Migraine Trigger Recorder: A Migraine Tracker

M. S. Wickramarathne
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.
mithwick.13@cse.mrt.ac.lk

***Abstract*— The purpose of the project is to develop an effective mobile application to record and track migraines and provide means to organize those records. The project aims to help people suffering from Migraine headaches by tracking, analyzing and giving meaningful reports and suggestions to improve the health condition of the users. Rational Unified Process (RUP) is used for the system development. The target platform for the project are the smartphones running on the Android Operating System. The Model-View-Controller architectural pattern is used in the system design and implementation. The end product of the project is a standalone Android mobile application with a SQLite database to persist the migraine records. The implications of the project output are to provide a simple, easy to use and trustworthy digital companion to improve the quality of life of Migraine patients.**

Keywords*— *Migraine; Android application; Model View Controller architecture; Rational Unified Process

I. INTRODUCTION

Migraine is a headache disorder suffered by approximately 15% of the world population. It is characterized by headaches occurring often or repeatedly with varying levels of severities. Migraines can severely affect the day to day activities since it causes vomiting, sensitivity to light, sound, smell etc. There is no exact cause of the disorder and no permanent cure found to date. They are believed to be caused by a mix of genetic and environmental factors [1].

Therefor it is very important to track and record the causes and patterns of Migraines per individual basis to have a better chance of evading or minimizing the effects of the disorder. Scientists can use the collected data from the users to get a better understanding of Migraine to produce effective drugs and even possibly to find a permanent cure.

Migraine Trigger Recorder is a standalone android mobile application. Its main purpose is to help Migraine patients to track and record their Migraines along with the possible triggers of the headache. Special attention is given to user's health conditions while developing the application. The application hopes to assist users by recording the relevant data regarding each Migraine attack the users' experience. The main expectation of the app is to assist Migraine patients in predicting future Migraines and thereby helping to prevent them by taking proper precautions. This is important since Migraine cannot be cured, but only prevented and controlled.

This paper describes how the Migraine Trigger Recorder application was designed and developed. Section II surveys related work in building applications for Tracking Migraines. Section III provides an overview of the system. System models, design, implementation and testing of the application are described in the later sections respectively. Final section of the paper describes conclusion and future work.

II. LITERATURE REVIEW

Prior to the development of the system began, research had to be done on the current findings and knowledge base about the Migraines. This included finding about the definition, symptoms and current research done on the subject. "Mayoclinic.org" has exceptional details on the particular subject [1]. Data was gathered from reputed sources such as Mayoclinic.org to provide base questions and answers to be filled by the users of the application. It was found out that Migraines are not permanently curable and there are many causes for it. The triggers vary from person to person and it might have an effect from genes.

Apart from the research on Migraine, research and testing were done to get familiarized with the android developing environment. The coding standards, design guidelines were referred extensively since the project has a high concern on usability and simplicity of the final application. Documentation by google on "Android core app quality" was strictly followed when implementing the user interface and performing tests on the system [2].

The proposed application is targeted for the android echo system. Many headache management applications are available in the google play store, but only a handful of them are Migraine specific [3].

"Migraine buddy" application is able to stand out from others and has a good reputation and usability [4]. It provides headache tracking and sleep recording. However, it has a very long procedure to record Migraines. The user has to navigate through many required screens to record a single Migraine. Applications such as "Migraine Diary" and "Migraine" do not provide adequate features and functionality to perform the tracking task properly. They do not provide the ability to record broad information and analyze the past records as desired.

Although most of the existing applications provides well-established migraine management solutions, our application provides a specific solution to Migraine management and analysis while taking usability, productivity and practicality in to great consideration. New system is developed using the Android Studio IDE. It provides built in debugging and support for test harnesses. The minimum supported SDK

API level will be KitKat. The data persistence will be handled by the inbuilt SQL API of android SDK which uses SQLite as the database [5].

III. Overview of the System and Process Followed

Migraine Trigger Recorder is an Android mobile application developed using the Rational Unified Process (RUP). RUP is based on four main phases; inception, elaboration, construction and transition. In the inception phase most of the planning was done while in the elaboration phase most of the design work was carried out. The project included analysis, design, implementation and testing phases where some were overlapped with each other as allowed by RUP. The Application supports devices running Android KitKat and above. Model-View-Controller architectural pattern was incorporated when designing and implementing the application. Construction phase was used for the implementation of the features. And refactoring of the code, testing and deployment were mainly carried out in transition phase. The outcome of the project is an Android application (APK) ready to be deployed on the Google Play store.

IV. System Models

A. System Requirement Specification

The main functional requirement of the application is to record the data related to Migraine attack. Most of the data is to be collected as user input. However, the application must provide the user with an option to choose the level of information recorded. These levels can be categorized in to basic, moderate and full. The application must also support the managing of the answers to the question categories. Initially a set of most common answers would be provided by the application. The user will have the ability to add, delete and reorder the answers according to personnel taste.

One of the most common triggers for Migraine is the surrounding weather conditions. The application must gather the relevant weather data such as temperature, humidity and pressure at the time of the Migraine based on the location of the user. The Migraine application must provide functionality to view past records of Migraines saved in the application. The records will be available in two views, namely the list view and the calendar view. The records must also be able to be filtered according to user needs. Upon clicking on a particular record all the data on that record will be displayed. One of the main features of the application is to provide a comprehensive report view to the user after enough data has been collected on several Migraine attacks. The report would help the user to analyze their situation in a more understandable manner.

The application is used by Migraine patients. Most users will be sensitive to light levels, and colors and they might even be triggers for a Migraine attack. The user must be able to use the application comfortably. Hence a set of customization options must be provided to personalize the experience according to user needs. The application must be able to provide a solid and correct knowledge base to the users. Users can learn about Migraine and the results obtained from past research. The collected data is of great value to the users. It is a record of their medical history. Therefore, the application must have functionality to securely backup the app data and restore them later in the event of an unexpected application behavior or any other unpredicted situation.

When considering the nonfunctional requirements, following are important. Intuitive User Interface, adhere to google material design standards, accuracy and reliability in data and reports. The application must use the mobile hardware resources efficiently. The application is expected to be used by a user for a prolonged time. It should have the ability to record all inputs by user for a several years.

B. Use Case Diagram

The application is a single user standalone android application. The user will have access to all functionalities of the system. When viewing past records the user can filter the records and switch between list view and calendar view. The user can update individual past records. When recording a new Migraine, the user can select the detail level and retrieve weather data. Once the user requests weather data the system would obtain device location and pull the information from internet and display back to the user. The use case diagram of the application is shown in Figure 1.

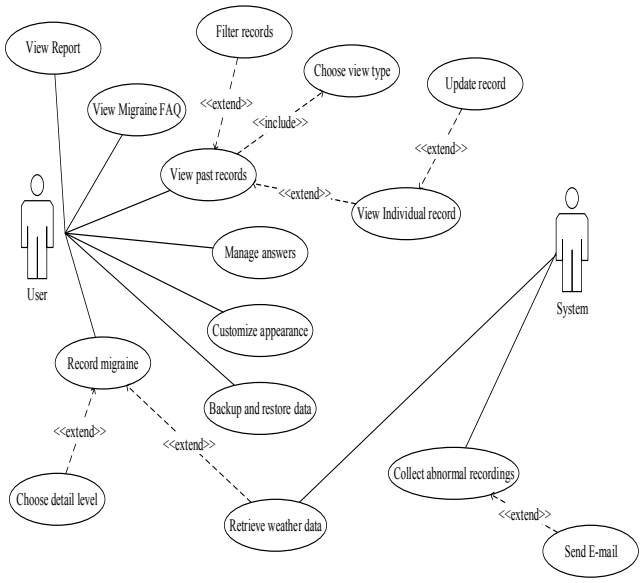

Figure 1. Use case diagram of the system

V. System Design

A. Architecture of the system

The system is a standalone application targeted for the Android operating system. Therefor it is not necessary for the application to be compliant with an existing software or interface. The architecture of the application can be made from the ground up. However, it is mandatory to follow the Google guidelines and recommended architecture for building apps for Android [6]. Android applications use a version of Model-View-Presenter pattern which is

structurally similar to Model-View-Controller (MVC) pattern which will be used to model the architecture of the application.

The overall architecture of the application follows the recommended Android application structure. Android User interfaces are defined as XML files. Those files and related resources such as images, theme definitions will be put under "res" package or folder. The actual java class files are contained under "src" folder. The developer is given the freedom to package the application java classes according to his/her need.

Since the application Uses MVC model, the packages are divided in to the three sections model, view, and controller. Model classes represent the domain objects of the system. They function as data units. Data will be transferred as Model objects within the application. The model classes contain the important attributes and getters and setters for those attributes.

The view classes are the activity classes created along with the XML files. The activity classes function as the interface between Java and XML written user interfaces. It will handle the function of transferring data to and from the user interface to relevant controller classes. These classes are part of the android SDK and inherited from the Android SDK classes. They refer to controller package classes to obtain application logic and data.

The most important package of the model is the controller package. This is the core of the application, where all the application logic is be handled by controller classes. Controller classes accept input via view classes, perform the necessary logic and calculations and return an output. They can access the persistent storage for data using Database Access Objects.

Database Access Objects (DAO) contains logic to handle the database. They are responsible for querying and updating the database for the application. DAO classes include SQL code to communicate with SQLite database. The application need to access location services and internet services. These actions are performed by utility classes and will be included in the Utility package. The package diagram visualizing the above details is shown in Figure

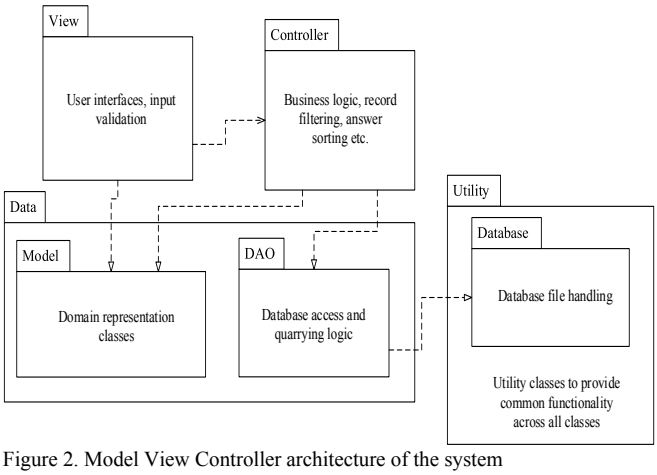

Figure 2. Model View Controller architecture of the system

B. *Logical view*

The class diagram for functionality of adding a new migraine record is shown in Figure 3. Record is the container class for the other domain classes. The user interaction is captured by the "AddRecordView" class and the control is passed to the record controller. The record controller manages the Record class and the classes contained within it. Some of the answers have priority. They are inherited from the PriorityEntity abstract class. The PriorityEntity class implements comparator to compare the relative priorities of answers and sort them in a given list of answers.

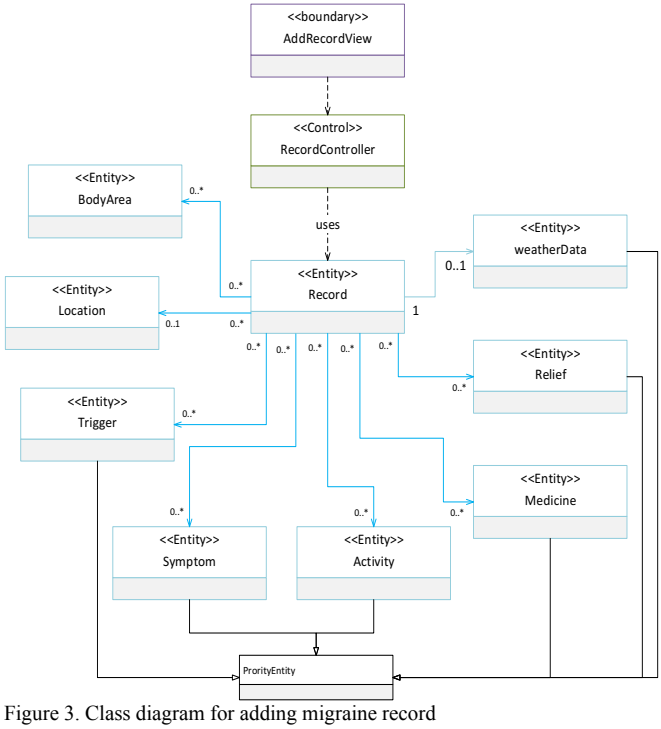

Figure 3. Class diagram for adding migraine record

C. *Process view*

The sequence diagram for adding a new migraine record is shown in Figure 4. This specifies object interactions are arranged in time sequence. When the user wants to add a record, the system askes the level of information to be filled. Depending on the level chosen, a set of questions and answers are provided to be selected by the user. Only filling the start time is mandatory. User can add new answers while filling the record. The user can postpone filling at any moment and choose to fill the data at a later time.

D. *Database design*

The system must be able to manage the answers to different question categories. Since the number of questions is fixed, the answers for the fixed number of questions can be managed in tables apart from the Migraine record itself. User can change the priority of answers in some categories according to their preference.

A priority attribute is required to store this information. The taken Medicine and reliefs can be effective. An attribute is used to store the effectiveness of the taken medicine/relief.

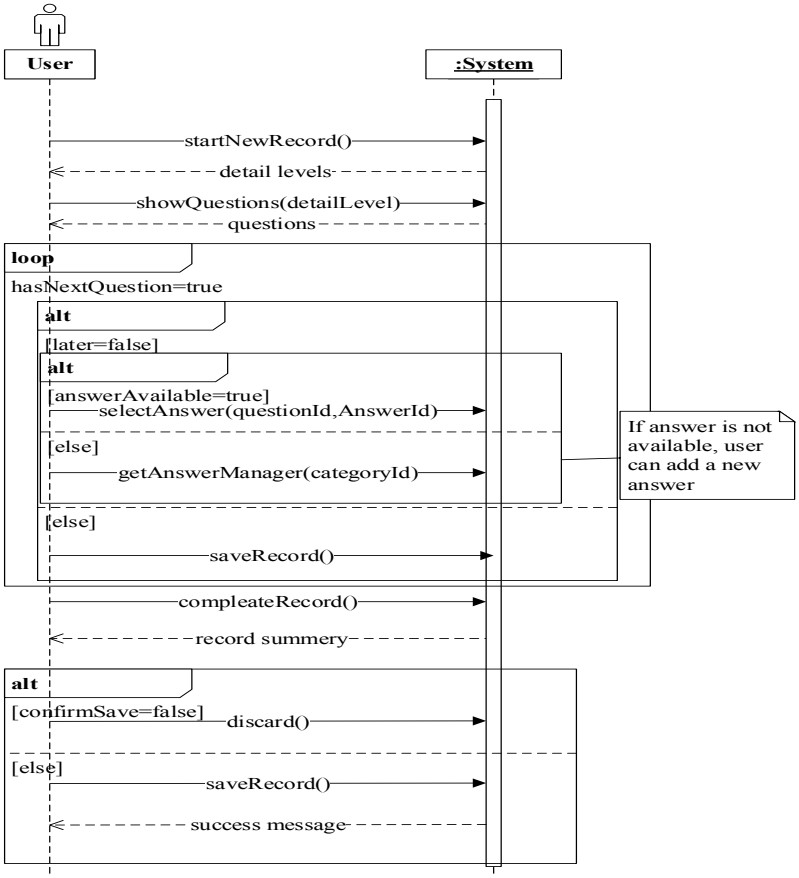

Figure 4. Sequence diagram for adding new migraine record

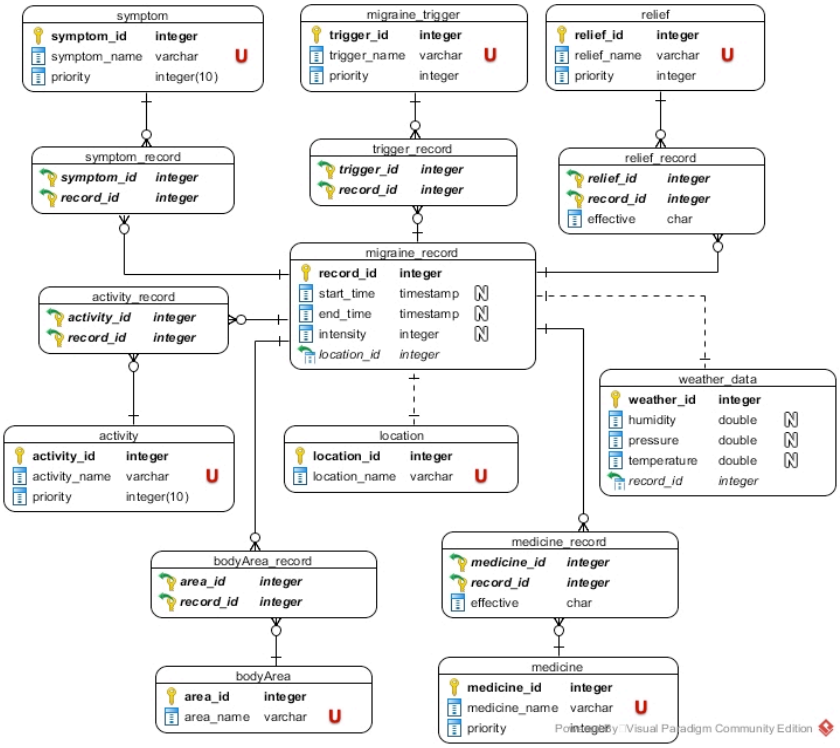

Figure 5. Database schema of the application

A record can contain multiple answers for same question category; therefor the relationship is many to many. Additional association table is required to store these types of relationships. Each Migraine record can associate exactly one weather data record. However, it is not mandatory to have a weather report for Migraine records although a weather record must have a related Migraine record. Therefore, Migraine record and weather report have one to one relationship and a total participation. Figure 5 shows the Data view of the application after above considerations and database normalizations.

VI. SYSTEM IMPLEMENTATION

A. Implementation procedure

Java language is used to implement the business logic and the user interfaces are designed using XML as per the Android design specification. The official IDE provided by Google "Android Studio" was used for the development phase. The IDE has support for visual editing of user interfaces, unit testing, application profiling, logging and debugging. These features make it very versatile software for Android developers. The development was done on a standard laptop computer. The provided Android Device Emulator was used to run and test the application on an emulated Nexus 5X smartphone running Android Marshmallow operating system (API level 23). The target API level of the application is 23 while the minimum API level is 19.

The implementation was tightly bound to Android design specifications and standards. The visual design is inspired by android Material Design. The core logic of the application was implemented using Model-View-Controller classes as designed in the Architecture document. The database was implemented using the SQLite API provided. The access to the database is provided by the singleton class "Database Handler". The business logic is implemented separately from the Android classes responsible for the User interface (UI). The user interface classes, Activities and Fragments were implemented separately in another package. Then the business logic classes were connected with the UI classes. This strategy was useful in checking the correct logic and database functionality without the need of a user interface.

It was required to sort the answers by their priority, for this, a "PriorityEntity" abstract class was created with the priority attribute defined and the answer classes were extended from PriorityEntity class. The comparator interface was implemented in the "PriorityEntity" abstract class to implement the comparison logic to sort the answers.

The official android developers' web site [7] and stackoverflow QA website [8] were of great help in providing required source code examples and answering questions aroused during the development.

B. Algorithm

The record view section incorporates a filter functionality in list view to provide the user with a set of filters to find a set of records by specifying the different answers that the records should contain and match. The algorithm was implemented in "Record Controller" class to filter the set of all records based on the filters sent by the user interface as an ArrayList of strings. The implementation takes a list of all records and a list of strings specifying the filter names and outputs a list which contains records matching the filter criteria. The pseudo code of the algorithm which searches a list for matching filter in a single answer category is shown in Figure 6.

```
filters ← list of filter strings
recordList ← all records
GetFilteredRecords (filters, recordList) {
   filteredRecordList ← new empty list
   recordLoop:
   FOR EACH (record in recordList) {
        bodyAreaFilters ← get body area filters
                          from filters
        bodyAreas ← get bodyArea list of current
                          record

        FOR EACH (bodyArea in bodyAreas) {
            IF (bodyAreaFilters contains name of
                                    bodyArea) {
               add record to filteredRecordList
               CONTINUE recordLoop
            }
        }
   }
   RETURN filteredRecordList
}
```

Figure 6. Pseudo code of the filter algorithm

The algorithm has $O(n^2)$ time complexity since it iterates for each record and for each record checks in the filters for a matching one. The filters are categorized under the question sections, so the algorithm must iterate over each filter set. The algorithm is optimized to be terminated at the first identification of the record.

C. Main interfaces

When the application is launched the user is directed to the main page in Figure 7. This page behaves dynamically and shows time since the user was Migraine free and gives various suggestions if the user is currently suffering from a migraine. Clicking the plus sign gives the user to choose information level for new record.

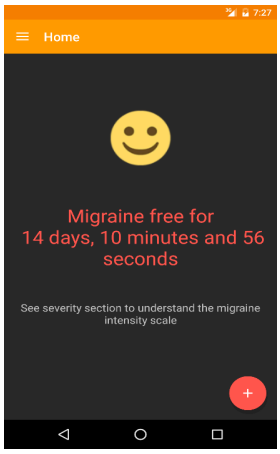

Figure 7. Main page of the application

Once a choice is made, relevant new record form is shown as in Figure 8. The view reports user interface in Figure 9, gives analytical information to the user based on the past records that has been made. The user can select a date range for the analysis. The report can be sent to a third party via email.

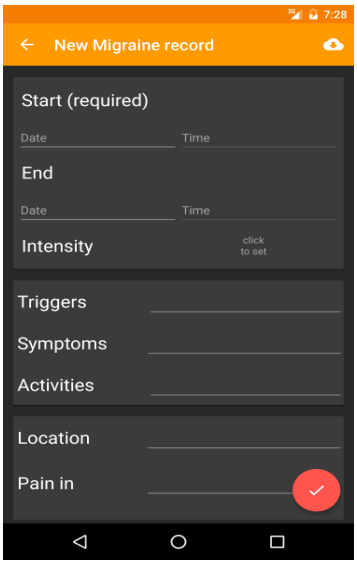

Figure 8. Add new record page of the application

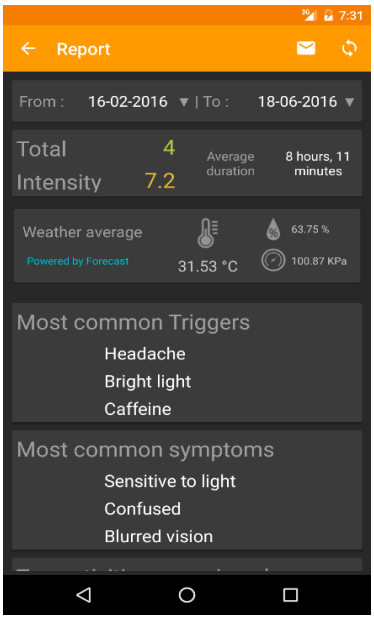

Figure 9. View report page of the application

VII. SYSTEM TESTING AND ANALYSIS

The testing of the application was done under several sections. The database was tested independent of the User interface for correct creation and verification. The database handler class was tested for correct retrieval of readable and writable SQLite database. This was achieved using JUnit testing framework. Unit testing was used to test the correct and intended functionality of the "AppUtil" class which contains all common methods used by various classes in the application. Each method was extensively checked for any unexpected behavior.

The functional tests, performance tests and user interface tests were combined to a single testing strategy where user interface tests were recorded using "Expresso" testing library to check the user interface and to test the fulfillment of functional requirements by the application. The UI tests were used to ensure the use cases identified were able to be fulfilled by the provided interfaces effectively and the standard was acceptable. Performance profiling was done using the integrated device monitor in the Android Studio IDE. While the Interface tests ran, this utility was used to monitor and identify performance issues such as high CPU and memory usage and memory leaks.

VIII. CONCLUSION AND FUTURE WORK

The paper is described the design and implementation of an Android application to record and analyze Migraines. The project produced a decent Android app with pleasing aesthetic features and great functionality. The project development was successfully able to fulfill all the functionalities and cover all use cases as specified originally.

The source code of the application is available at GitHub under the Apache License, Version 2.0. The application has many uses when it comes to researching on Migraines. Researches can use the application as a data collection front end. The vast amount of data collected with the approval of the users would be very useful in research done on Migraines. The past records search algorithm can be further improved using more efficient data structures and algorithms. Furthermore, any contributor can continue with the application and improve the functionality. The application source code is available at the following URL for anyone to check out and contribute in bug fixes and improvements.

REFERENCES

[1] M. C. Staff, "Migraine headache," [Online]. Available: http://www.mayoclinic.org/diseases-conditions/migraine-headache/basics/definition/con-20026358. [Accessed 5 3 2016].

[2] "Core App Quality," Google, [Online]. Available: http://developer.android.com/distribute/essentials/quality/core.html. [Accessed 31 03 2016].

[3] "Google play," [Online]. Available: https://play.google.com/store/search?q=migraine&hl=en. [Accessed 5 3 2016].

[4] "Migraine Buddy," [Online]. Available: https://play.google.com/store/apps/details?id=com.healint.migraineapp. [Accessed 5 3 2016].

[5] "Saving Data in SQL Databases," [Online]. Available: http://developer.android.com/training/basics/data-storage/databases.html. [Accessed 6 3 2016].

[6] "Application Fundamentals," [Online]. Available: http://developer.android.com/guide/components/fundamentals.html. [Accessed 10 04 2016].

[7] "Android developer," Google, [Online]. Available: https://developer.android.com/index.html. [Accessed 18 03 2016].

[8] "Stackoverflow," [Online]. Available: http://stackoverflow.com/. [Accessed 18 06 2016].