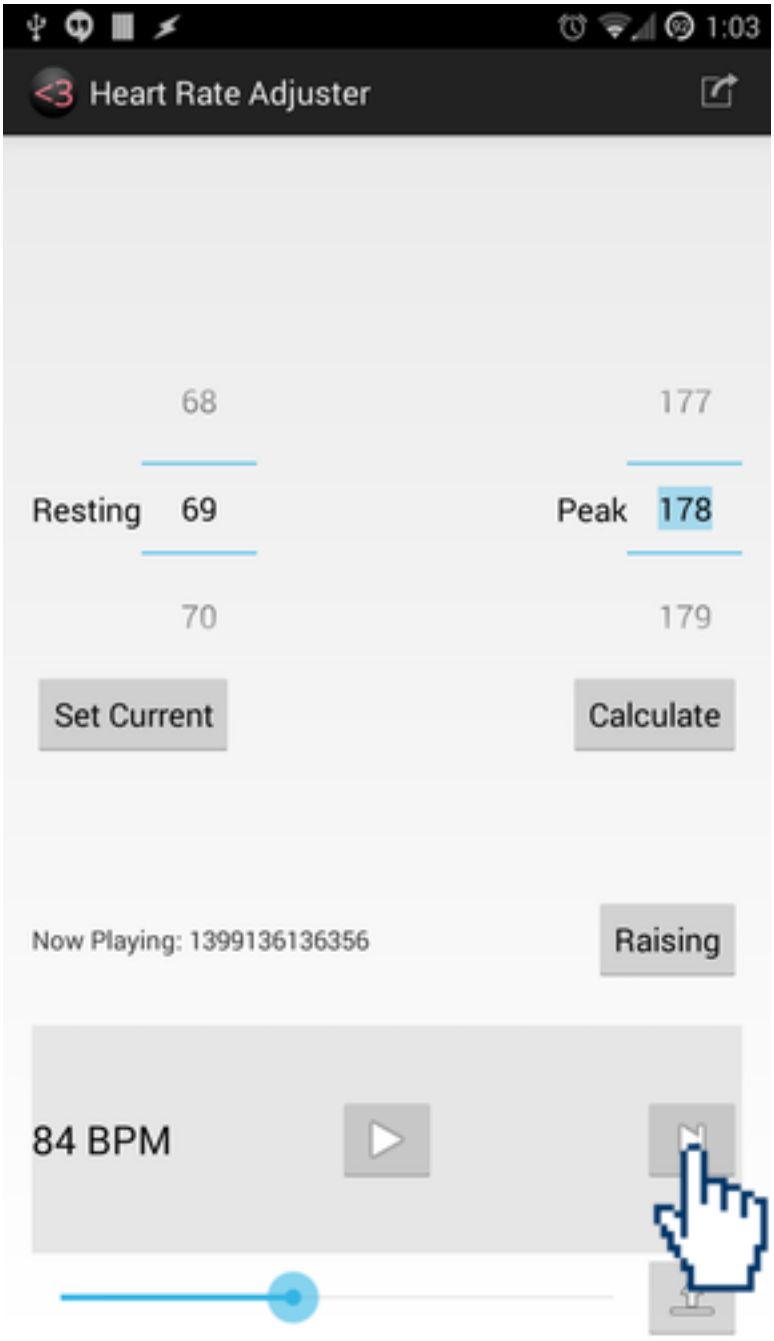


For this Use Case, the user's goal is to select a target heart rate for his workout. There are two ways a user can trigger this Use Case: modification of the currently active number selector, or pressing the toggle between Raise/Lower. As seen from the screenshots of our "home" screen above, we seek to minimize user effort in accomplishing his desired goal. The number selectors are standard Android UI components, so the user is presumably

already familiar with their functioning. Changing the target direction requires only one press, of the Raise/Lower button. The existence of this button means that, once a user has set their target preferences, they won't need to change the sliders much, reducing the effort of selecting numbers.

Once the user has selected a number, the system uses it in playback.

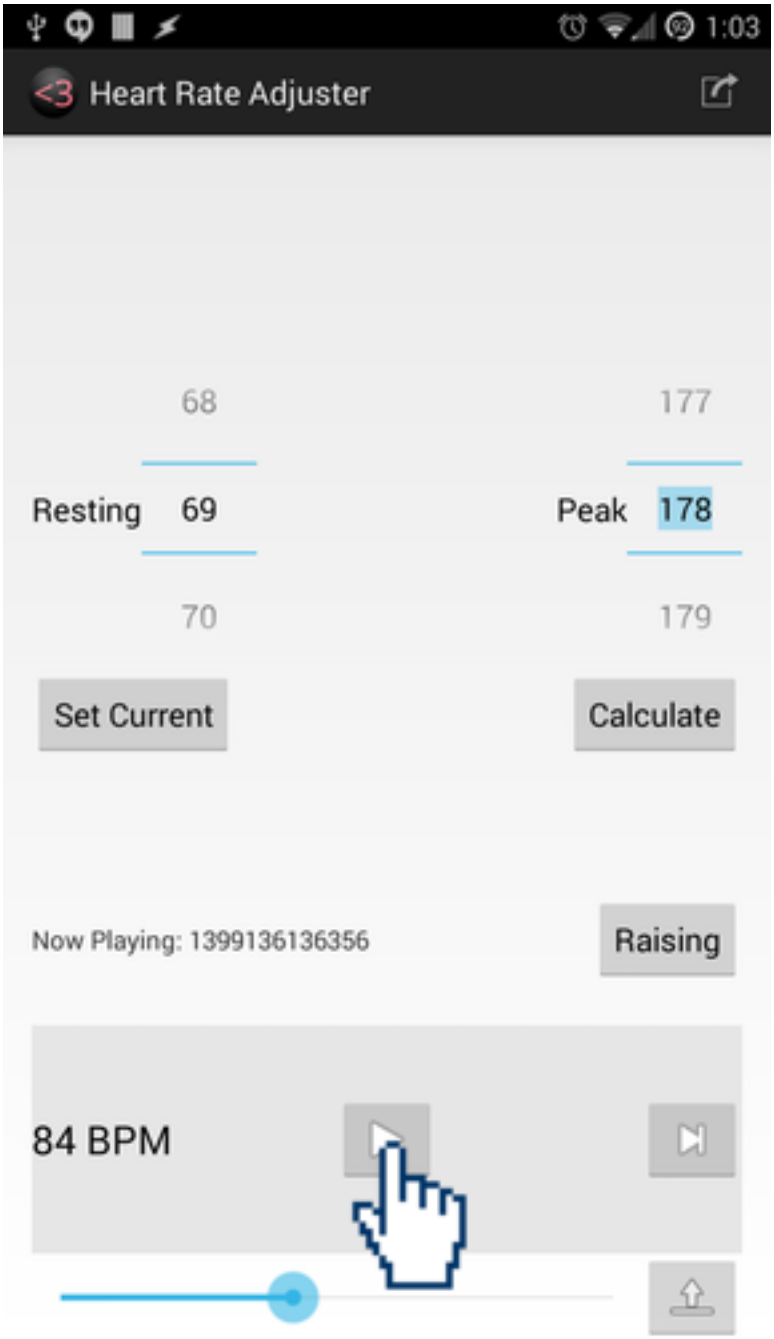
3.1.3 Use Case UC-3: Skip Track



To switch tracks is also very simple. It takes the user one simple tap to achieve his desired outcome. On the provided image of our concept interface, our application appears very similar to a mainstream music player. In the bottom right corner is the double-arrowed

fast forward button. The user taps this button to advance to another song, and then the system fulfills that request by running its algorithm and picking out another track from the user's music library. The "Current Track:" label will also be updated accordingly.

3.1.4 Use Case UC-4: Toggle Playback

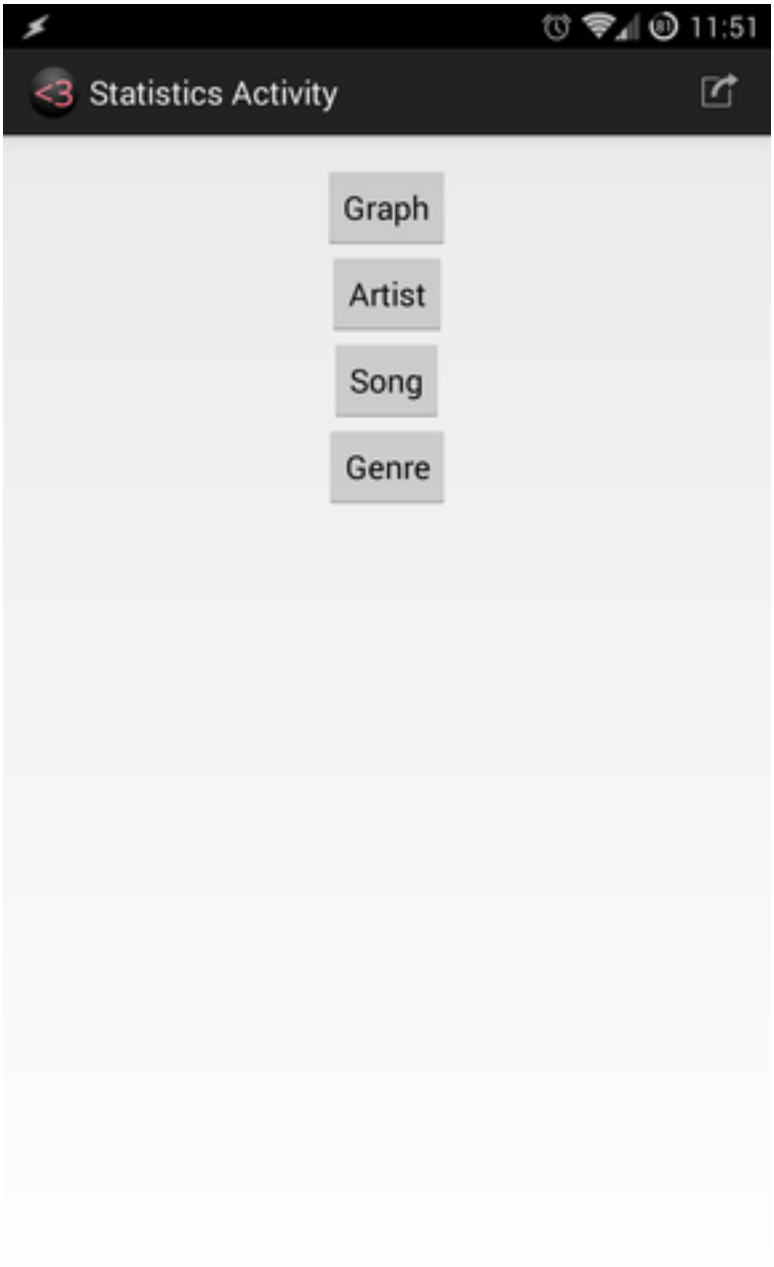


Use Case 4, togglePlayback also proves to be intuitive. Just like most music players, our application has a button located on the bottom center of the screen designed for the purpose of pausing the current song, or playing it, depending on the current state. The user

just needs a single tap on the universal play/pause button to achieve his goal of playing or pausing the song.

When this is done, if the system was previously playing, the system responds by stopping its collection of heart rate data, and freezing the screen in its current state. If the system were not previously playing, the system responds by beginning its collection of heart rate data, and beginning playback.

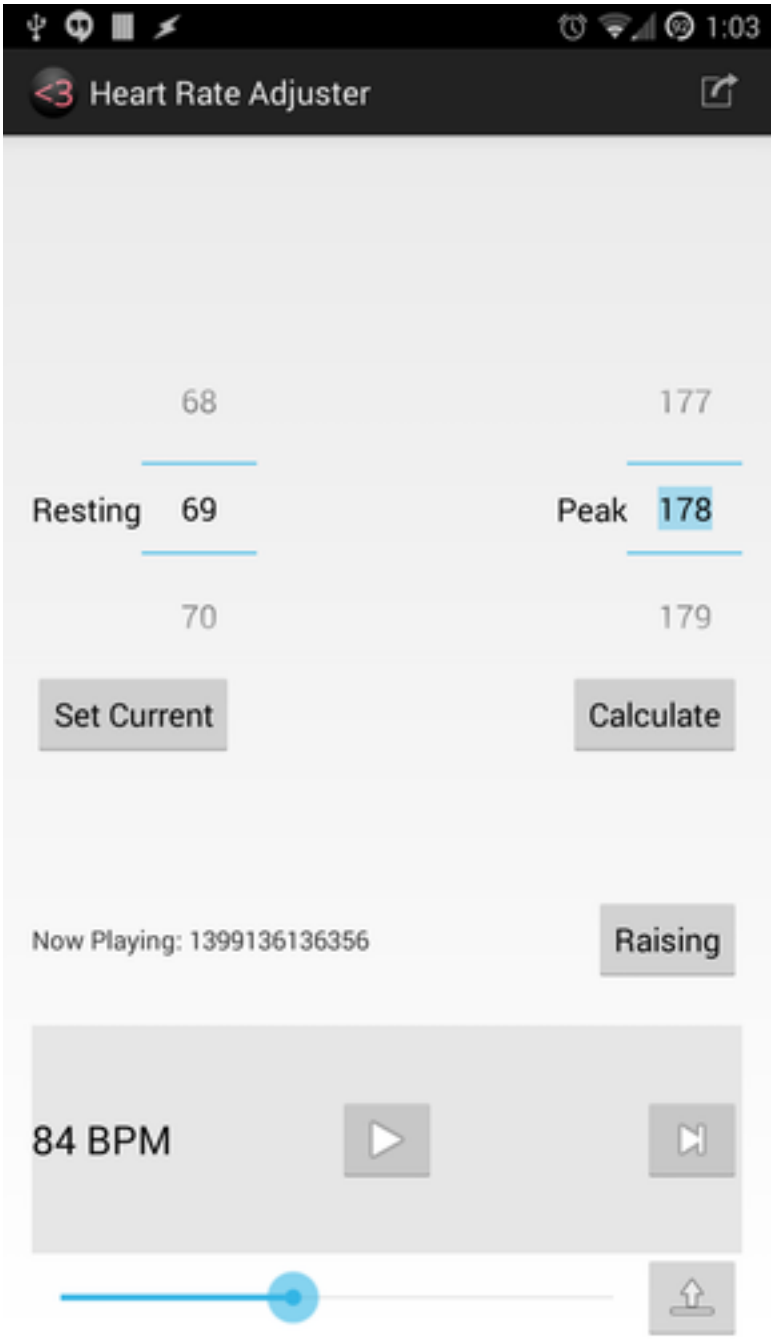
3.1.5 Use Case UC-5: Display Statistics



For use case 5, the user desires to view the statistics of his workout. To simplify the process for the user down to two clicks, we added a menu button in the top right corner of the screen. After pressing that menu button, a scroll-down menu with three options

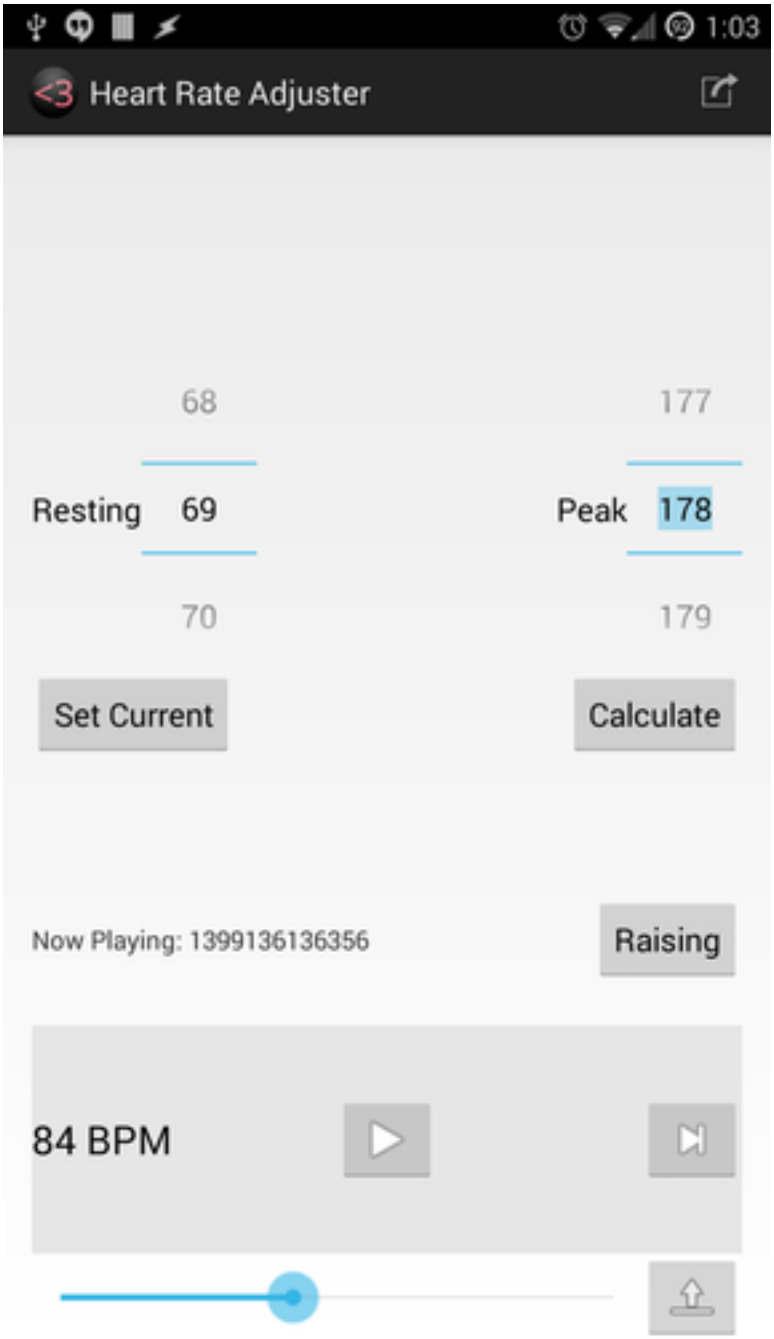
appears. The user needs to tap "Statistics" to bring up his workout information. The system is constantly logging the user data, and compiles a few useful graphs such as heart rate versus time.

3.1.6 Use Case UC-6: Get Heart Rate



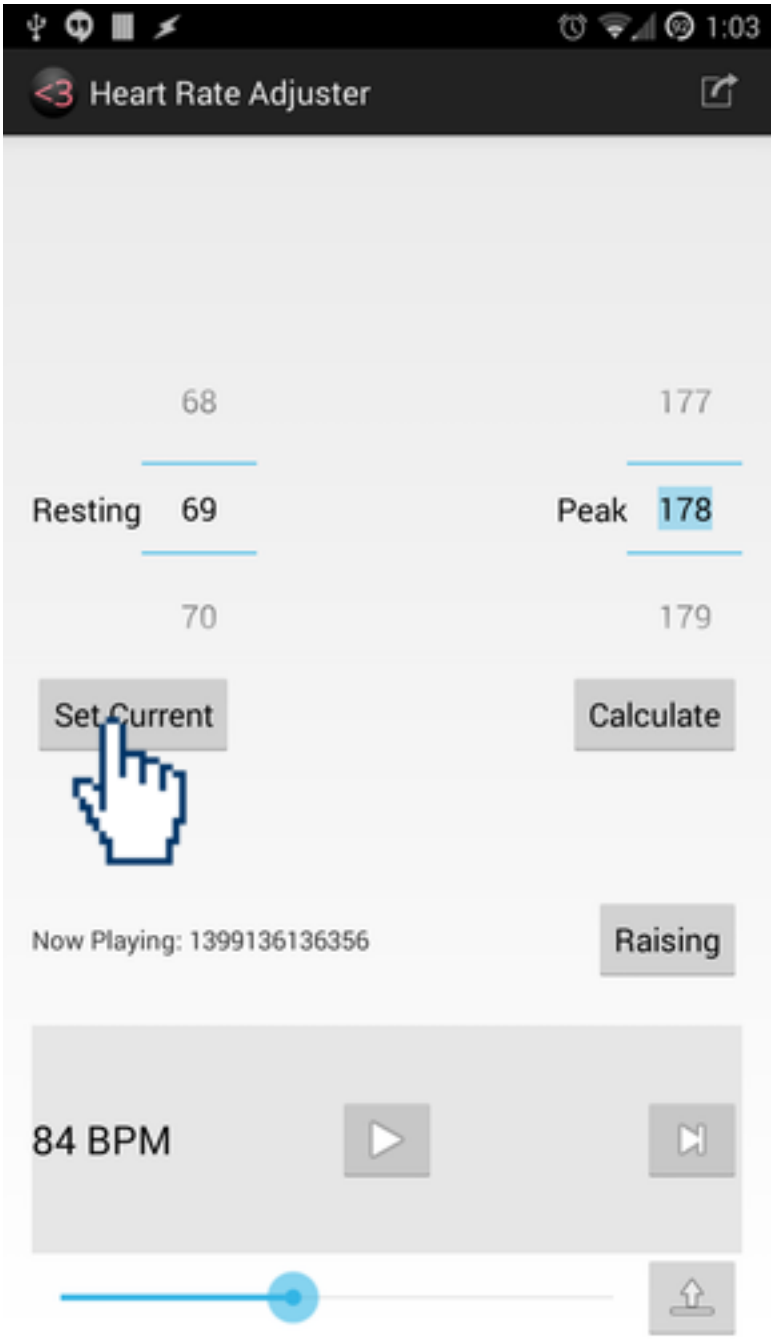
This Use Case requires no user interaction.
The UI is updated once a second with the current heart rate read from the chest strap.

3.1.7 Use Case UC-7: Provide Music Data



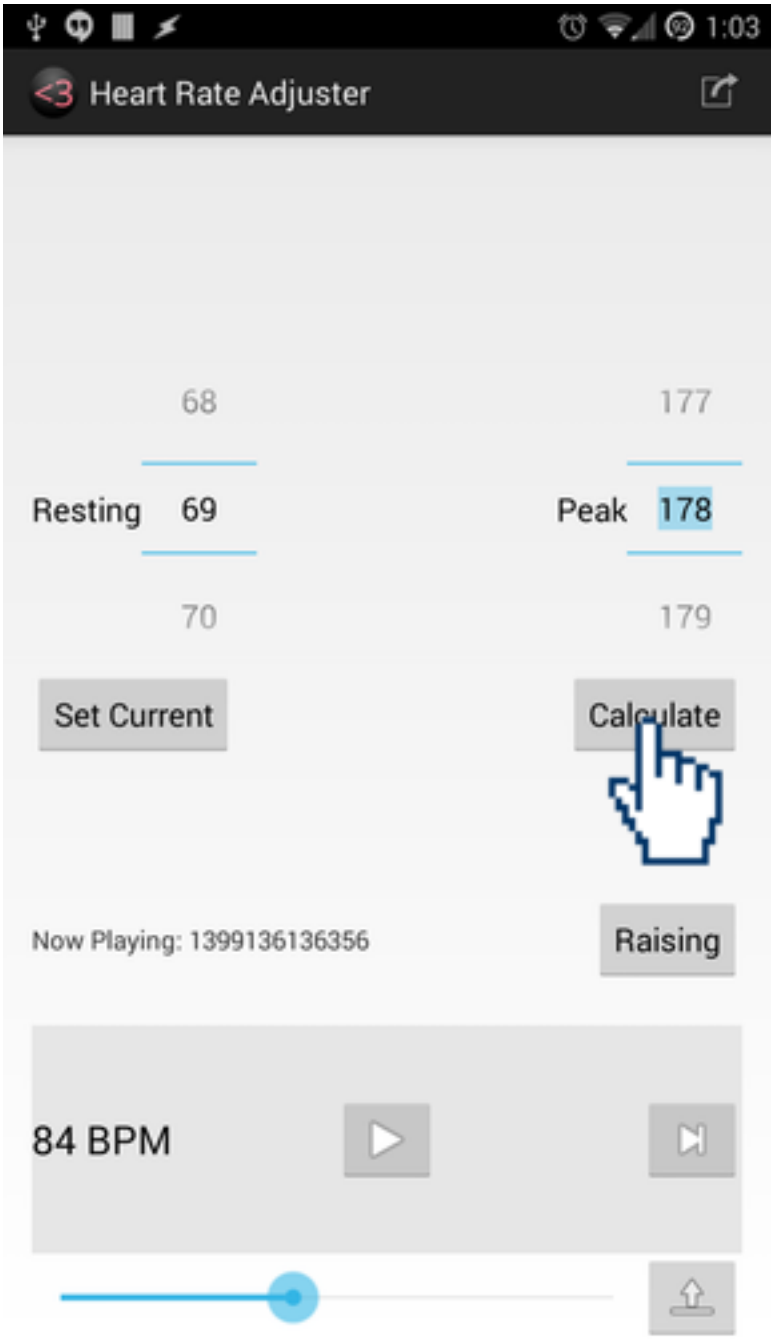
This Use Case requires no user interaction.
The UI is updated by the Audio subsystem with the title of the current track.

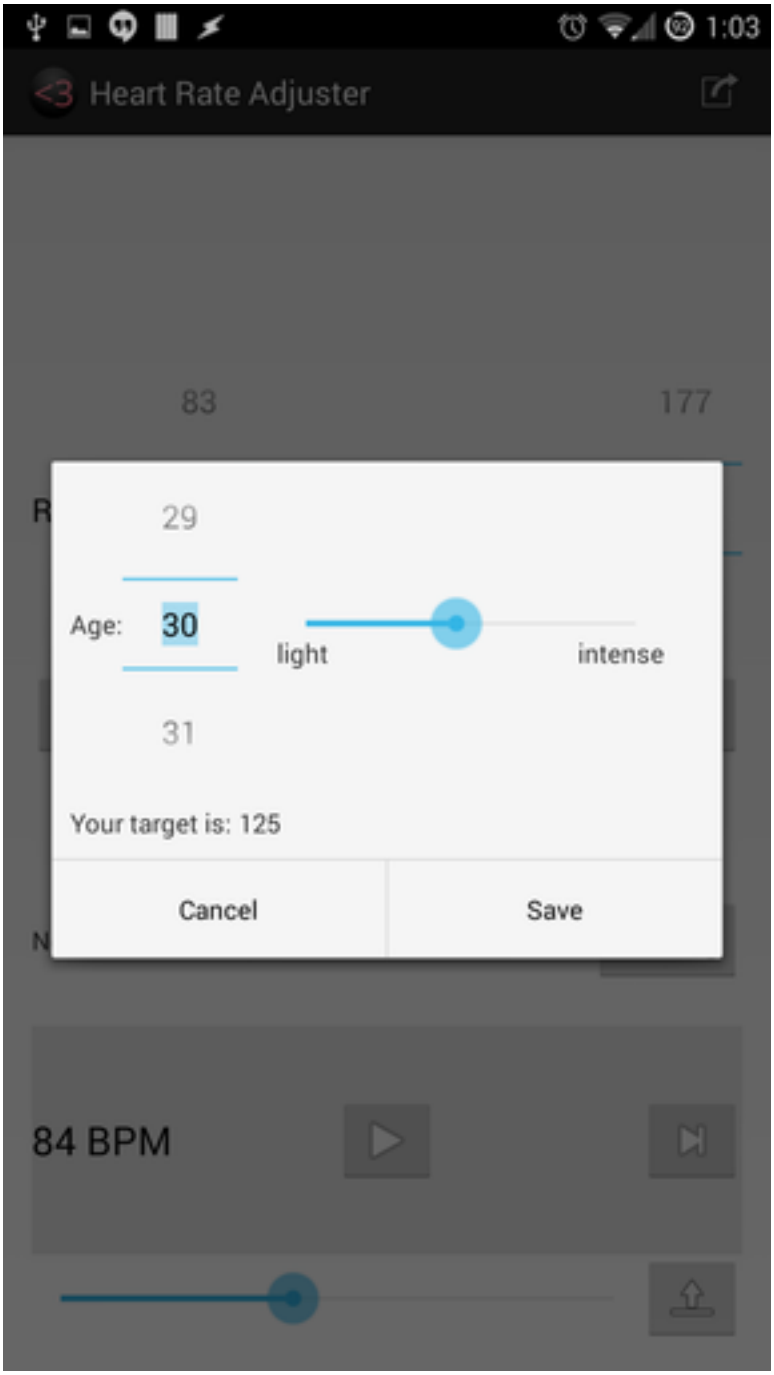
3.1.8 Use Case UC-8: Set Resting Heart Rate



For this use case, the user simply needs to press the "Set Current" button. The user should press this button at the beginning of his/her workout. The system will then retrieve the user's current heart rate and set it as the user's resting heart rate.

3.1.9 Use Case UC-9: Calculate Peak Heart Rate





For this use case, the user must first press the button labeled "Calculate". They will then be shown a pop-up where they may enter their age, and level of activity. Age is chosen similarly to selecting a peak heart rate manually, and level of activity is chosen on a default android scrollbar between light and intense. The system then uses this information to calculate a recommended target heart rate.

3.2 Effort Estimation

First we need to calculate the Use Case Points (UCP).

$$UCP = UUCP * TCF * ECF \tag{3.1}$$

Where Unadjusted Use Case Points (UUCPs) are computed as a sum of these two components:

1. The Unadjusted Actor Weight (UAW), based on the combined complexity of all the actors in all the use cases.

2. The Unadjusted use Case Weight (UUCW), based on the total number of activities (or steps) contained in all the use case scenarios.

Unadjusted Actor Weight (UAW) and Unadjusted Use Case Weight (UUCW)

Actor	Complexity	Weight
Users	Complex	3
Mobile App	Average	2

$$UAW = 3 + 2 = 5 \tag{3.2}$$

Now we reference the Use Case table from 2.5.1 to calculate the UUCW.

$$UUCW = 37 + 37 + 21 + 17 + 25 + 19 + 21 + 11 + 12 = 200 \tag{3.3}$$

There the UUCP is:

$$UUCP = 5 + 200 = 205 \tag{3.4}$$

Technical Complexity Factor (TCF)-Nonfunctional Requirements

Below is a table of Technical complexity factors and their weights.

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Friendly interface that the user understands	2	1	2
T2	Internal processing of heart beat data to music and adjusting BPM shouldn't be too complex	2	3	6
T3	Good Performance	1	2	2
T4	Security is a minor concern	1	2	2
T5	No direct access to third parties	2	3	6
T6	Ease of use is very important	3	3	9
	Technical Factor Total (TFT)			27

And $TCF = C1 + C2 \times TFT$, and $C1 = 0.6, C2 = 0.01$, so

$$TCF = 0.6 + 0.01 * 27 = 0.87 \tag{3.5}$$

Environment Complexity Factor (ECF)

The environmental factors measure the experience level of the people on the project and the stability of the project.

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor
E1	Mostly beginners at UML based development	1.5	1	1.5
E2	Decent familiarity with application problem	0.5	3	1.5
E3	Quite knowledgable about the Object-Oriented approach	1.5	3	4.5
E4	Somewhat motivated about the problem	1	2	2
E5	Progamming language proficiency	2	3	6
	Environmental Factor Total (EFT)			15.5

Here is the formula to calculate ECF

$$ECF = C1 + C2 * EFT \tag{3.6}$$

Where $C1 = 1.4, C2 = 0.03$. Therefore we calculate the ECF.

$$ECF = 1.4 + (-0.03 * 15) = 0.965 \tag{3.7}$$

So we calculate the final UCP:

$$UCP = 205 * 0.87 * 0.965 = 172.11 \tag{3.8}$$

If we assume that productivity factor is 28 hours per user case point. The effort estimation would be 4,189.

4 Domain Model

4.1 Concept Definitions

Responsibility	Type	Concept
Pairing/communicating with HRM	D	HRM manager
Retrieve logged data	D	log retriever
Musical Playback	D	music playerbacker
Logging tracks as they are played	D	track logger
Queue Next Track(s)	D	track queuer
Listen for user input	D	general UI
Recommend target heart rate	D	peak calculator
Set the user's resting heart rate	D	rest setter
Alert the user in case of danger	D	user alerter
Graphically displaying music information	D	playback view
Graphically displaying heart rate info	D	heart beat view
Graphically displaying current workout data	D	workout view
Graphically displaying previous workout data	D	history view
Data store for workout data	K	workout store
Data store for music metadata	K	metadata store
Data store for music files	K	music store

4.2 Association Definitions

Concept Pair	Association Description	Association Name
music playerbacker ↔ metadata store	music playerbacker retrieves information about the current track from metadata store	data retrieval
history view ↔ workout store	history view retrieves data about previous workouts from the workout store	data retrieval
track logger ↔ music playerbacker	tracks played by music playerbacker are logged by track logger	data logging
music playerbacker ↔ track queuer	music playerbacker retrieves the next track from the track queuer	data retrieval
music playerbacker ↔ playback view	playback view displays information based on the data in music playerbacker	human data interface
rest setter ↔ HRM manager	HRM manager retrieves user's current heart rate to set as resting	human data interface
user alerter ↔ HRM manager	HRM manager retrieves user's current heart rate and activates user alerter if in dangerous levels	human data interface
hrm manager ↔ general UI	general UI pairs and reports hrm status based on hrm manager	human data interface
heart beat view ↔ hrm manager	retrieves and displays heart rate data from the hrm manager	human data interface
log retriever ↔ workout store	log retriever fetches logs from the workout store and barks at the mailman	data logging
music playerbacker ↔ music store	music playerbacker plays songs from the music store	data retrieval

4.3 Attribute Definitions

Concept	Attribute	Attribute Definition
HRM manager	data logging	Data logging has to with the storage or retrieval of logged data or the logging of data.
log retriever		
track logger		
music playerbacker	human data interface	Human data interfaces deal with the interaction between the user and the data.
track queuer		
rest setter		
peak calculator		
general UI		
user alerter		
playback view		
heart beat view		
workout view		
history view		
workout store	data storage	Data storage deals with the storage of the data.
metadata store		
music store		

4.4 Traceability Matrix

	HRM manager	log retriever	track logger	music playerbacker	track queuer	general UI	playback view	heart beat view	workout view	history view	workout store	metadata store	music store	rest setter	peak calculator	user alerter
UC-1	X			X	X	X	X				X	X				
UC-2	X			X	X	X	X				X	X				
UC-3			X	X	X	X	X					X	X			
UC-4	X			X		X	X	X	X		X					
UC-5		X				X				X	X					
UC-6	X					X		X	X		X					
UC-7	X					X										X
UC-8					X									X		
UC-9	X				X										X	

5 Plan of Work

Looking ahead at some of the due dates in the near future, our group will probably be mixing documentation with development during the course of March. Up until now, we have been primarily working on laying the foundations of our project. We have been examining customer and system requirements and extracting specifications from them. We have been envisioning preliminary design as well as analyzing user estimation. We have derived our domain and mathematical models. In the upcoming weeks, we will be working on some UML diagrams to make our lives a lot easier when we begin to code. We will complete the sequence and class diagrams to help us understand exactly which attributes and classes we need to account for. We will also design our system architecture to further our understanding on how each of the components of our system interface and communicate with each other. We will identify any data algorithms or structures needed to store the information generated by the phone and the heart rate monitor. Finally, we will continue to revise our designs and implementations from our previous iteration and construct tests to ensure that our system works as planned. By doing this preliminary design and analysis in conjunction with the upcoming deliverables, we will develop a solid understanding of what pieces of software actually need to be written. We will have a snapshot of our system and hopefully foresee some of the problems that might have occurred had we rushed straight into coding. The included Gantt chart serves as a visual roadmap of our project plan.

5.1 Gantt Chart

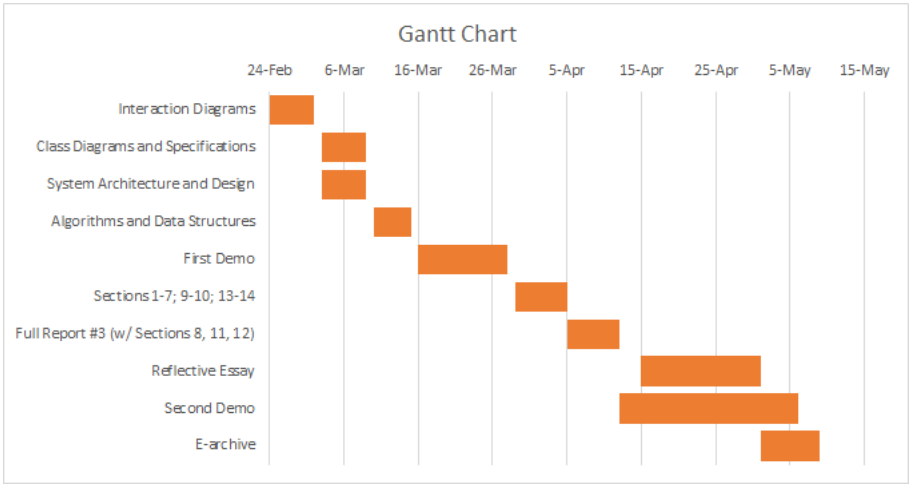


Figure 5.1: Our Gantt Chart details our plan of work in the upcoming weeks leading up the second report, and also lists our hopes for the second iteration.

5.2 Product Ownership

Our team will be divided into three smaller sub-teams of two individuals each, the pairings listed below. Each sub-team will be responsible for developing a specific sub-product during the bulk of their time. Upon completion of a significant portion of work, they will provide a brief description of their activity before they push it to our Github repository. In addition to documentation, they will also include the necessary UML diagrams, charts, algorithms, and source code. Every week, or at least once before each deliverable, we will meet together for 1-3 hours during a timeframe determined by a combination of GroupMe and When2meet. During the meeting, we will have a specific agenda that primarily involves the week's progress and upcoming deliverable. Our discussion will probably be centered along the following questions: 1) What did you work on this past week? 2) What do you plan on working on next week? 3) Are there any revisions that need to be made to the project? Every week, a team member will take the lead for the next deliverable to ensure that everything is on time.

- Revan and Tae-Min will work on the general user interface of the mobile application. They will design the layout and basic functionality of the Android app, and set up the communication protocol between the heart rate monitor and the phone.
- Kenny and Samani will work on the music aspect of the mobile application. They will study music playback features such as track selection and queuing to provide

customers with a seamless music player experience. They will also be responsible for the nuances of audio playback such as crossfading transitions and tempo adjustments.

- Jon and Nikhil will work on the data logging faculties that are necessary to provide the user with feedback about his/her workout. They will work with a server that captures heart rate and music data from the phone and possibly directly from the heart rate monitor. They will determine how to process this information on the server and create customizable graphical displays for user's to view.

5.3 Breakdown of Responsibilities

	Past	**Present**	**Future**
Jonathan	Problem Statement, Enumerated Functional Requirements, Preliminary Design, References	Plan of Work	Project Management, Interaction Diagrams, Class Diagrams
Kenny	User Effort Estimation	Domain Analysis	System Architecture and System Design
Nikhil	Enumerated Nonfunctional Requirements, Use Cases (Casual Description, Traceability Matrix, Fully-Dressed Description)	System Operation Contracts	Project Management, Interaction Diagrams, Class Diagrams
Revan	On-Screen Appearance Requirements, Use Case Diagrams	Mathematical Model	Class Diagrams, System Architecture and System Design
Samani	System Sequence Diagrams	Domain Analysis	System Architecture and System Design
Tae-min	Glossary of Terms, Stakeholders, Actors, Goals	System Operation Contracts	Algorithms and Data Structures

6 Interaction Diagrams

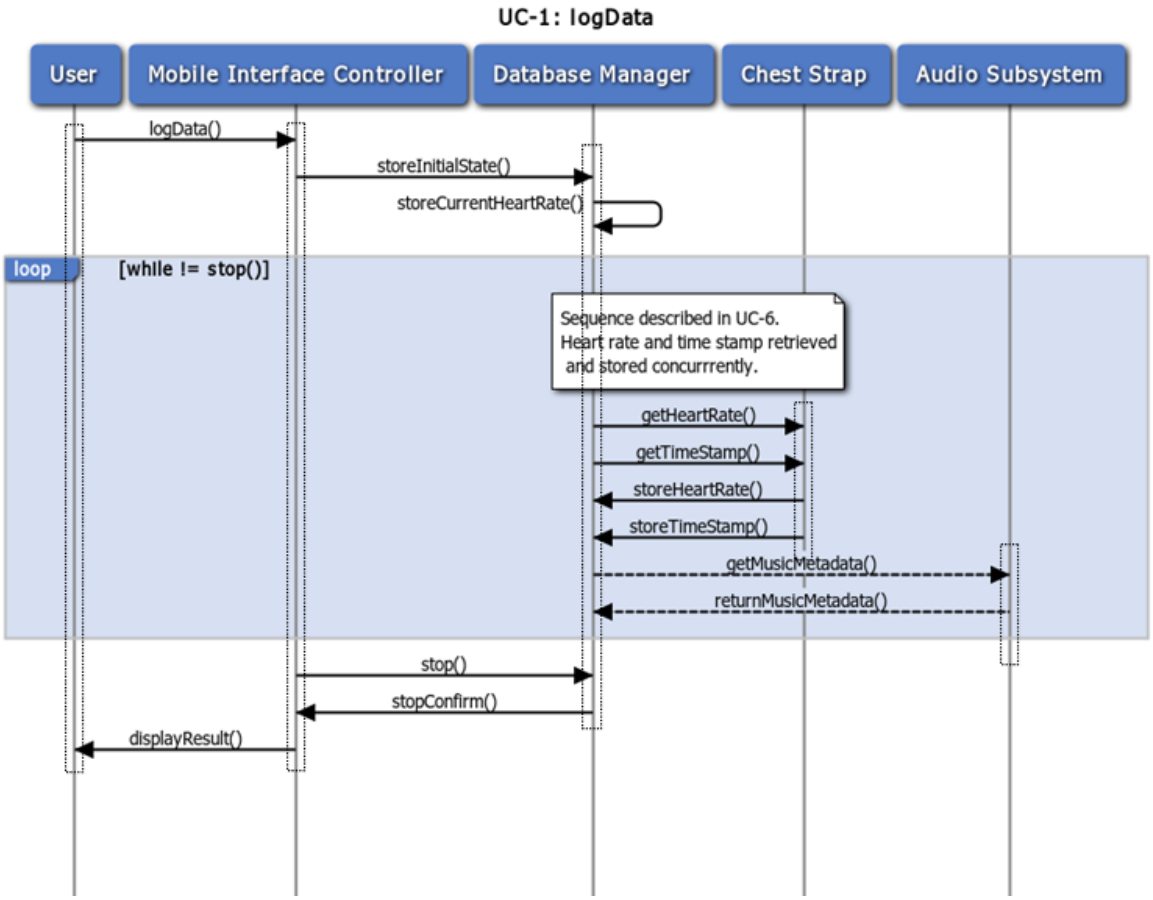


Figure 6.1: Interaction diagram for the logData() use case.

For our first use case, we wish to record information about the user's heart rate as well as some additional information. This includes a time stamp of when the heart rate data was recorded, and could also include some of the song metadata such as artist, album, or genre. For our main success scenario, the user interacts with the UI which communicates with the Database Manager. As long as user does not tell the UI to stop recording, the Database Manager will continually ask the Chest Strap for BPM Data and a time stamp for every piece of data received. When the user tells the UI to stop, we break out of our loop, and

the Database Manager returns to the UI and displays updated information. Alternate scenarios could occur when the Chest Strap is not functioning correctly and reports an error in measurement, and when this occurs up to 10 times, the Database Manager stops recording. Another alternate scenario would be if the system detects an abnormal heart rate in the user, and alerts the user to stop his workout. This process is described more thoroughly in the Algorithms section.

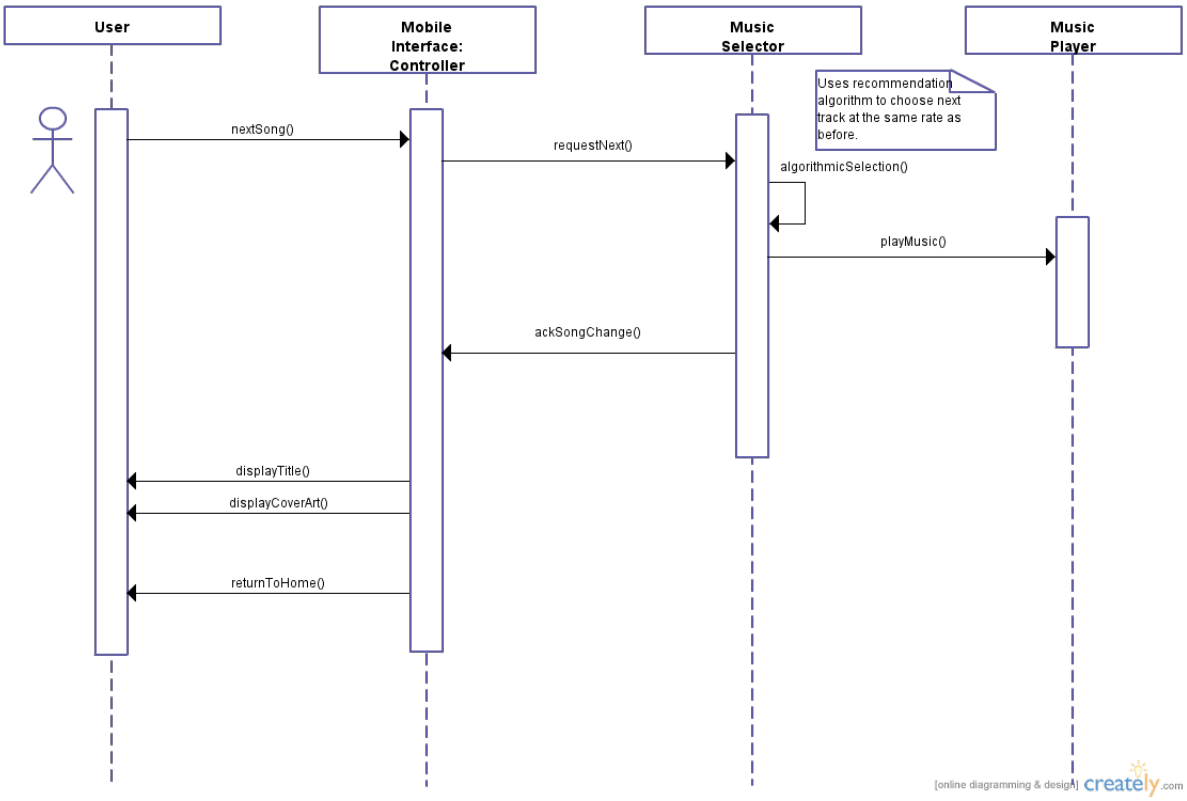


Figure 6.2: Interaction diagram for getNextSong()

For our third use case, the user's goal is to listen to another song. Similar to the "skip" button on most standard music players, we included a double fast-forward arrow for users' convenience. When pressed, the controller immediately contacts the database manager. The database manager selects another track based on its song-selection algorithm for the music player to play. (If the user is currently trying to change his/her heart rate, the database manager will take that into count and select a song of a similar speed.) It also returns the cover art and new song title for the mobile interface to display.

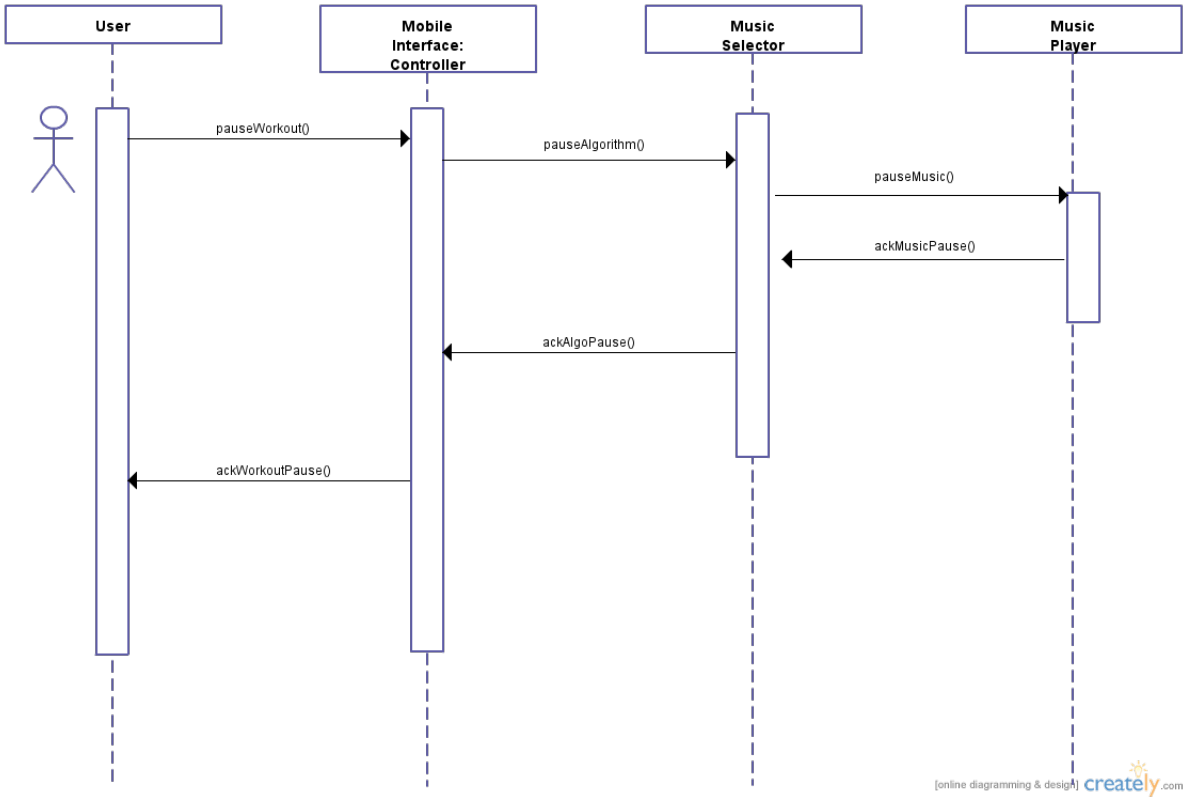


Figure 6.3: Interaction diagram for togglePlayback()

Our togglePlayback sequence is fairly simple. The user initiates the request by tapping the play/pause button. Then the controller informs the database manager to store the current state settings for future use, and the database manager proceeds to stop recording data and allows the music player to stop the song. The mobile display also updates accordingly.

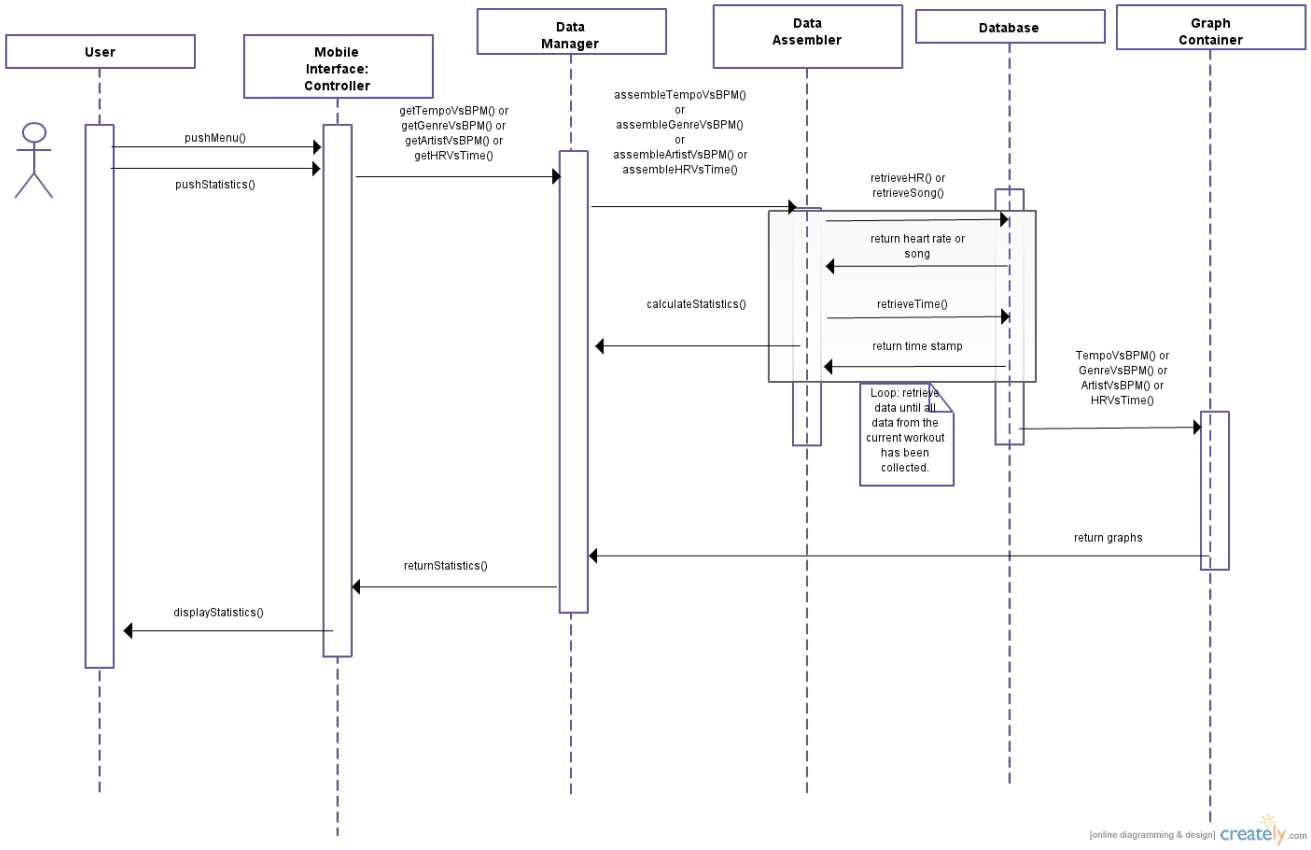


Figure 6.4: Interaction diagram for displayStatistics()

In UC-5, displayStatistics, the user begins with two button presses to achieve their goal. First, they bring up the menu button in the top right hand corner and then press "Statistics" from the dropdown. The controller passes this request to the database manager, which quickly retrieves and updates the data. Then, it calculates statistics, creates some graphs, and then returns the output to the controller to display. (Note that the user must select from the options available in order to view his workout history.)

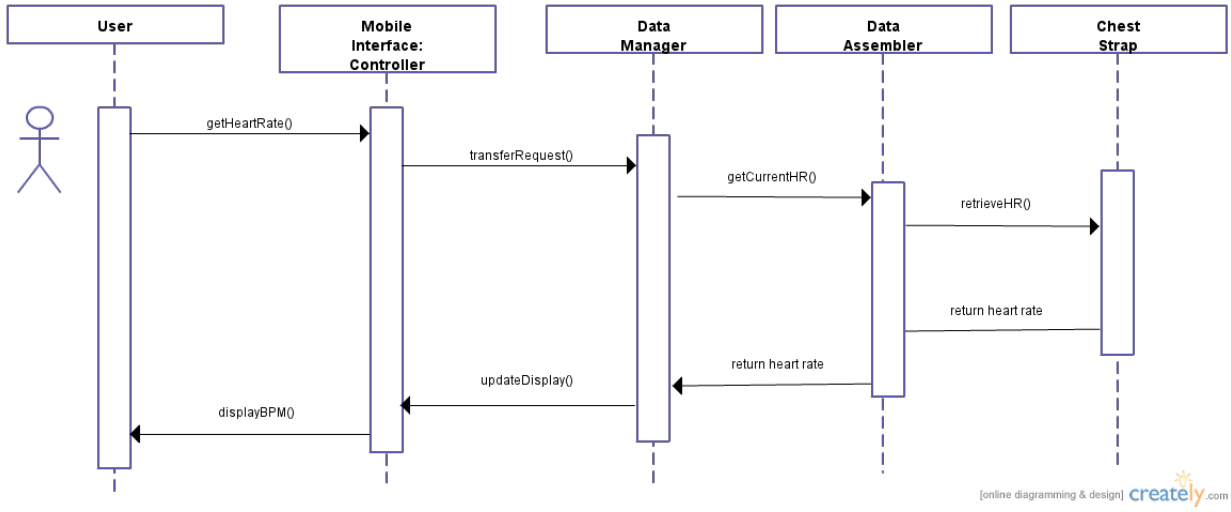


Figure 6.5: Interaction diagram for getHeartRate()

UC-6, getHeartRate is very similar to UC-5. This use case is also applicable for UC-1, because the Heart Rate is important, needing to be determined continuously. The controller takes the request from the user and passes it to the Data Manager to handle. The database manager then takes the current BPM value from the Heart Rate Monitor and updates the value for the mobile interface to display to the user.

6.1 Alternate Scenarios

Alternate scenarios could occur mainly during the getStatistics() use case. Here, the user can select which particular graphs or data he may want to see. In the original diagram, dedicated Data Assembler and Graph Container objects were created to handle the organization and display of all statistics. These objects greatly simplify the displayStatistics() case because all the calculations are done outside of the database. However, this is not the only way to solve the case, with the alternative solution being that the statistics are generated directly by the database. Advantages of this are that there is no need to transfer the data between two objects and worry about the communication scheme between them. Also, the time needed to flesh out an entirely new class to handle the calculations would be eliminated. Although the statistics-generation functionality can be contained entirely within the database, we implemented it through the Data Assembler and Graph Container combination. This decision was made to keep consistency with the High Cohesion and Low Coupling Principles. If we hadn't made the separation, then the database would have too many responsibilities to take care of. By creating dedicated Assembling and Graphing objects, the statistics can be fully fleshed out and managed without having to worry about other responsibilities.

6.2 Design Patterns

This project was developed through a mainly object-oriented approach. We were able to boil our system down to a few, distinct actors, and an object-oriented approach seemed appropriate to model our system. As evidenced by the interaction diagrams, there are five main actors: the user, the mobile interface, the data manager, the music player, and the heart rate monitor. Each of these actors has a specific set of actions that they can perform, and these actions are not very tightly coupled to those of other actions in the system. For example, the music player is designated to solely play the music on the device. It does not have access to the music algorithm or the heart rate; its sole job is to respond to requests about playing or pausing the music. Similarly, the data manager is solely responsible for manipulating data and handling requests. These characteristics are prime examples of the High Cohesion principles, because each actor in the system has its own specific tasks, and the respective actors are designed to that their tasks are carried out very well. We acknowledge that some of the actors are more important than the others, such as the data manager and the mobile interface, but this is out of necessity. The user interacts directly with the interface, and the interface talks directly to the database in most cases. The other objects in the system are more supplementary in the sense that they carry out the commands given to them by the database-mobile-interface pair. Although there is some communication between the different objects, the communication has been designed such that each method call is specific, efficient, and effective. This cuts back on unnecessary communication between the objects and allows for the system to be optimized. The Low Coupling Principle requires that objects should "not take on too many communication responsibilities", and our design fulfills the requirement because we have minimized the number of interactions to just the necessary ones. All in all, our object-oriented design encompasses aspects from both the High Cohesion and Low Coupling principles, and creates and effective solution to the heart rate monitoring problem.

6.3 Assignment of Responsibilities

A prime example can be found in UC-4, togglePlayback. Each object submits a request to the next object in line before reaching the Music Player, which is supposed to fulfill the "pause" function. At this point, the interactions start coming back. It is clearly seen that each object in this example transmits at most two messages, and no object performs more than a single computation. A similar theme exists in UC-6, getHeartRate. Each object essentially sends one message and receives one message. Furthermore, each object does not need to fulfill more than two active responsibilities. We believed that by distributing the workload for each object through the High Cohesion Principle and the Low Coupling Principle, we would be reaching the best balance. The Expert Doer Principle was not followed as closely because the communication links of the objects we used are a bit longer. In our design, the "one who knows" often passes on the knowledge to another object that

"needs to know" before the task is actually fulfilled. For instance, the Database Manager often causes the Controller to update the display and show the user rather than directly communicating with the user. Basically, our Controller and Database Manager are both extremely important, so oftentimes, they both end up with most of the implementation details.

7 Class Diagram and Interface Specification

7.1 Class Diagram

7.1.1 Class Diagram for Data Management

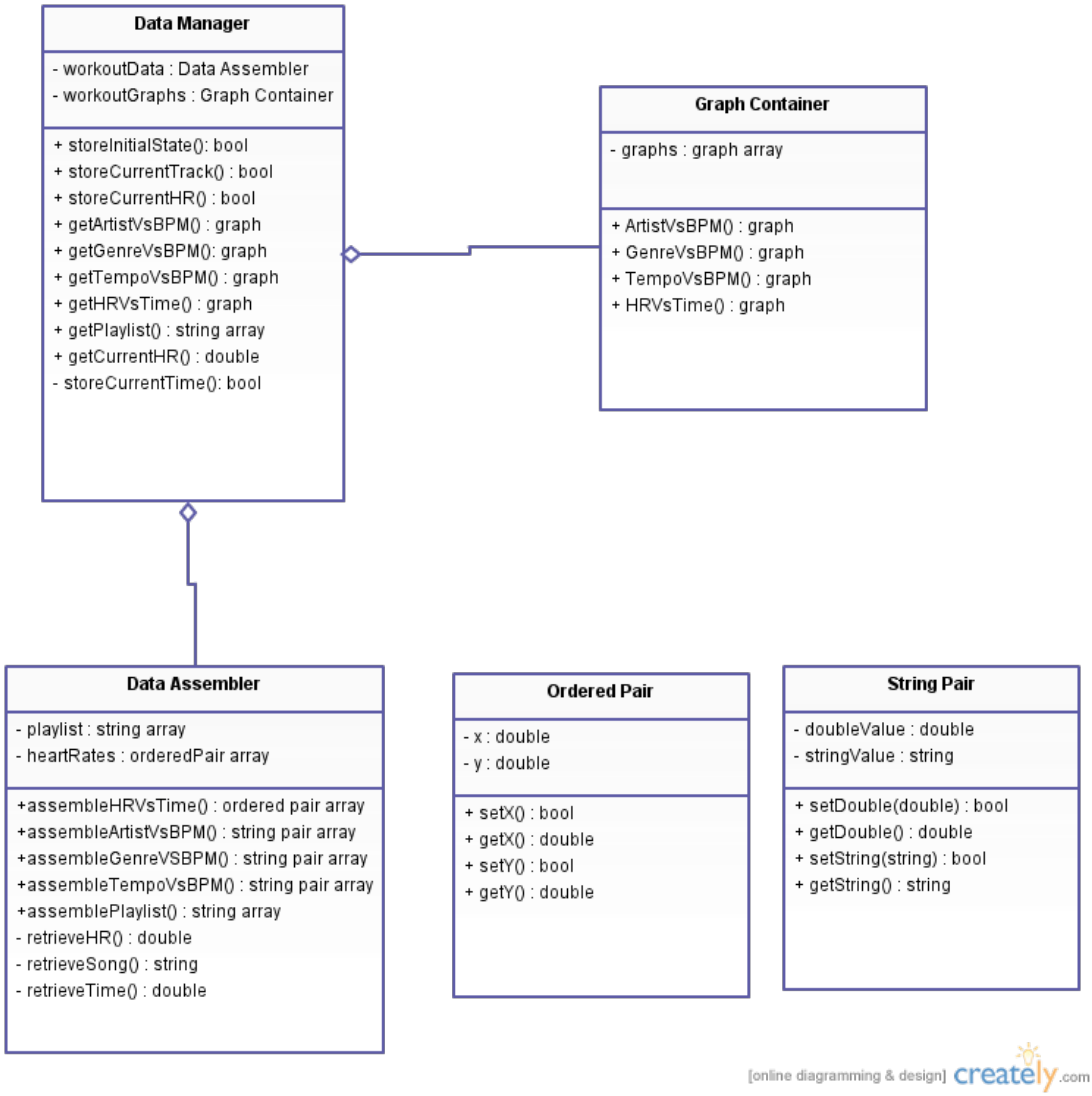


Figure 7.1: Class relationship for managing data

7.2 Data Types and Operation Signatures

Data Manager

Variables:

- workoutData: Data Assembler. This is a Data Assembler object which stores the data points retrieved from storage. It also stores the songs and their metadata.
- workoutGraphs : Graph Container. This is a Graph Container object which stores the different graphs that were requested by the User Interface.

Functions:

- storeInitialState(double initialRate, double targetRate) : bool. This function stores the initial state of the system, which is captured in the form of the user's initial heart rate and the target heart rate.
- storeCurrentTrack(string song) : bool. This function is used by the Music Module to store the current track being played. This information is later used for the playlist.
- storeCurrentHR(double heartRate) : bool. This function is used to store the current heart rate into the data storage. It is used by the User Interface while the user is working out with the system.
- getArtistVsBPM() : graph. This function will access the data stored in the Data Assembler and create a histogram displaying the artist who's songs were played most often at different BPMs. The Data Manager will select the appropriate graph from the Graph Container and return it to the UI.
- getGenreVsBPM() : graph. This function will create a histogram of the most frequent genres at different BPMs. The data will be accessed from the Data Assembler and plotted by the Graph Container. The final graph will be selected by the Data Manager from the Graph Container
- getTempoVsBPM() : graph. This function will return a graph of the music's tempo versus the user's BPM.
- getHRvsTime() : graph. This function will return a graph of the user's heart rate over time.
- getPlaylist(): string array. This function will return an array of the music that was played during the workout.
- storeCurrentTime() : bool. This is an auxiliary function which stores the current system time in the data storage every time storeCurrentHR() is called. It will associate the time with the current heart rate.

Data Assembler

Variables:

- playlist: string array. This variable will store the names of the songs that were played during the workout.
- heartRates: Ordered Pair array. This variable will store the retrieved data points in ascending order.

Functions:

- assembleHRVsTime() : ordered pair array. This function assembles the data for an HR vs Time graph.
- assembleArtistVsBPM() : string pair array. This function assembles the data for an Artist vs BPM table.
- assembleGenreVsBPM() : string pair array. This function assembles the data for a Genre vs BPM table.
- assembleTempoVsBPM() : string pair array. This function assembles the data for a Tempo vs BPM table.
- assemblePlaylist() : string array. This function assembles the music playlist.
- retrieveHR() : double. This function retrieves a heart rate from the data storage.
- retrieveTime() : double. This function retrieves a time from the data storage.
- retrieveSong() : string. This function retrieves a song name from data storage.

Graph Container

Variables:

- graphs : graph array. This variable contains an array of the different graphs that were requested by the User Interface.

Functions:

- ArtistVsBPM(Data Assembler workoutData) : graph. This function takes the names of the artists and heart rates stored in the Data Assembler and graphs them against each other.
- GenreVsBPM(Data Assembler workoutData) : graph. This function graphs the genre of the music versus the user's heart rate using the data from workoutData.
- TempoVsBPM(Data Assembler workoutData) : graph. This function graphs the tempo of the music versus the user's heart rate using the workoutData object.
- HRVsTime(Data Assembler workoutData) : graph. This function graphs the user's heart rate versus time using the workoutData object.

Ordered Pair

Variables:

- x : double. This variable stores the x-coordinate of the data point.
- y : double. This variable stores the y-coordinate of the data point.

Functions:

- setX(double newX) : bool. This function sets the value of the variable x.
- getX() : double. This function returns the current value of the variable x.
- setY(double newY) : bool. This function sets the value of the variable y.
- getY() : double. This function returns the current value of the variable y.

String Pair

Variables:

- doubleValue : double. This stores the double value of the pair
- stringValue : string. This stores the string value of the pair.

Functions:

- setDouble(double newDouble) : bool. This sets the double value of the pair
- getDouble() : double. This returns the double value of the piar
- setString() : double. This sets the string value of the pair.
- getString() : string. This returns the string value of the pair

Heart Rate Adjuster GUI

Variables:

- targetHeartRate: int. This variable will store the target heart rate for the user's workout. This variable is private but can be accessed and changed from other methods.
- restingHeartRate: int. This variable will store the user's initial resting heart rate before the workout begins. This variable is private but can be accessed and changed from other methods.
- isWorkingOut: boolean. This variable will store data on whether the application detects that the user is currently working out or not.
- isRecording: boolean. This variable will store data on whether the application is recording the user's heart rate or not.
- currBPM: int. This variable will store the BPM of the current track that is playing.
- isReady: boolean. This variable will store data on whether the system is ready to begin or not.

Functions:

- getRestingHeartRate(): int. This function will return the data stored in variable targetHeartRate.
- getTargetHeartrate(): int. This function will return the data stored in variable currentHeartRate.
- setTargetHeartRate(): void. This function will set the data stored in variable targetHeartRate to equal the given parameter.
- setRestingHeartRate(): void. This function will set the data stored in variable restingHeartRate to equal the given parameter.
- displayBPM(): int. This function will return the data stored in variable currBPM.
- setBPM(): void. This function will set the data stored in variable currBPM to the passed parameter.
- displayReady(): boolean. This function will return the data stored in variable isReady.
- startWorkout(): void. This function will initiate the workout and attempt to adjust the currentHeartRate toward the targetHeartRate.
- stopWorkout(): void. This function will stop the workout from continuing its functions.
- nextSong(): void. This function will skip the current track which is being played, and use the algorithm to play the next appropriate song.
- displayTitle(): String. This function will display the title of the currently playing track.
- checkRecording(): boolean. This function will return the data stored in the variable isReady.
- checkPlaying(): boolean. This function will return the data stored in the variable isREady.
- menu(): void. This function will display the menu for the application.
- getStatistics(): graph. This function will return graphs which contain statistics from the data manager.

Heart Rate Adjuster Hardware
Variables: • currentHeartRate: int. This variable will store data on the user's current heart rate, as measured by the hardware device.
Functions: • recordCurrentHeartRate(): int. This function will retrieve the current heart rate of the user as measured by the hardware device and update the variable currentHeartRate.

7.3 Traceability Matrix

	Class					
Domain Concepts	**Data Manager**	**Data Assembler**	**Graph Container**	**Ordered Pair**	**Heart Rate Monitor GUI**	**Heart Rate Monitor Hardware**
HRM Manager	X					X
Log Retriever	X					
Track Logger	X					
Music Playerbacker	X				X	
Track Queuer	X	X				
General UI	X	X			X	
Playback View	X	X	X		X	
Heart Beat View	X	X	X		X	
Workout View	X	X	X		X	
History View	X	X	X		X	
Workout Store	X	X		X		
Metadata Store	X	X		X		
Music Store	X	X		X		

From our domain concepts, we derived four classes: data manager, data assembler, graph container, and ordered pair. Our data manager is essentially involved with every domain. Its purpose is to log and manage various types of data, and store the packaged data other objects to retrieve. Essentially, the data manger acts as an intermediary in most steps, but only providing a minimal interface for modules so that data cannot be tampered with or seen, just used.

Next, our data assembler is charged with retrieving the appropriate data from the database and packaging it in a convenient form for usage. For instance, we can take songs and metadata from their storage locations and return playlists. We can also take our data and create ordered pairs for our graph container. Then, our graph container contains an array of the requested graphs, and it is involved with the domain concepts that require various views. Using our data assembler allows us to have a nice container of data to graph. Finally, our ordered pair class was derived from the storage concepts. We use it to store data points, so that we will be able to access them.

For the user interface and hardware communication, two other classes were derived rather clearly: the Graphical User Interface class, and the Hardware communication class. The graphical user interface is involved with many domains, save the few domains relating to data storage - that is taken care of by the data manager portion. The purpose of the graphical user interface is for users to be able to easily interact with the application. This includes being able to easily view different portions of the application such as information on their current workout, their history, etc. The next class, Hardware, was derived as a modularized way to communicate with the Heart Rate Monitor which is required to retrieve information about the user's heart rate. This class is simple - its only function is to receive information from the HRM being used, and to update the user's current heart rate in real time.

7.4 Design Patterns

Although our system can be refactored to match any of the design patterns we learned in lecture, we believe that the command pattern most accurately describes our system. The command pattern is as simple as it seems: an object invokes methods on other objects. Because our project is an Android application, the user input occurs primarily through the touchscreen. Our UI is responsible for receiving these inputs and then calling the appropriate methods to fulfill these calls. Each "invoker" knows which object or method to call, and after requesting, the receiving class fulfills the requests. Our system does not really support reversible actions, which are physically impossible, but it can either pause or attempt to return to a previous state for instance by reverting to a previous song.

Our system also makes use of the proxy class with the HeartRate class. As learned in lecture, a proxy object preprocesses requests before forwarding them to subjects when appropriate. In order to interface with our piece of hardware, the chest strap, we made an object to deal directly with the chest strap returning the information to the User Interface, and thus aid the process. Meanwhile, the decorator pattern is used to add non-essential behavior to key objects in our design. In our project, we used a number of "fake" classes to simulate behavior that we were unable to implement for various reasons. For instance, the ChestStrapFake and AudioSystemFake classes were created so we could view the effect of interaction between the UI and Chest Strap or the UI and the Audio System. Upon pressing the appropriate button, we get a nice display on our UI that confirms that the

appropriate method has been called. There are a number of other embellishments used that follow the decorator pattern such as the graph appearances within the data subsystem. However, these are non-essential functional-wise, and the command pattern captures the essence of our design.

7.5 Object Constraint Language

Class	Invariant	Precondition	Postconditionr
IHeartRateDevice	The chest strap is properly connected via Bluetooth to the Android phone on which the application is running.	User begins a new session or updates his existing workout information.	The current heart rate is returned from the chest strap.
ArtistActivity	The database is able to continually receive artist entries.	User requests a graph displaying artist activity (âĂIJPie buttonâĂİ), and the database holds the data to create the appropriate graphs.	TA pie chart of the frequency of each artist played is plotted and displayed.
CalculateTargetFragment	The UI presents a pop-up box for the user to entire information relevant to calculating a target heart rate.	The user touches the âĂIJcalculateâĂİ button on the home screen of the application.	Based on the pattern recognition conditions described in the algorithms section, an appropriate target heart rate is determined based on age and desired intensity.
DataAssembler	The user has accumulated the data to necessitate an SQLite database.	The user begins a workout session that necessitates the storage of various data.	A SQLite structure is created and functions for adding, removing, or changing records to the database are provided.
LineActivity	The database is updating the graph to continually display new heart rates	The user presses the button to graph heart rate vs. time.	A line graph of heart rate vs. time is created and plotted and displayed on the screen for the user to analyze.
Record	The chest strap is continuously sending heart rates over to the Data subsystem.	The user begins a workout session where record objects need to be used.	A record object is created that contains heart rate, time stamp, artist, genre, and song.
SongActivity	The database continually updates the graph as new songs are being played.	The user presses the âĂIJsongâĂİ button to view the graph of the songs played so far.	A pie graph of the frequency with which each unique song is played is displayed on the screen in different colors.
StatisticsActivity	The user has accumulated data to be able to generate graphs.	The user presses the graph button.	The system gives the user a variety of options to select from: Data, Artist, Song, and Genre.
	The UI is fully functional and stays		The home screen

8 System Architecture and System Design

8.1 Architectural Styles

Our system utilizes a three-tier architecture system and consists of 3 layers. These include a presentation tier, an application tier, and a data tier. Our presentation layer is primarily represented by our mobile interface which is used to display our applicationâĂŹs relevant information. It also allows the user to interact with our system by inputting commands and accepting outputs. Meanwhile, our application layer consists of logical operations and data access. For example, our song-selection algorithm would be included in this layer. This application layer uses logical operations to convert raw user data into readable results. Finally, our data tier consists of our database where our information is stored and retrieved.

These three tiers are separated from each other to allow for encapsulation and data abstraction. We want each tier to hide its usage from implementation and to preserve the integrity of our data. We also want to reduce the overall complexity of our system. However, each tier must maintain a sufficient level of communication and be able to retrieve needed data from each other. In a common scenario for our system, our application layer may request information from the data tier. It then processes this information and returns it to the presentation tier in response to the user request. A visual diagram was provided in our earlier stage of planning in the section titled System Architecture Diagram.

The Data Management section in specific implements a two-tiered layered architecture. The first tier consists of the Data Manager, which serves as an interface between the Data Managment module and the other two modules. It provides functions which the other modules can use to store and access data from the database. In the second tier of the architecture, we have the Graph Container and the Data Assembler components. These components handle the actual manipulation and graphing of the data. The main reason for this architecture is to implement data hiding and abstraction; the Data Manager in the first tier serves as an intermediary which commands the Graph Container and Data Assembler to perform certain actions. Thus, the management of the data is done internally, and all the other modules get to see is the interface provided by the Data Manager.

8.2 Identifying Subsystems

Our software is designed around three primary subsystems. The core subsystem is the UI Subsystem, responsible for interfacing with the user and other subsystems. The Audio BlackBox Subsystem handles audio playback, and the selection of track. The Data Logging Subsystem saves heartrate and music playback information, and produces graphs of this data.

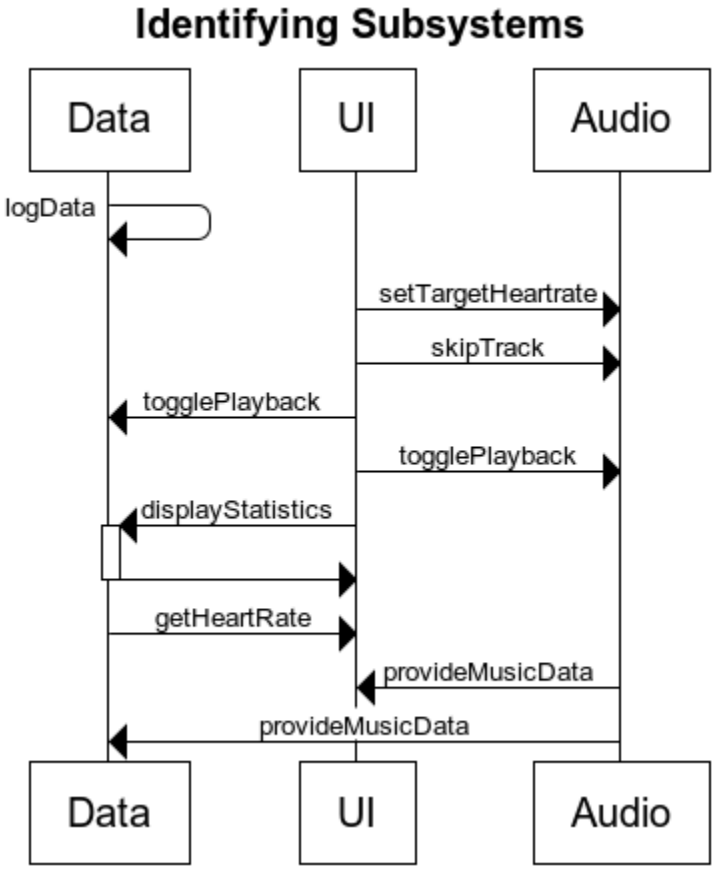


Figure 8.1: Subsystems

8.3 Mapping Subsystems to Hardware

8.4 Persistent Data Storage

Since Android provides full support for SQLite databases, it is the type of storage that we have chosen for the application. The wide variety of fully-developed features allows us to focus more on the actual organization and management of the data in relation to the other modules. All that is needed is a simple call to the data base to retrieve the raw data, and the custom designed objects illustrated in the Class Diagram then do their own processing on the data. SQLite allows us to store all the data specific to application on the device itself, which is advantageous for a mobile application such as ours. The goal is for the user to be able to record and view his workout data without having to use any external devices other than his phone and the chest strap, and internal data storage via the SQLite database allows our application this benefit. The database will be accessible only to the Data Manager and the Data Assembler. In regards to the Data Manager, the only interactions with the database will be to store the initial state of the system, store the current music track, and store the current heart rate. It will not retrieve anything from the database, because that is the purpose of the Data Assembler. The Data Assembler is the other object that will interact with the database. It will issue requests for the various data that the UI would like to graph, which include the heart rate, the current times, and the songs. Thus, the Data Manager and Data Assembler are the only objects that have direct access to the database.

8.5 Network Protocol

8.6 Global Control Flow

8.6.1 Execution Orderness

The execution order is a mix of procedure-driven and event-driven. From a broad view, the use of the program follows the same steps: the user starts the music, the system runs, then the user stops the music. However, the system provides a variety of interface options to activate events during the execution: a user may pause or skip playback, and view their statistics, at any time.

8.6.2 Time Dependency

The system is real-time, with a timer firing once a second. This timer triggers the fetching of the heart rate from the monitor, and triggers the logging of this data.

8.6.3 Concurrency

The Android standard concurrency model is that the main thread handles UI, so lengthy tasks must be performed on a background thread else the UI becomes unresponsive. As such, the network IO of the music selection system must certainly be in a different thread. Synchronization is unnecessary as there is no shared resource.

8.7 Hardware Requirements

The system requires:

- Touch screen display with minimum resolution of 640 x 480 pixels
- Storage space for music library, minimum size of 100 Mb
- Bluetooth for communication with a heart rate monitor
- Network connection for communicating with music selection service
- Audio playback capabilities

All of these requirements are met by most Android phones on the market.

9 Algorithms and Data Structures

9.1 Algorithms

As mentioned throughout this document, music has an undeniable effect on heart rate and exercise. The goal of this project is to both induce and measure that effect. Our system requires good algorithms to ensure functionality, the most important of which, are discussed here.

9.1.1 Pattern Recognition

We wish to recognize a trend in user heart rate data to determine the state of exercise a user is in for a given range. For simplicity, we break down bpm ranges into three categories of intensity:

Light ($x<50\%$)

- sitting
- walking
- golfing
- shopping
- fishing

Moderate ($50\leq x< 70\%$)

- lifting weights
- riding the bike
- playing doubles tennis
- mowing the lawn

Vigorous ($70\leq x<85\%$)

- running

- skiing
- playing basketball
- hiking
- playing soccer
- shoveling snow

Now to determine which of these categories a user"s workout falls under, we need to determine the user"s maximum heart rate by subtracting user age from 220. A light workout is defined as less than 50% of a user"s maximum heart rate. A workout of moderate intensity falls in between 50-70% of a user"s maximum heart rate. Finally, a vigorous workout occurs at 70-85% of a user"s maximum heart rate. These numbers allow us to calculate the minimum and maximum bounds for each category given a specific user.

Thus, classification is extremely simple. We can just find the average bpm for a time period and determine which of the following category ranges it falls under. If for some reason, a user does not offer his age, then we will assume the maximum heart rate to be 180.

9.1.2 Danger Detection

It is important to determine when the user"s bpm is abnormal. If the user"s heart rate is abnormally high or abnormally low, we need to alert the user of their condition and advise them to discontinue their workout and seek medical help if necessary. Our data logging subsystem continually receives bpm data from the chest strap, and for every heart rate received, it does a periodic check to ensure that the user"s workout doesn"t cause an unhealthy stress on his heart.

For our comparison purposes, we use the following conditions. Any normal adult (older than 18) should not have a resting heart rate below 60 bpm, and any normal child (aged 6-17) should not have a resting heart rate of lower than 70 bpm. This means that children younger than 6 years old should not be using our heart rate adjuster. There can be some exceptional cases where people in excellent physical conditions can sustain even lower heart rates than the cutoff conditions we used, but it is safer to send a warning anyway.

As for our upper bound, we use the formula from our previous algorithm, maximum heart rate = 220 –age. That is, if a user"s bpm is above their maximum heart rate, we will send an alert message as a warning. The actual comparison implementation is trivial,

but it is important to recognize how we are determining our thresholds. Again, if the user does not provide his age, we will assume a minimum heart rate of 60 bpm and a maximum of 180 bpm.

9.2 Data Structures

The overarching data structure that our system uses is the database. More specifically, we will use the SQLite relational database management system to store the information pertinent to our system. Our database will use the following data types in conducting logging sessions.

Name: string –The name or ID will be the unique identifier to differentiate between users.

Age: int –The age of a user will be necessary for the algorithmic purposes of determining the maximum heart rate described in the previous section.

Session: int –The session number is incremented and stored every time the user conducts a new workout. This is necessary if a user wishes to view information about a particular workout.

TimeStamp: int –The time stamp is available in conjunction with BPM. Every time our data logger retrieves BPM from the chest strap, it will also take note of the time the information was received.

BPM : int –Last but not least, we have the most critical piece of data, the beats per minute measure of a user"s heart rate which will probably be used in every graph.

As introduced earlier on, we will also need to introduce two classes, GraphContainer and OrderedPair. GraphContainer is essentially an array of possible graphs that the user requests. For instance, our most common graph will probably be the graph of Heart Rate vs. Time. We could also introduce BPM vs. Tempo or BPM vs. Song. However the data type that allows us to create this data structure in the first place is the Ordered Pair. Java does not contain such a class, so we implement our own as described in our class section. Our y variable will almost definitely include BPM as an int or double because we are interested in seeing change in heart rate. Meanwhile, our x value will usually be the time stamp, but could also include Tempo, Song Title, or other music metadata that will be retrieved from the audio subsystem.

Obviously, GraphContainer and OrderedPair are utilized to ensure that when a user requests a graph, it will be readily available. Our graphs will be stored under session

number, which is the intuitive choice for a user to request a graph. If a user can request a graph during a session, it will be done using the current session number.

10 User Interface Design and Implementation

Since we spent the time to make high quality mockups early on in this project, the UI was already well thought out and designed with standard Android UI elements such that it does not have to change much in implementation.

One significant difference with regards to our initial design was the removal of a few features. In the original mockup, there is a settings page with the options to change Music library location, Login, and enable music generation. These features are nonessential, not documented in our Use Cases, and therefore will be removed from the design until essential features are delivered. The Music library location shall be the default Android Music library location, the app shall be single-user (reasonable, since phones are personal items), and music generation is of secondary interest to music playback.
We also removed the About page, reasoning that the simple UI should be intuitive to the user and it would not be worth cluttering the UI with help.

Since we now only have one element to display on the dropdown menu, we will instead have a fixed button to access the Statistics screen in place of the menu. This halves the user effort to access the Statistics screen, now only requiring one click, therefore friendlier to the exercising user. Otherwise, User Interface interactions remain as planned.

11 Design of Tests

Increase/Decrease Target Heart Rate
Test covers : Graphical User Interface **Assumption:** The application is showing the correct screen. **Integration Testing** **Steps:** • Press the button to increase target heart rate • → If the target heart rate has diisplayed an increase in its value, press the button to decrease target heart rate **Expected:** Target heart rate is successfully incremented/decremented when the correct buttons are pressed **Fails if:** • Target heart rate does not change • Target heart rate changes in an incorrect direction

Start/Pause Workout (Music Playback)

Test covers : Graphical User Interface

Assumption: The heart rate monitor is ready to begin collecting data and a target heart rate has been selected.

Integration Testing

Steps:

- User presses button to initialize music playback.
- → If the music playback successfully begins, press button to pause music playback.

Expected: When the user presses the button to begin the music playback, the workout will begin. When the user presses the button pause the music playback, the music playback will pause.

Fails if:

- The music playback does not begin when the button is pressed
- The music playback does not pause when the button is pressed

Skip Track

Test covers : Graphical User Interface

Assumption: Application has already begun music playback and a song is currently playing.

Integration Testing

Steps:

- User presses the button to skip the current track

Expected: The application will play a new song

Fails if:

- Pressing the button does not play the next song

Display Graphs

Test covers : Graphical User Interface

Assumption: User has logged data into the application and is on the correct screen

Integration Testing

Steps:

- User presses the button to display statistics

Expected: The application will display graphs for the

user

Fails if:

- The user presses the button and graphs do not display

Music Algorithm

Test covers : Data Manager

Assumption: The application has been running long enough for sufficient BPM and Heart Rate data to be logged for graphing.

Integration Testing

Steps:

- User requests a graph on Music Tempo vs Heart Rate

Expected: A graph that shows Music Tempo vs Heart Rate should be displayed, and there should be an approximately linear relationship.

Fails if:

- The BPM vs Heart Rate graph does not display.
- The BPM vs Heart Rate graph's data does not match the expected data from the selection algorithm.

Return from Graphs

Test covers : Graphical User Interface

Assumption: The application is currently displaying graphical data.

Integration Testing

Steps:

- User presses the "back" button on the android device

Expected: The application will return to the main screen from the graph display screen.

Fails if:

- The user presses the "back" button but the screen does not change.

Initiate Data Logging

Tests: Data Manager

Assumption: The User Interface has received an input from the user which indicates a desire to start the system.

Integration Testing

Steps:

- The User Interface calls the storeInitialState() function form the Data Manager
- → Data Manager retrieves the current heart rate from the chest strap and time stamp from the device, and stores it in the database.
- ← If the storage occurs successfully, a confirmation is sent to the Data Manager
- → Once confirmation is received, Data Manager begins calling storeCurrentHR() repeatedly.

Expected: The Data Manager is constantly listening for heart rate values, time stamp values, and music data and is logging that information to the database.

Fails is:

- The chest strap does not return a valid heart rate for storage
- The Android device does not return a valid system time.

Further testing must be done in order to validate the accuracy of displaying the user's current heart rate. This accuracy, however, is hard to validate because the application simply displays the number that it receives directly from the heart rate monitor. If the displayed number appears to be off, then the heart rate monitor may be faulty. Otherwise, there is no way to check whether the displayed number is the actual number that the monitor records.

12 Project Management and Plan of Work

12.1 Merging Contributions

We wrote this document in LaTeX, so uniform formatting and appearance is guaranteed by the compiler. We used git version control to share and merge everyone's work, such that each of us was responsible for merging elegantly when conflicts arose. However, as half the team were beginners with both of these technologies, issues arose in improper LaTeX coding leading to compile errors. In these cases, more experienced members would step in to fix the mistakes.

12.2 Project Coordination and Progress Report

Our project is in a unique situation where, due to teamwork disagreements, the Audio Subsystem team has split from the rest of the group. With the professor's approval, we will be submitting two separate reports for grading. As such, certain parts of this report pertaining to the Audio subsystem are not complete.

No Use Cases have yet been implemented, but for the first demo we will have a fully functional user interface and a working data logger. Since application of the Chest Strap is a complicated process requiring the user to moisten the electrodes and strip down, for purposes of the demo the Chest Strap's output will be faked by a UI element allowing real-time modification of the reported value.

12.3 Plan of Work

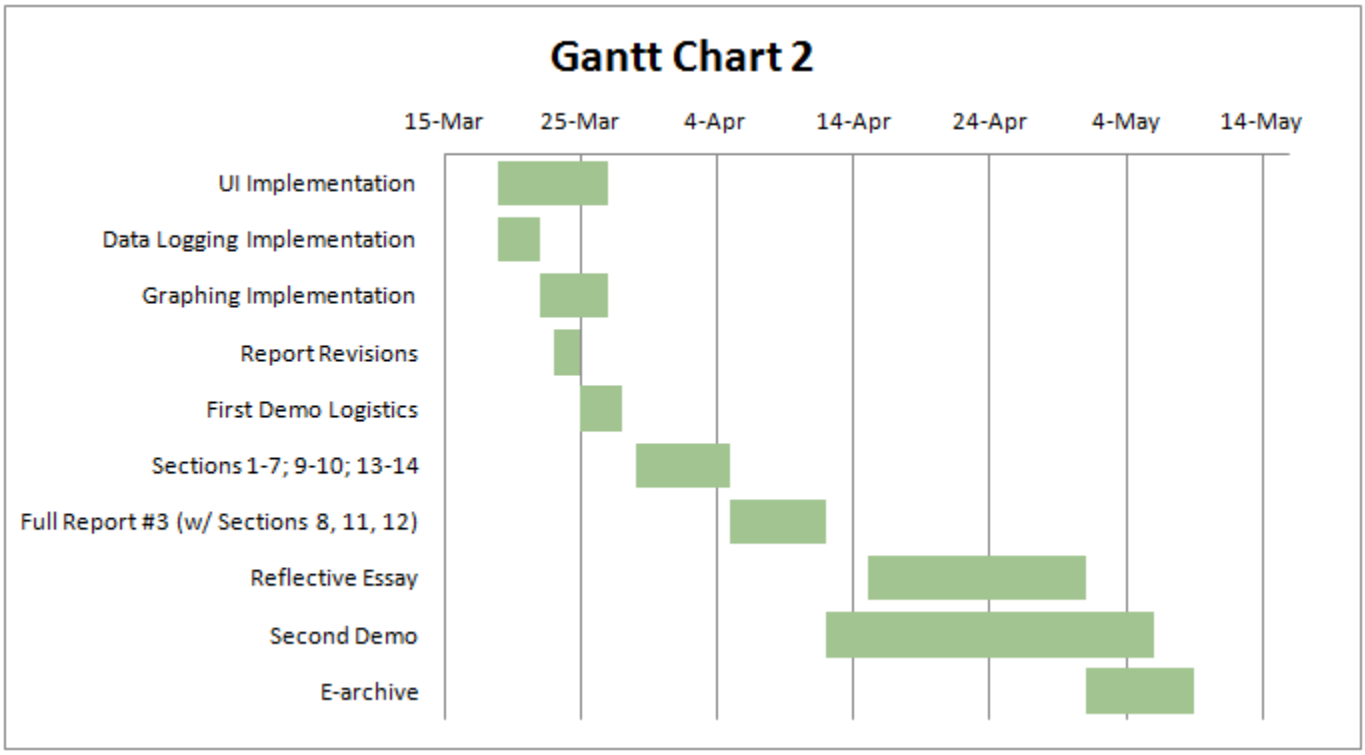


Figure 12.1: Our new Gantt Chart describes our plan of action until Demo #1, and it also highlights some of our projected milestones for the second iteration.

12.4 Breakdown of Responsibilities

The breakdown of responsibilities follows the initial division.
Revan and Tae-Min will develop the User Interface and Hardware Interface. The Hardware Interface was a part of the User Interface in our original design, but has since been moved to the Data Logging subsystem. As the more experienced Android developer, Revan will still be responsible for the Hardware Interface.
Nikhil and Jonathan will develop the Data Logging Subsystem.
Samani and Kenny will develop the Audio Subsystem, albeit independently.

Integration of the UI and Data subsystem will be handled by Revan and Nikhil.
Since one of the elements is a UI, integration testing will be trivial and performed during integration.

Individual Contributions Breakdown

Contributions of report 1:

	Kenny Bambridge	Jonathan Chang	Samani Gikandi	Tae-Min Kim	Nikhil Shenoy	Revan Sopher
CSR	17	17	17	17	17	17
System Requirements		23			43	33
Func. Requirements		50		50		
Non-Func. Req.					100	
Appearance Req.						100
Stake. Actors and Goals				100		
System Sequence Diagrams			100			
Use Cases				36	43	20
UI Spec	33	33				33
Domain Analysis	20		20	20	20	20
Plan of Work		100				
Total	70	223	137	223	223	223
Percentage	6.37%	20.3%	12.43%	20.3%	20.3%	20.3%

Contributions for report 2:

	Jonathan Chang	Tae-Min Kim	Nikhil Shenoy	Revan Sopher
Interaction Diag	50		50	
ClassDiag+Interface Spec		50	50	
Sys Arch+Sys Design	45.5		20.5	33
Algo+Data Struct	50		50	
UI Design+Implem				100
Design of Tests		100		
Project Manag+Plan Work	25			75
Total	170.5	150	170.5	208
Percentage	24.5%	21%	24.5%	30%

History of Work, Current Status, and Future Work

12.5 History of Work, Current Status, and Future Work

As emphasized in our Summary of Changes, our GitHub website lists every single change that has been made: https://github.com/revan/HeartRateAdjuster. Also, on the homepage, we have a log of the key meetings that took place over the semester. In sections 5.1 and 12.3, the Gantt charts depict pretty accurately, our actual schedule of work. We stuck almost exclusively to schedule and after the split within our group, we made an effort to meet all the deadlines. The work completed was within the intervals specified by the Gantt charts, and the milestones were roughly achieved towards the end of the intervals. Although these dates may not match up exactly with our GitHub account, the work was completed, just not pushed.

12.6 Key Accomplishments

Created an aesthetically pleasing Android Application

Constructed a SQLite database that could store information from the heart rate monitor

Application can calculate a good workout target heart rate and alert the user if their heart rate is at a dangerous level.

Application can graph various statistics of the user workout information in real time.

Integration between UI and Data Subsystems

12.7 Possible Future Directions

There are so many possibilities for our project that we have only begun to scratch the surface. First off, we could consider expanding to other platforms, namely iOS and windows phone. This way, our project is more universal. We could also test our project to ensure compatibility with different heart rate monitors. We could then add other functionality,

and record and convey other data to the user other than heart rate. However, the primary goal of our application is to aid the users in working out through the adjusting of their heart rates. Any other useful workout data is secondary. However, we could revise the algorithms used to suggest heart rate, or detect unsafe heart rates. We could add a personalized login for each user. We could compare workout data between friends. We could experiment more with the correlation between the music being played and its effects on heart rate. To conclude, there is always more work that can be done, and The Heart Rate Adjuster is definitely a project that is both reusable and can be built upon with a number of additional features. However, it was a good start, and a great way to familiarize ourselves with many of the concepts of software engineering.

References

References 1-5 are the final project reports of the previous groups who worked on the Personal Health Monitoring projects. They were consulted in conjunction with Professor Marsic's Software Engineering textbook as a guide for formatting guidelines, content ideas, and inspiration.

[0] http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf
[1] http://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2013-g7-report3.
[2] http://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2013-g8-report3.
[3] http://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2012-g1-report3.
[4] http://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2012-g2-report3.
[5] http://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2012-g3-report3.

Reference 6 is a review of the Motorola MOTOACTV device. They provided us with the specifications and usage details to help us develop our project proposal.

[7] http://reviews.cnet.com/specialized-electronics/motorola-MOTOACTV-gps-fitness/4505-

References 7-8 are Wikipedia articles that helped educate us on electroencephalography and electroencephalogram define the terms for our glossary.

[7] http://en.wikipedia.org/wiki/Electroencephalography
[8] http://www.scholarpedia.org/article/Electroencephalogram

Reference 9 provided us with an opening statistic to highlight the industry demand for fitness.

[9] http://www.statista.com/statistics/242190/us-fitness-industry-revenue-by-sector/

References 10-11 are pictures that we used for our cover.

[10] https://yt4.ggpht.com/-knZVRWVniHU/AAAAAAAAAAI/AAAAAAAAAAA/QN5_n28x_R0/s900-c-k-no
[11] http://www2.hu-berlin.de/fpm/graphics/logo_heartbeat-note.png

References 12-13 are information about target heart rates.

[12] http://www.webmd.com/fitness-exercise/healthtool-target-heart-rate-calculator
[13] http://www.livestrong.com/article/105256-normal-heart-rate-sleeping/

References 14-15 explain how exercise and sleep affect heart rate.

[14] http://www.active.com/fitness/articles/how-does-exercise-affect-your-heart
[15] http://www.webmd.com/sleep-disorders/features/how-sleep-affects-your-heart

Reference 16 was consulted in describing the Architectural style of our system.

[16] http://en.wikipedia.org/wiki/Multitier_architecture

References 17-18 were consulted in considering algorithmic design.

[17] http://www.urmc.rochester.edu/encyclopedia/content.aspx?ContentTypeID=1&ContentID=
[18] http://www.livescience.com/42081-normal-heart-rate.html