Musical Heart Rate Adjuster

Group #12
https://github.com/revan/HeartRateAdjuster

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

March 13, 2014

Contents

0.1 Product Ownership

Our team will be divided into three smaller sub-teams of two individuals. 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. corollary

- 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.

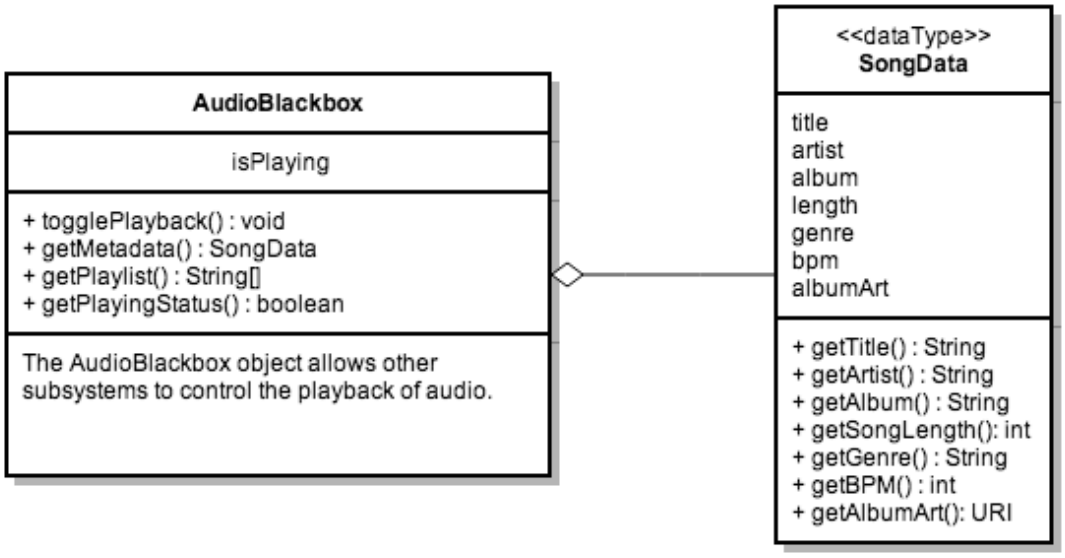
- Kenny and Samani will work on the music aspect of the mobile application. They will study music playback features such as track selection and queuing to provide customers with a seamless music player experience. They will also be responsible for the nuances of audio playback such as crossfading transitions 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.

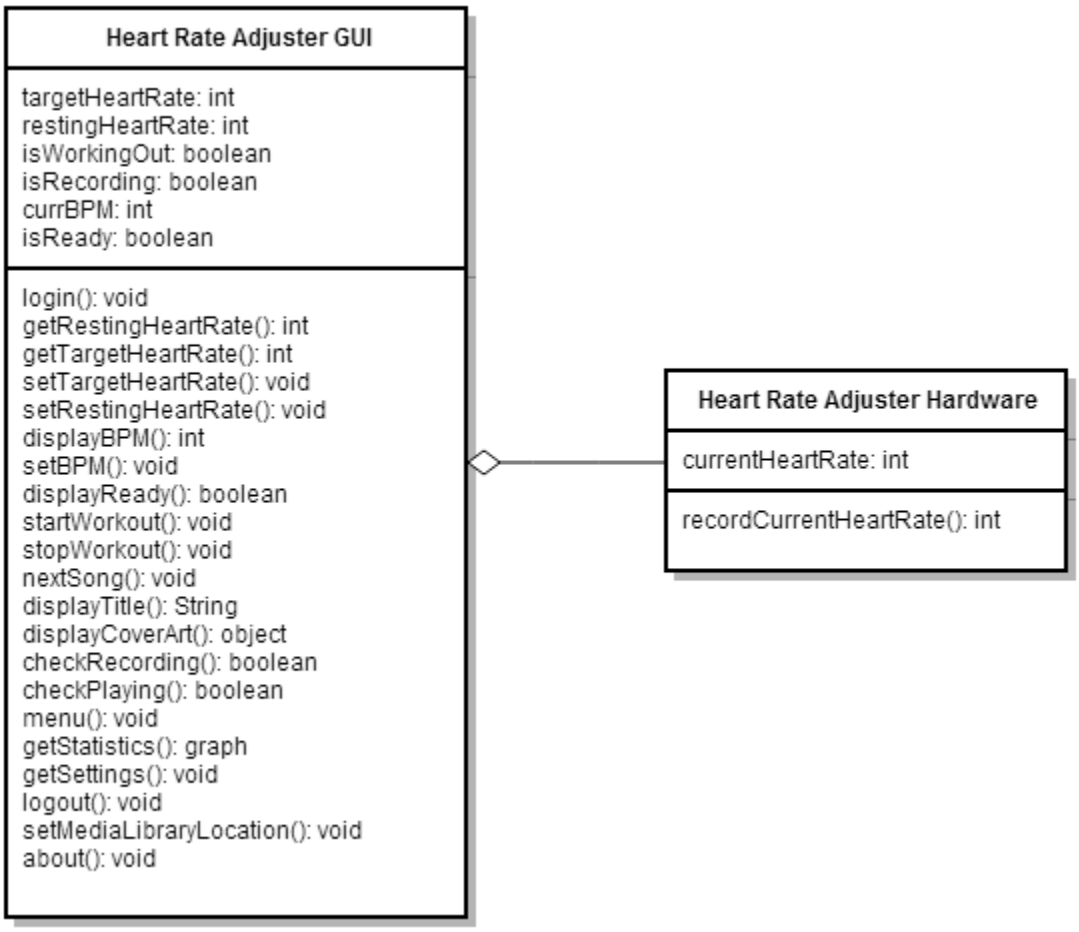
1 Class Diagrams and Interface Specification

1.1 Class Diagrams

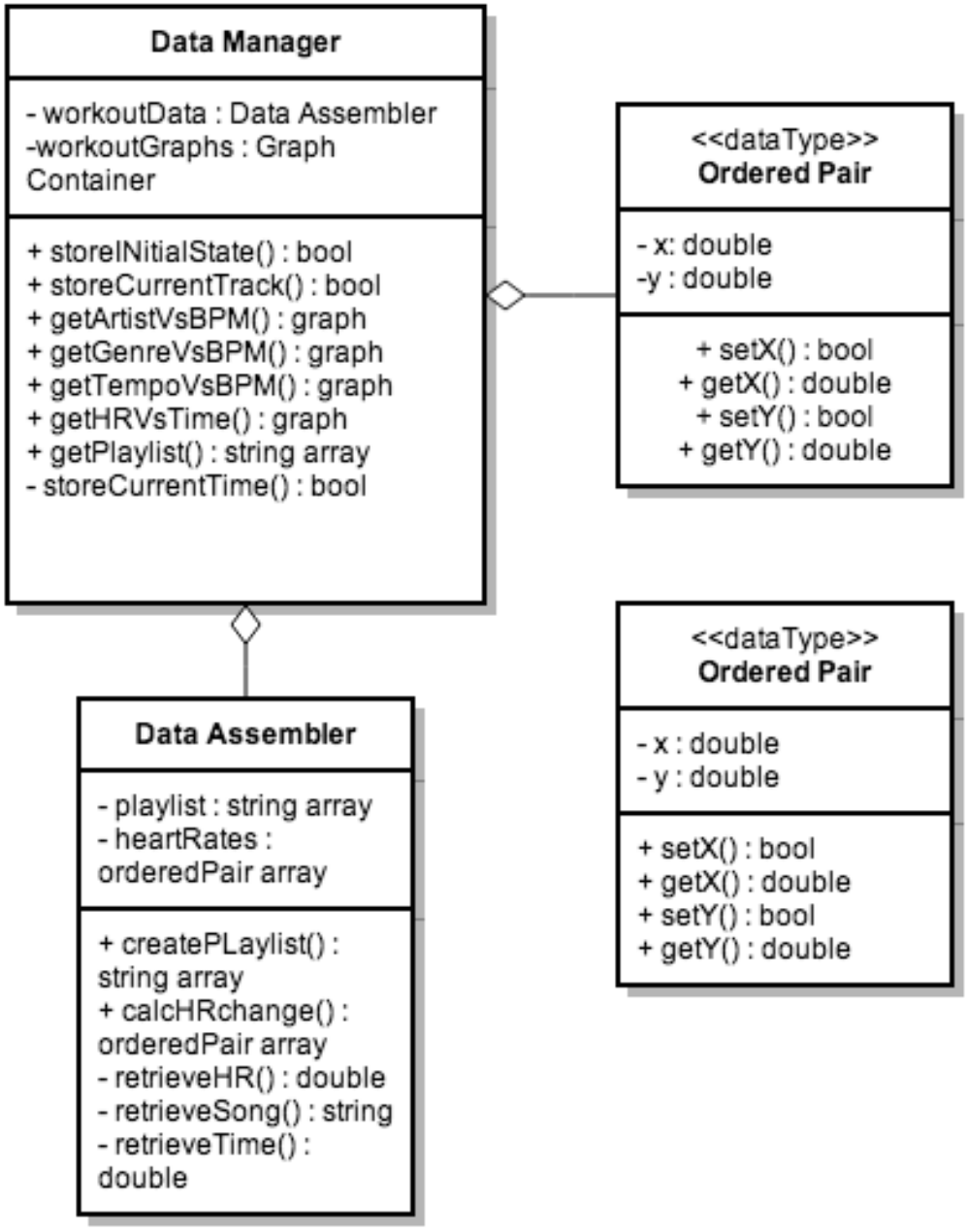
1.1.1 Audio BlackBox Subsystem



1.1.2 General UI & Hardware Subsystem



1.1.3 Data Logging and Display Subsystem



1.2 Data Types and Operation Signatures

1.2.1 Audio BlackBox Subsystem

1.2.2 General UI & Hardware Subsystem

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: • login(): void. This function will log the user in to a new session, keeping session persistence. • 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.

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.

1.2.3 Data Logging and Display Subsystem

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: • createPlaylist() : string array. This function will fetch the songs played during the workout from the data storage and arrange them in the playlist array and return it. • calcHRchange() : orderedPair. This function will retrieve all the heart rates and system times from the data storage and organize them into ordered pairs. The ordered pairs are stored in an array and returned to the caller. • 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: • createArtistVsBPM(Data Assembler workoutData) : bool. This function takes the names of the artists and heart rates stored in the Data Assembler and graphs them against each other. • createGenreVsBPM(Data Assembler workoutData) : bool. This function graphs the genre of the music versus the user's heart rate using the data from workoutData. • createTempoVsBPM(Data Assembler workoutData) : bool. This function graphs the tempo of the music versus the user's heart rate using the workoutData object. • createHRVsTime(Data Assembler workoutData) : bool. 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.

1.3 Traceability Matrix

1.3.1 Audio BlackBox Subsystem

Domain Concepts	**AudioBlackbox**
HRM Manager	
Log Retriever	
Track Logger	
Music Playerbacker	X
Track Queuer	X
General UI	X
Playback View	X
Heart Beat View	
Workout View	
History View	
Workout Store	
Metadata Store	X
Music Store	X

1.3.2 General UI & Hardware Subsystem

1.3.3 Data Logging and Display Subsystem

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

2 System Architecture and System Design

2.1 Architectural Styles

2.1.1 Audio BlackBox Subsystem

The Audio BlackBox Subsystem uses a Client-Server architecture internally. The server stores and coordinates all audio files/streams as well as the playlist determination logic. The client, in this case a mobile device, abstracts the server's behavior into an interface for retrieving metadata and altering playback. Additionally, the client interfaces with the mobile device's faculties for outputting audio.

2.1.2 Data Logging and Display Subsystem

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.

2.2 Identifying Subsystems

Our software is designed around three primary subsystems. The core subsystem is the UI Subsystem. This subsystem is responsible for interfacing with the user, the chest strap, and provisioning other hardware and software faculties. The Audio BlackBox subsystem handles all audio playback faculties. The Data Logging Subsystem is used for recording data about workouts. Additionally, this subsystem provides faculties for displaying and manipulating logged data. Within each of these subsystems are further subsystems.

2.2.1 Audio BlackBox Subsystem

The Audio Black Box refers to two things. 'AudioBlackbox' is a Java interface that lives on the smartphone. This interface is used by other subsystems of the heartBPM application to control audio playback and get information about the music that's currently playing. In general terms, the Audio Black Box is the entire audio subsystem. It handles the queueing of songs, interfacing with native Android subsystems for playing actual audio streams, and with a server backend.

2.3 Mapping Subsystems to Hardware

2.4 Persistent Data Storage

2.4.1 Data Logging and Display Subsystem

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.

2.5 Network Protocol

2.5.1 Audio BlackBox Subsystem

The Audio BlackBox uses live streaming and transcoding techniques to provide audio based on media files living on a server. The Transport layer protocol used will likely be TCP. The Application layer will likely use HTTP.

2.6 Global Control Flow

2.6.1 Audio BlackBox Subsystem

Execution Orderness

Internally, data transfer between the Audio BlackBox server and mobile device is procedural while events can trigger a change in behavior in the way the server behaves.

2.6.2 General UI & Hardware Subsystem

2.6.3 Data Logging and Display Subsystem

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.

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.

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.

2.7 Hardware Requirements

2.7.1 Audio BlackBox Subsystem

The Audio BlackBox Server component requires a server with the processing capacity to transcode media content as well as generate playlists based on the user's activity.

2.7.2 Data Logging and Display Subsystem

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.

Individual Contributions Breakdown

	Kenny Bambridge	Jonathan Chang	Samani Gikandi	Tae-Min Kim	Nikhil Shenoy	Revan Sopher
Problem Statement	X	X	X	X	X	X
Glossary of Terms				X		
System Requirements		X			X	X
Func. Requirements		X		X		
Non-Func. Req.					X	
Appearance Req.		X				X
Stakeholders				X		
Actors and Goals				X		
System Sequence Diagram			X			
Preliminary Design		X				X
User Effort Estimation	X					
Use Cases				X	X	X
System Sequence Diagrams			X			
User Effort Estimation	X					
Domain Model	X		X			
Operation Contracts				X	X	
Mathematical Model						X
Plan of Work		X				
Interaction Diagrams		X			X	
Project Management	X	X	X	X	X	X
LaTeX Maintenance	X	X	X	X	X	X

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