



# **ASHESI UNIVERSITY COLLEGE**

**MONITR: A MOBILE APPLICATION FOR MONITORING ONLINE**

**ACCOUNTS' SECURITY**

**APPLIED PROJECT**

B.Sc. Management Information System

**Isaac Coffie**

**2018**

**ASHESI UNIVERSITY COLLEGE**

**MonitR: A Mobile Application for Monitoring Online Accounts'  
Security**

**Applied Project**

Applied Project submitted to the Department of Computer Science, Ashesi  
University College in partial fulfilment of the requirements for the award of  
Bachelor of Science degree in Management Information System

**Isaac Coffie**

**April 2018**

## Declaration

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

.....

Candidate's Name:

.....

Date: .....

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied project laid down by Ashesi University College.

Supervisor's Signature:

.....

Supervisor's Name:

.....

Date: .....

## **Acknowledgement**

Glory be to the almighty God for keeping me through this four years stay at Ashesi. I would like to express my sincere gratitude to my supervisor for his patience, guidance and insightful comments. Many appreciations are extended to my faculty members, who provided me with the basic building blocks of programming and business concepts. I am truly grateful to my family for giving me the encouragement and always being there for me when I needed them most. Lastly, I would like to thank my colleagues for assisting in diverse ways towards the completion of the project.

## **Abstract**

The 21st century has seen a world where almost everything is carried out by digital or electronic means—signalling an end to the traditional approaches. People nowadays transact businesses over the internet due to convenience and accessibility. Taking full advantage of the internet, most institutions especially the financial institutions have also digitalized their services. For this reason, there has been a dramatic increase in the number of online banking and shopping customers. This development has given rise to corresponding increase in cybercrimes, hacking and internet frauds. One of the greatest challenges, hence the cost of migrating an institution's service online is to provide sufficient internet security mechanisms to secure online users' accounts.

However, most institutions lack the technology needed to support their online presence. Consequently, several cases of unauthorized login and stealing of customers' information have become rampant. To address this issue, this project not only investigates the dangers of internet insecurity but develops a novel approach to monitoring a user's online account from fraudsters. The aim of this project is it to monitor, detect and prevent unauthorized account usage using device identification and geofencing techniques.

**Keywords:** Monitor, unauthorized login, internet security, device identification, geofence

## Table of Contents

<b>Declaration</b> .....	i
<b>Acknowledgement</b> .....	ii
<b>Abstract</b> .....	iii
<b>List of Figures</b> .....	vi
<b>Chapter 1: Introduction</b> .....	1
1.1 Background Information .....	1
1.2 Problem Statement .....	2
1.3 Related Work .....	3
1.4 Aim .....	4
<b>Chapter 2: Requirement Analysis</b> .....	5
2.1 Overview .....	5
2.2 Scope.....	5
2.3 Overall Product Description.....	6
2.3.1 System Interface.....	6
2.3.2 User Interface.....	6
2.3.3 Hardware Interface.....	7
2.4 Software Product Functions .....	7
2.5 User Classes and Characteristics.....	8
2.6 Scenarios .....	8
2.7 Use Case and Actor Diagrams .....	10
2.8 Design Constraints and Assumptions .....	11
2.9 Functional Requirements .....	12
2.9.1 User Requirements .....	12
2.9.2 System Requirements.....	13
2.10 Non-Functional Requirements .....	13
2.10.1 Product Requirement.....	14
2.10.2 External Requirement .....	14
<b>Chapter 3: System Architecture and Design</b> .....	15
3.1 Overview .....	15
3.2 High Level System Architecture .....	15
3.3 Data Design.....	17
3.4 Database Design.....	18
3.5 System Modules .....	19
3.5.1 User Interface Module .....	19
3.5.2 Web Service Module.....	24

3.5.3 Login Anomaly Detection Module .....	25
3.5.4 Service Integration Module.....	26
<b>Chapter 4: Implementation.....</b>	<b>27</b>
4.1 Technologies and Tools Used .....	27
4.1.1 Languages .....	27
4.1.2 Libraries .....	27
4.1.3 APIs.....	28
4.1.4 Tools .....	28
4.2 Description of Components .....	29
4.3.1 Implementation Techniques and Process .....	32
4.3.2 Implementation Challenge .....	35
4.4 Evidence of Implementation .....	36
4.4.1 User Registration.....	36
4.4.2 User Login .....	36
4.4.3 User Dashboard.....	37
4.4.4 Service Providers and Account Sync .....	37
4.4.5 Login History Map View .....	38
4.4.6 Remote Action View.....	39
4.4.7 Geofence View.....	40
4.4.8 Manage Devices View .....	41
4.4.9 Push Notification Alert .....	42
<b>Chapter 5: Testing and Result .....</b>	<b>43</b>
5.1 Development Test .....	43
5.1.1 Unit Testing .....	43
5.1.2 Component Testing .....	45
5.1.3 Integration Testing .....	48
5.1.4 System Testing.....	50
<b>Chapter 6: Conclusion and Recommendation.....</b>	<b>51</b>
6.1 Conclusion .....	51
6.2 Recommendation .....	51
References.....	53

## List of Figures

Figure No.	Description	Page No.
Figure 2.1	Use Case Diagram for an Online User	10
Figure 2.2	Use Case Diagram Showing the Interaction Between monitR and Service Provider	11
Figure 3.1	Software Architecture	15
Figure 3.2	Entity Relationship Diagram	18
Figure 3.3	Interaction Between the User Interface Layer and the Web Service	19
Figure 3.4a – f	Mock-Up Design for Screens	23-24
Figure 3.5	Class Diagram of the Application Layer	26
Figure 4.1	Sign-Up Screen of monitR	36
Figure 4.2	Login Screen of monitR	36
Figure 4.3	Dashboard Screen	37
Figure 4.4a	List of Integrated Applications	38
Figure 4.4b	Rules for Syncing Accounts	38
Figure 4.5a	Account Activity Map	39
Figure 4.5b	Details of Login Activity	39
Figure 4.6a	Dialog for Adding a Device	40
Figure 4.6b	Dialog for Taking Action Remotely	40
Figure 4.7	Screen for Adding a Geofence	41
Figure 4.8	Screen for Managing Synched Devices	42
Figure 4.9	Push Notification Message Received	42
Figure 5.1	A PHPUnit test of the User Operation model.	44



Figure 5.2	Test Result of the User Operation model	44
Figure 5.3	Postman Test of the Synchronized Account	46
Figure 5.4	Test Result After Checking Login Request with Valid Data	49

## **Chapter 1: Introduction**

### **1.1 Background Information**

The internet, which was discovered a decade ago is one of the greatest inventions since the dawn of humanity. We have all benefited from the internet in different ways. Majority of businesses and institutions have developed applications to deliver digital services that take full advantage of the features of the internet such as efficiency, simplicity and cost-effectiveness (Aiman, Dafa-Allah, & Elhag, 2017). Examples of such services include e-health, e-education, e-banking, e-commerce, digital marketing, are a few to name. Like government and corporate institutions, individuals use smart devices to enjoy cost-saving benefits and convenience while surfing the web. It is important to note that as users interact with these applications, there is always an exchange of data, some of which are highly confidential. However, the major setback of choosing to use internet driven applications is the fear of information insecurity.

The field of information security aims at securing users' data from unauthorized users. Venter and Eloff (2003) define information security as “the protection of information and minimizing the risk of exposing information to unauthorized parties” (p.301). In as much as software and application developers aim at improving the level of security of their applications, cyber-attacks, internet frauds and session hijacking are still persistent. A recent 2016 *Cybersecurity Insight Report* from AT&T Mobile and Business Solution revealed that there is about 458% increase in the number of times hackers searched Internet of Things connected devices for vulnerabilities (Morgan, 2017). Additionally, according to a 2016 *Security Annual Report* from Cisco, cyber criminals are evading and reconstructing their cyber-attacks and that there were nearly 221% increase in compromised WordPress websites (Morgan, 2017). The statistics is staggering, making internet security, online privacy and information access a crucial topic worth exploring.

## 1.2 Problem Statement

Online users interact with applications using the HTTP (Hypertext Transfer Protocol). However, by default, HTTP is stateless. This means that the communication between a client and a server is not tracked or recognized. Developers nevertheless use the concept of session and cookie to reverse stateless communication protocol into a stateful one. With this implementation, both the client and the server remember the state of the communication. This makes it possible for a user to login into an online platform like Facebook, Google, YouTube and any online banking website and stay active on the page. It is good to remember that the stateful connection is mostly destroyed when the user logs out of the application.

In reality, however, we fall victims of instances where we forget to logout of an application or shut down completely our web browsers. In the event of this, our privacy is compromised as we might be afraid that someone else might use our devices to their advantage. This is a major problem when the information is confidential in the case of banking information. In a recent study, Hewamadduma (2017) examined the usage of phishing attacks and the consequences it poses to online customers and service providers. Surprisingly, about 22.2% of all attacks involved faked bank website. This was attributed to the insecure logins and authentication systems banks currently use.

Furthermore, a report from Ghana's Daily Graphic newspaper in 2016 revealed that Ghana lost about \$50million to cybercrimes (Awiah, 2017). The figure was projected to double in 2017, according to the data security analyst. Additionally, the Identity Theft Research Center (ITRC) in 2015 announced that there were more than 177, 866, 236 personal records exposed via 780 data security breaches (DiGiacomo, 2017). In 2016, 868 cases of security breaches were reported. As it stands, the ability to detect and prevent unauthorized login attempts remains a big challenge for online user and institutions. Also, there exists no available technology that monitors the status of an online account. As such, online users cannot tell if an

unauthorized user has had access to their account. Having been painted with this gap, a novel solution that monitors, detects and prevents unauthorized login attempts using machine learning algorithms, device identification and geofencing technologies is highly recommended.

### **1.3 Related Work**

Users of Google accounts are however somewhat secured when such issues arise. Currently, Google sends email notification to its users if it detects that an account has been logged in with a different device. Information sent include the location the account was logged in, the IP address of the device, the time, the browser details and other information in the past 28 days (Diwanji, 2017). Google then gives users the functionality to log out all devices connected to the account. The same is true for users of Netflix and Facebook.

Account security has been the epicentre of development at Facebook. Facebook recently added new features and system designs that keeps users alerted of suspicious activity so immediate actions can be taken (Axten, 2017). Some of the features described include detecting and blocking suspicious login attempts, deleting deceptive posts and messages, returning hacked accounts to rightful users, are a few to name. Another important feature is the login notification, a new security feature recently released. This feature allows a user to provide details of the device commonly used to log in into Facebook and then be notified when the account is logged in with a device which has not been approved.

Banking institutions also advice their customers to be cautious of hackers and phishing websites. They further advice their clients to report to them when they feel their account is used by an unauthorized person. Regularly, banks send email notifications or SMS to their customers on how to avoid internet frauds (Hewamadduma, 2017).

However, this paper considers these approaches a “passive” way of internet security. Account security should not be exclusively managed by a service provider like Google. The user should have control over his or her account and for that matter, securing the account. This

paper then propose a more “active” way of managing users’ account through this integrated framework called monitR –a mobile application for monitoring online accounts’ security.

#### **1.4 Aim**

This project offers users a proactive, yet secure way of monitoring their online accounts. The aim then is to provide a solution to the question, “how can we reliably detect and prevent unauthorized login attempts?”. The idea is to give users the ability to set security control measures over their online accounts. Stated differently, he or she will not wait for Google or Facebook to send a notification that the account has been logged in by device X at time t. Users of banking platforms also will not have to report to their bank administrators of any unauthorized use of their account before a forensic analysis is performed. They can take action from the app and control how their accounts are accessed; from what location and /or device.

The system not only benefits online users, but it caters for the security issues faced by service providers and institutions. Usually, when a service provider is not able to detect and prevent an unauthorized login request, the customer may file a law suit against them and they may suffer reputation damage and/or financial loss. With this approach, financial institutions can deliver good security services as they can now reliably detect and prevent unauthorized login attempts based on the security policy set by the user of that account X.

## **Chapter 2: Requirement Analysis**

### **2.1 Overview**

This chapter highlights the functional and non-functional requirements of the proposed application. It provides guidelines which will be followed in the development of the application. This document also contains narratives, use case diagrams, and supporting documents needed for a successful implementation of the software application.

### **2.2 Scope**

The monitR application is a mobile application that helps online users to proactively take remote action if a suspicious login is detected. The system is largely an integration of Application Programming Interfaces (APIs) from service providers in a mobile application to allow users to view the security posture of their online accounts. Users are allowed to set their own security policies for their respective account(s) after successfully synchronizing these account(s) with monitR. This project is more concerned with the overall software efficiency and performance, with less attention given to the user interface and other aesthetics.

#### **The software system will:**

- I. Give users the ability to synchronize their online accounts unto the application for monitoring.
- II. Provide a means for users to set their own security policies concerning which devices, geographical location, and IP addresses should have access to their account.
- III. Provide real-time information to users about their account posture in an interactive interface.
- IV. Give users the ability to proactively take actions against any suspicious login either by remotely signing out or blocking an unauthorized device.

## **2.3 Overall Product Description**

The monitR application is a mobile-based application that provides institutions and users with an extra layer of security and authentication mechanism. The system requires both internet connection and GPS location services to be able to effectively monitor the account status of a user. The system makes use of the client-server model, where the client (the mobile application) makes an HTTPS request to the web server and gets a response. In addition, the application communicates with an online service provider's system through RESTful API calls. The service integration makes it possible for monitR to provide real-time information to users about the status of their account from their service provider. Lastly, all information and data collected from the users are securely saved in a database, which is hosted on a web server.

### **2.3.1 System Interface**

As stated above, the system interfaces with a service provider's system through RESTful APIs connection. The API requests between the system and external services will be expressed in a JavaScript Object Notation (JSON) format.

### **2.3.2 User Interface**

This section presents a set of interfaces that must be defined to enable users interact with the system. The user interface follows design conventions, which pave way for easy usage, hence excluding likely difficulties to be encountered by the user. The navigation menus, dialogues, icons, input fields etc. of the interface are visible with clear descriptions of their functions. The system will have an interface that:

- I. Allows users to sign up or register with the application.
- II. Allows users to sign in to the application.
- III. Provides users the ability to synchronize their account(s) with the application.
- IV. Sends push notifications to users about the status of their account
- V. Displays a list of online accounts that the user has synchronized with the monitR application.

- VI. Allows users to view their account's login activity in a map with the locations from which the account is currently being used.
- VII. Allows a user to remotely take action per account.
- VIII. Allows a user to change his or her profile.

**A Walkthrough of the screens:** A first-time user of the mobile application should see the login screen. If the user has not created an account yet, he or she will have to navigate to the signup screen by clicking on the signup button. Upon successful registration, the user can then log in. After a successful login, the user will see a dashboard where he or she will see a list of all accounts that have been synchronized on the system. The user can select any of the accounts and will be taken to the account's screen. The user can then view the account's login history which will open another screen with a map of where the account was previously logged in or actively being used. The user can click on any of the markers on the map and an alert dialogue will show up, asking the user to either log out or block the account. Besides, every user should have a profile screen where they can update their user information.

### **2.3.3 Hardware Interface**

The system runs on an Android operating system with a minimum SDK version of 19 or higher, which is Android 4 or higher (KitKat). It does not require any special configuration to be made on the system's hardware. The system also requires devices that support GPS location service, so it can accurately provide real-time information to users based on their current location.

## **2.4 Software Product Functions**

The functions of the software product include the following

- I. Synchronization of Online Account:** The system will first authenticate a user's login credentials after which its online accounts are synchronized unto the platform.



- II. **Generation of Login Activity Report:** Once a user has successfully integrated his or her account unto the system, the system will automatically generate real-time information about the account's login activity.
- III. **Support for Setting Security Policy:** The system will provide means for a user to set basic security policy for an account. That is, the user can add a list of devices that can exclusively access an account.
- IV. **Remote Control Over Account Usage:** The system will give users the ability to take proactive actions over their account anytime they detect suspicious account usage. The user can then decide to either block the account the device that has had access to the account remotely or terminate the session by signing out of the user.
- V. **Subscription unto the platform:** The system will give users the ability to perform login and signup functionality.

## 2.5 User Classes and Characteristics

The various users or actors identified in this system are online users, service providers and the system administrator (monitR). However, the primary users of the system are account users and service providers. A general characteristic of these users in relation to the system include:

- I. They are literate. Thus, can read and write to effectively communicate with the system.
- II. They can use an Android or iOS mobile device.
- III. They care about the security of their online accounts.
- IV. They are conversant with web development concepts.

## 2.6 Scenarios

This section gives narratives of how the actors interact with the system and what activities they can perform. The findings gathered during the elicitation process translates to the following scenarios.

### **Case 1: Online User Perspective**

Peter Brown, a young undergraduate student has three online accounts. One with Jumia, his shopping account; one with Facebook, his social media account; and another with Ecobank, his bank. As an adult, he fears the dangers of using the internet, especially of the recent online attacks and frauds as highlighted in the above chapter. As such, he would love to know the status of his accounts – that someone has or has not had access to any of his accounts. To overcome this problem, his close friend Faith, recommends to him a novel mobile application called "monitR". Peter downloads and installs the application on his mobile device. He successfully signs on unto the platform and he synchronize all his online accounts.

After synchronizing his accounts, Peter checks his Ecobank account from monitR and was taken to a map screen where all the devices that are currently using his Ecobank account were shown to him. He realizes his Ecobank account is actively used by someone in Kumasi while he is currently at Ashesi campus. From the application, he remotely logs the person out of the account and was safe. For a more secure service, Peter again realizes he skipped the security settings –that explained why his account was used by the unauthorized user. He quickly navigates to the settings screen where he set some basic security policies like adding geofence to block access outside that geographic area. More to this, he configures a list of devices that can access any of his accounts. Peter is now a happy adult as with monitR, he is assured no unauthorized users will have access to his account.

### **Case 2: Service Providers Perspective**

Jumia, a successful online shopping platform targeting the African market, is much concerned about securing its clients and subscribers from online attacks and session hijacks. MonitR, a third-party service provider that gives online users the ability to monitor their online account contacts Jumia for service integration. MonitR and Jumia are now partners because of

the shared interest. Anytime a client or a subscriber of Jumia (say client X) tries logging in into his online Jumia account, Jumia first sends an API request to the monitR application to verify the security policy of that client. MonitR then performs background checks and sends an API response to Jumia based on the security policies set by client X. Jumia now allows or blocks the login request based on the status of the response from monitR.

## 2.7 Use Case and Actor Diagrams

A visual representation of the use case scenarios depicted above is explained in this section.

The identified actors include:

- I. **An Online User:** User subscribes unto the platform and configures his online accounts to be added to the system. The user is responsible for setting basic security policies for his account. The user can check the status of his account, from where he or she can take actions to secure the account. That is, by remotely blocking or logging out the unauthorized user. Figure 2.1 below shows the use case diagram for the online user.

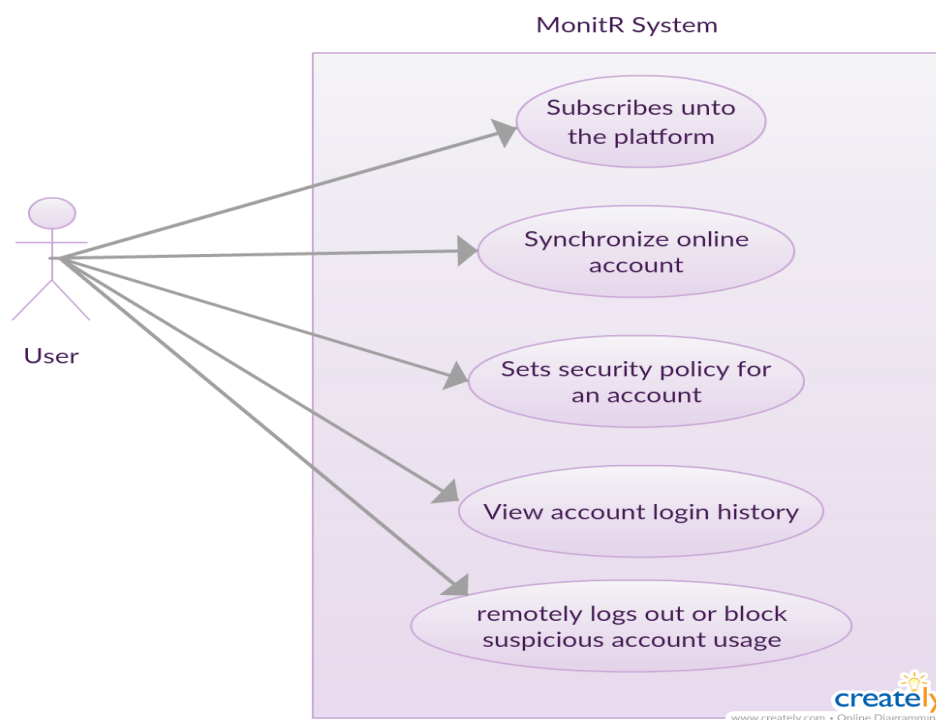


Figure 2.1 Use Case Diagram for an Online User

**II. A Service Provider:** A service provider provides services to online users. A provider can integrate its services with monitR. A provider can send an API request to monitR and gets a response from where it can allow or reject login requests.

**III. A third-party Service Provider (monitR):** The proposed system (monitR) acts an intermediary between an online user and a service provider. It manages all user requests from registration, signing in, synchronizing online account, setting security policy and viewing account history. In all, the system is responsible for monitoring the accounts of an online user. Figure 2.2 presents a use case diagram showing the interaction between a service provider and monitR.

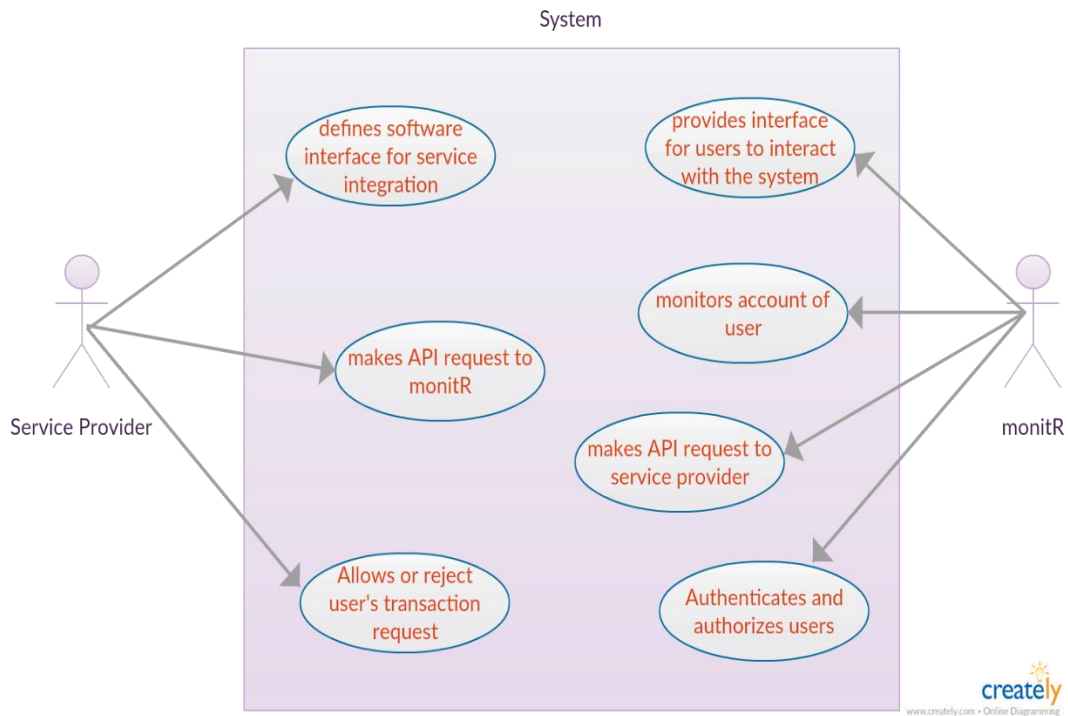


Figure 2.2 Use Case Diagram Showing the Interaction Between monitR and Service Provider.

## 2.8 Design Constraints and Assumptions

To provide a successful solution to the problem statement, the software made the following assumptions

- I. The user can read and write English.

- II. The user can use an Android mobile device.
- III. The device supports GPS services.
- IV. There is an uninterrupted supply of internet connection.

## **2.9 Functional Requirements**

Included in this section is a detailed set of requirements that enables software designers to design a system that meets the goals and requirements of the user. These requirements are divided into user requirement and system requirements.

### **2.9.1 User Requirements**

These requirements define all the activities a user can perform while using the monitR application. It is of high priority as it is the first point of contact to any user who needs to secure his or her online account. They include the following:

- I. An online user must be able to create an account with the proposed monitR application.  
The user must enter his or her basic information to register with the system.
- II. An online user should be able to log in and log out of the platform. A user must provide the correct login credential before login can be allowed. Any wrong login credentials or credentials not matching the predefined validation policy would be flagged as an error.
- III. A user must be able to synchronize his or her online account unto the monitR platform.
- IV. A user should see a list of all online accounts that has been synchronized with monitR.
- V. A user must receive notification about the status of his or her account
- VI. A user should see a report of his or her login history well displayed on a map
- VII. A user must remotely take actions – either log out or completely block any unauthorized user.
- VIII. A user must be able to set security policy per online account. Possible security policy includes geofencing and device identification techniques.

### 2.9.2 System Requirements

This section highlights the requirements that the system must have to meet the functional requirements listed above.

- I. **Simple User Interface:** The system must define interfaces through which the user can interact with the system. The software interface must follow design conventions which allow easy usage and do not pose any difficulty to the user. The navigation menus, dialogues, icons, input fields etc. of the interface must be visible with clear descriptions of their functions.
- II. **Software Interface:** As the proposed solution is intended to communicate with other systems over a RESTful API connection, it is required that the system defines a clear interface which facilitates a seamless communication between the API endpoints.
- III. **Authentication and Authorization:** The system should provide correct authentication and authorization mechanisms. Thus, users must first be authenticated using secured authentication techniques like login verification, password, secret PIN etc. before granted access to the system. After authentication, the system must define authorization policies to ensure users access the resources they are supposed to.
- IV. **Security:** The system must be secured and the communication between the client and the server must be protected from attacks. The system should prevent MySQL injections and possible breach of systems. Also, all sensitive information like user password and token must be hashed or encrypted by the system.
- V. **Scalability:** The system should be able to scale up as more users join the platform.

### 2.10 Non-Functional Requirements

Discussed in this section are services that the proposed system does not provide directly for a user but are relevant for the operation of the system. They include performance, reliability and privacy.

### **2.10.1 Product Requirement**

**Performance:** The proposed monitR application should be designed for high performance such that there is low latency when making requests to the server. The mobile application should not drain much battery power.

**Availability:** The system should equally be up and running with no downtimes. That is, it should be available to authorized users always.

### **2.10.2 External Requirement**

**Privacy:** The system must adhere to the privacy policies as defined by the Ghana Data Security Association. The system must not use gathered from the user for any reason other than the contract between the system and the user.

## Chapter 3: System Architecture and Design

### 3.1 Overview

This chapter provides a high-level system overview and architecture of the proposed MonitR software. It contains details of key modules and data design for the requirements specified in chapter 2 above.

### 3.2 High Level System Architecture

The figure below is an overall diagrammatic view of the system architecture. It defines the component of the MonitR application and its interaction with external services.

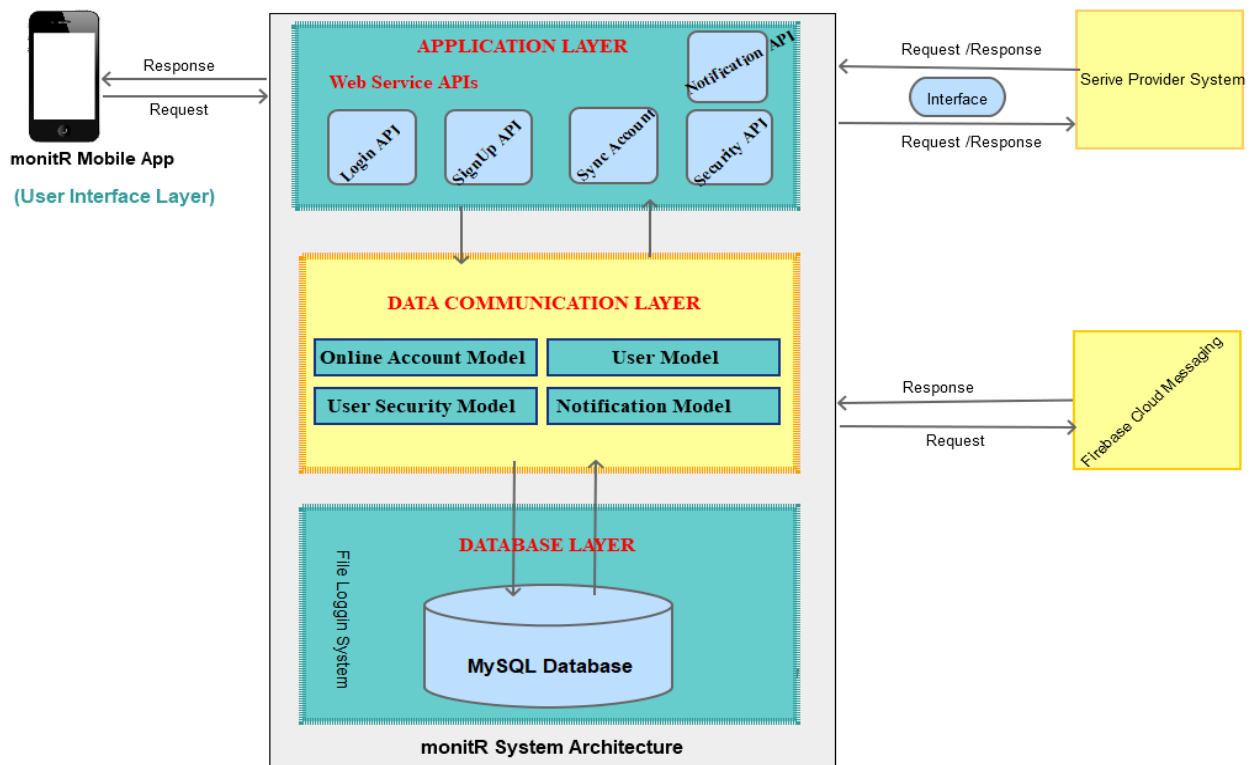


Figure 3.1 Software Architecture

The system architecture as shown in Figure 3.1 above was modelled using the n-tier layered architecture. The n-tier architecture is an architectural pattern that separates a software system into different layers. The most commonly used layer is the three-tier architecture, which separates a software application into user interface layer, business logic layer and the database



layer. This design pattern helps developers to create flexible, maintainable, and reusable codes. The separation of software system in layers equally makes changes to be made at one level without having to change the entire application. A description of each layer is explained below.

**The Presentation Layer:** The presentation or user interface layer is the android mobile application (monitR). It is the component that feeds data to the database layer via API calls. It also displays data to the user in an interactive interface. A detailed description of the user interface design is shown in a subsequent section (see section 3.5.1) of this chapter.

**The Application Layer:** The Application Layer, also the business logic, is made up of Application Programming Interfaces (APIs). This layer is responsible for listening to requests from the user interface layer, processes the request and sends respond thereof. The APIs include

- I. Login API: Responsible for handling login and logout requests.
- II. Signup API: For handling signup requests.
- III. Online Account API: For handling synchronization and management of online account.
- IV. User Security API: For handling security requests. It supports adding and managing user devices and geofences.
- V. Notification API: This API is responsible for sending firebase push notifications to users anytime a suspicious login attempt is detected.
- VI. App Engine API: This API is the core of the application. It handles the anomaly login detection and service integration module.

**The Data Communication Layer:** The data communication layer contains models or objects that defines operations that can be performed on the system. The system has the following models.

- I. User Operation: A class that abstracts the login and signup functionality of the system.

- II. User Account Operation: A class that contains methods that a user can perform on an online account. This is linked to the functional requirement where a user can manage his or her online account.
- III. User Security Operation: A class that contains methods to help users set security policies for an online account.
- IV. Notification Model: This model handles push notification operations.

**The Database Layer:** This layer stores and retrieves data to the other layers. It includes a file logging system, which keeps track of all request and responses made by users. All data are stored in a MySQL database.

### 3.3 Data Design

Data from the monitR mobile app will be hosted on a secured online server. As the system is heavily dependent on data from service providers, with the data being of security and privacy concern, the classes that handle the data can only be accessed using the n-tier layer pattern as described above. These classes will be implemented using PHP and the data will be stored permanently on a MySQL database. More to this, the system will maintain a history of users' requests using a log file for each operation. With this implementation, the system can easily keep track of users' transaction should there be a security concern. Lastly, data from the client to the server and vice versa will mainly be in JSON (JavaScript Object Notation) format, which is easy to handle and manipulate. A sample JSON response (see Listing 3.1) shows the data received from querying a user's login activity.

```
{
  "user_id": "Any valid Id",
  "user_name": "Some Name",
  "login_activity": [
    {
      "device_imei": "some device serial or IMEI",
```

```

    "device_name": "Some device name",
    "last_login": "2018-02-14 11:25pm",
    "lat": 5.7594880,
    "lng": -0.2201840
  }
]
}

```

Listing 3.1 Sample JSON File Received from the Web Service

### 3.4 Database Design

Figure 3.2 below shows a high-level overview of the database system. The *App Users* table stores information about registered users. The *Service Provider* table keeps details of partnered service providers. *Online Account* table stores online accounts information. A user is mapped to an online account in the *User Online Account* table. *User Geofence Policy* table stores geofencing policy set by a user. Similarly, *User Device Policy* table saves all devices that has been authorized to access an online account.

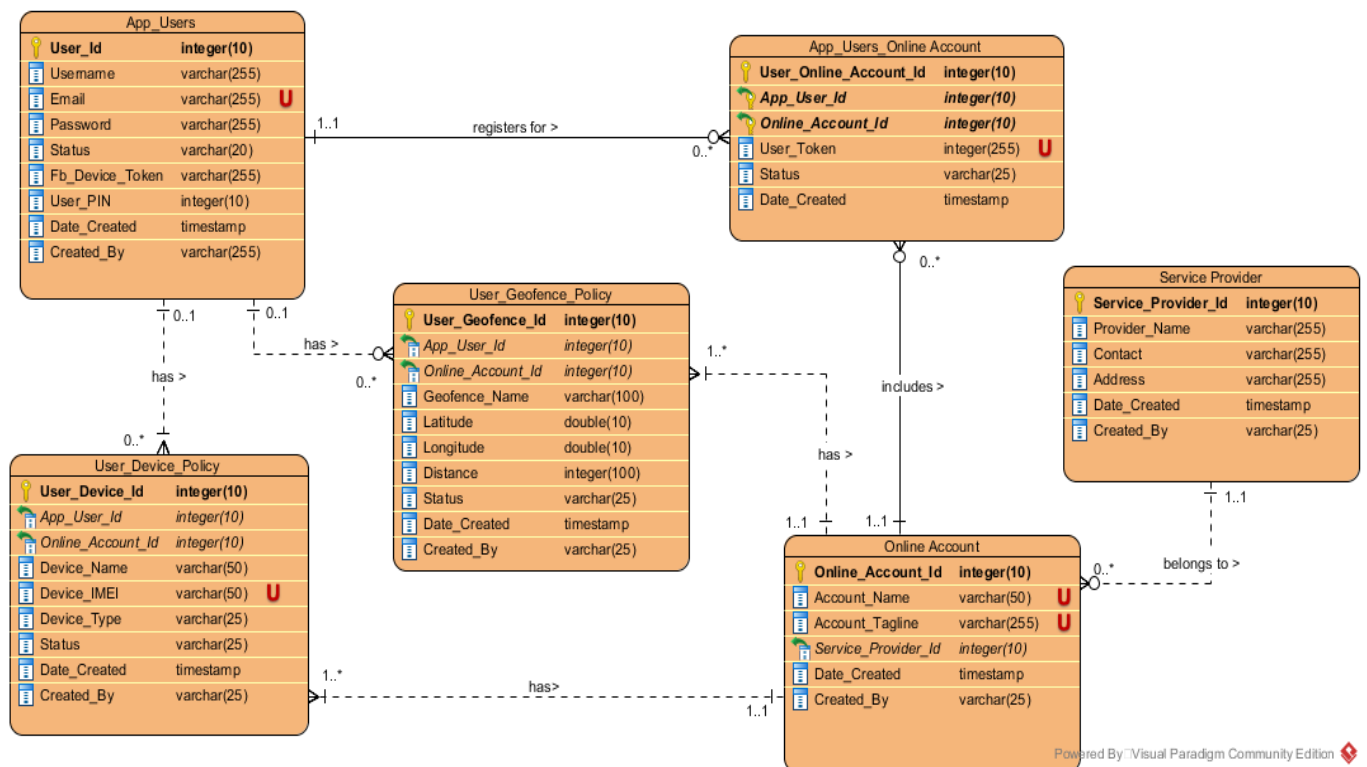


Figure 3.2 Entity Relationship Diagram

### 3.5 System Modules

This section contains all the modules that make up the monitR application.

- I. User Interface Module
- II. Web Service Module
- III. Login Anomaly Detection Module
- IV. Service Integration Module

#### 3.5.1 User Interface Module

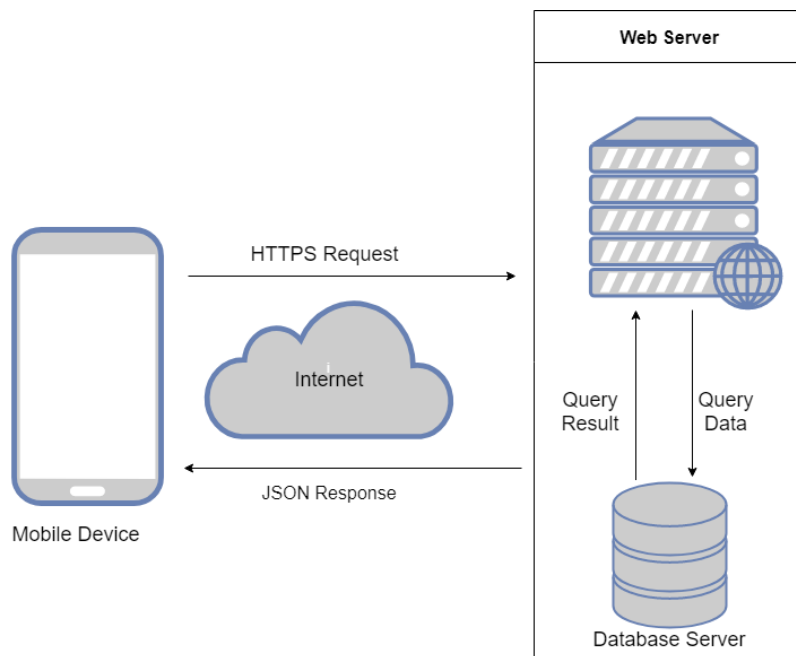


Figure 3.3 Interaction Between the User Interface Layer and the Web Service.

The user interface module is the android mobile application (monitR). It functions to give users a view with which they can interact with the system and to perform the operations specified in the functional requirement document. Owing to the fact that the user interface is a mobile application, it will be implemented using a modified version of the MVC (Model View Controller) design pattern. That said, the design pattern for this module will be Model, View and Activity (MVA). The model contains the data that will be used in the system. The view is

simply an XML layout file. The Activity, also the brain of the application, connects the view to the data. It is also responsible for handling user interactions. Each component is explained as follows:

### 3.5.1.1 The Model

The model contains classes and objects that will be used in the project. They include:

1. **UserAccount Model:** This model describes or abstracts the data that makes up an online account. It contains the following attributes or member variables:
  - a. **Account Name:** The name of the online account. An example is Facebook or Jumia.
  - b. **Account Tagline:** This is the account's tagline or motto. For instance, the tagline of Jumia is "The largest online shopping center in Africa".
  - c. **Account ID:** The unique identifier of an online account.
  - d. **User Account ID:** This attribute uniquely links a user to an online account.
2. **UserDevice Model:** This model represents a user device object. It has the following attributes:
  - a. **Device Name:** The name of a device.
  - b. **Device IMEI:** The device's serial or IMEI (International Mobile Equipment Identity)
  - c. **Device ID:** The unique identifier of the device.
  - d. **Device Type:** The type of device, whether trusted or blocked.
  - e. **Account ID:** The unique identifier of an online account.
3. **Login History Model:** This model handles user's login history operation. Its attributes include:
  - a. **Device Name:** The name of a device.

- b. **Device IMEI:** The device's serial or IMEI (International Mobile Equipment Identity).
  - c. **Latitude:** The GPS latitude of where the login request was sent.
  - d. **Longitude:** The GPS longitude of where the login request was sent.
  - e. **Token:** This unique token is created for every login.
  - f. **Date:** The date where the login request was sent.
  - g. **Status:** The status of the login activity. It can be either active (still in session) or inactive (when the session is terminated).
4. **Shared Preference Manager:** This model is responsible for storing small basic information about the current user. Thus, anytime a user successfully logs into the application, an object of this model is created to save the details of the user. The data is saved on the user's device and cleared when the user logs out. The attributes include the following.
- a. **User Name:** The name of the currently logged in user.
  - b. **User ID:** The unique ID of the currently logged in user.
  - c. **Email:** The email of the currently logged in user.
  - d. **Status:** The status of the currently logged in user. the possible value of this attribute is either "Active" or "Inactive".
  - e. **Firebase Token:** The unique firebase token for a particular device.
5. **HTTP Call Manager:** This class manages all the HTTPS calls made to the PHP web service, hosted on the server.
6. **Adapter:** This model defines how data from the web service are displayed unto the view.

### 3.5.1.2 The View

XML files will be used for designing the view of the android mobile application. The supported views of the application as per the functional requirement includes the following.

- a. Login View: This screen will allow a user to log into the application.
- b. Signup View: This screen will allow a user to register with the application.
- c. Dashboard View: After a successful login, this screen will display all the online accounts a user has synchronized with the monitR application if any.
- d. Account View: This screen allows a user to perform action for an online account. On this screen, the user can add security policy like adding devices or geofences.
- e. Map View: This screen will display a report of the login history for a user's online account.
- f. Settings View: This view will allow users to make basic settings on the app. Thus, a user can change his or her profile details, delete his account, set two factor authentications and change device status (from trusted to blocked or vice versa).
- g. Service Provider View: users will use this view to select from amongst the available service providers or online accounts that can be integrated with the monitR application.
- h. Dialogs: dialogs will present custom views to the user for minor functionalities.

Figures 3.4a – f show design screens for: *Sign In* – allows registered users to be authenticated to use the service provided by monitR; *Registration* – allows users to register to use monitR; *Dashboard* – landing page where users can synchronize their online accounts; *Map* – which provides map interface showing login locations; *Add Device* – allows users to add device to list of recognized devices or not and; *Take Remote Action* – allows users to remotely log out a session or block device.

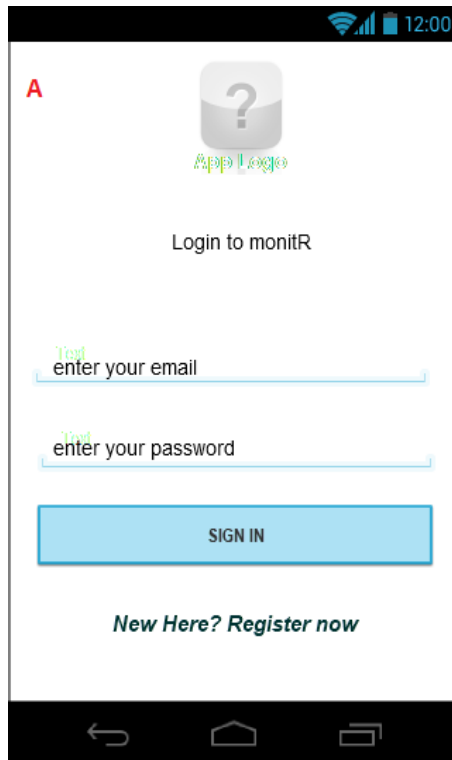


Figure 3.4a Login screen.

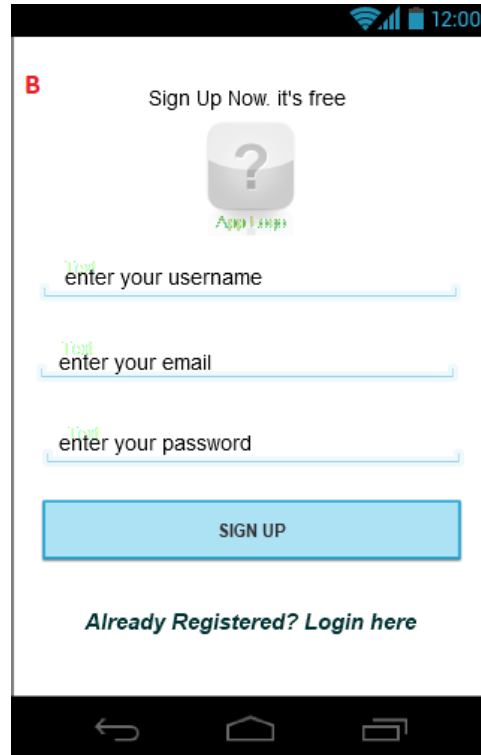


Figure 3.4b Registration Screen.

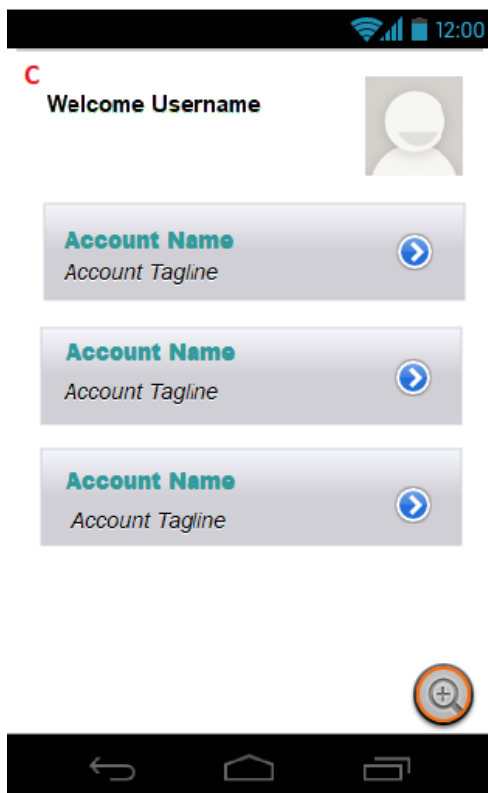


Figure 3.4c Dashboard Screen.

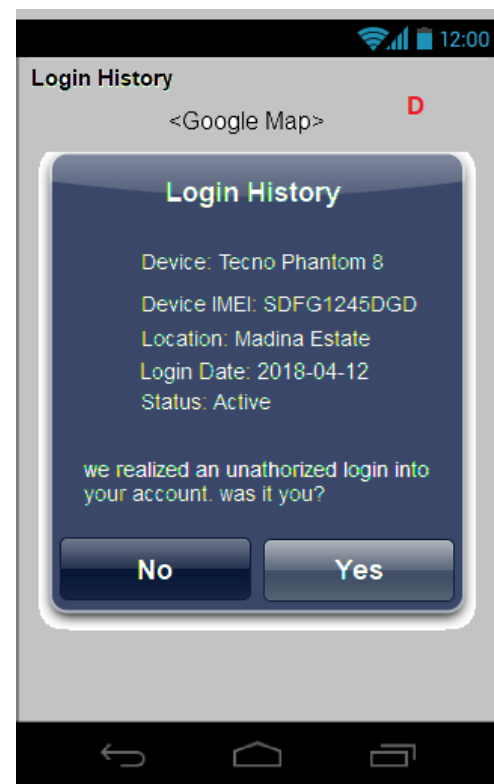


Figure 3.4d Map Screen with Login Dialog.



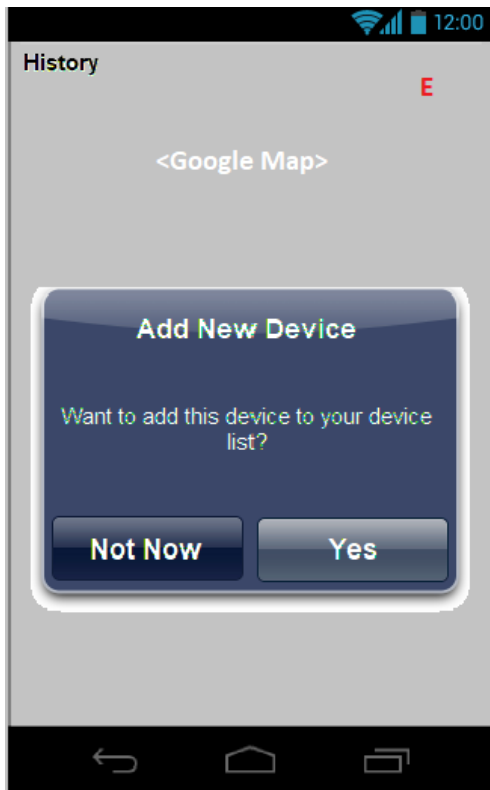


Figure 3.4e Screen for Adding A Device.

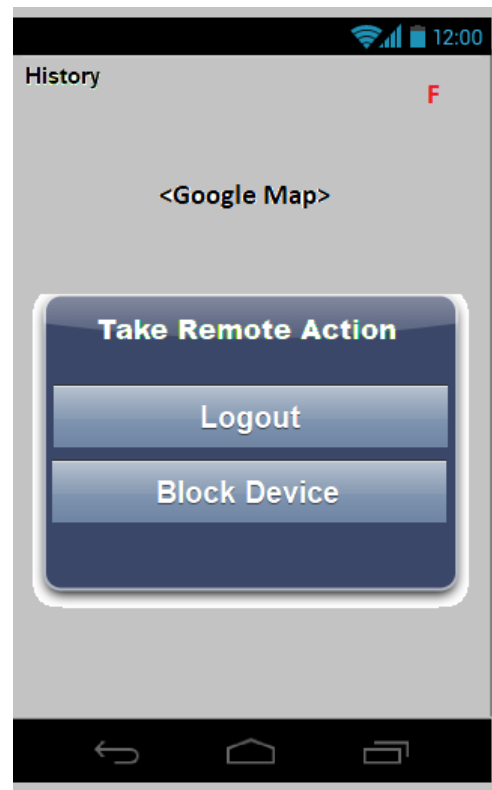


Figure 3.4f Remote Action Screen.

### 3.5.1.3 The Activity

This component of the user interface design is responsible for binding the data from the model to the view. All major screens as shown above, will have a corresponding activity, which will be the activity's controller. In addition to this, sub components called fragments, which are smaller activities, will be created to support the implementation of the functional requirement.

### 3.5.2 Web Service Module

The Web Service module is implemented using PHP. It is the main brain or business logic of the entire system. For that reason, any action or functionality that requires sending or receiving data to and from the database will have to pass through this module. The module will provide a channel of communication mainly via API endpoints. The API endpoint will be accessed by both the android application and the service provider(s) system. The web service module functions to:

- 1) Provide a communication endpoint for all client devices and service providers.
- 2) Accept user request, process it and send response to client.
- 3) Retrieve data from the MySQL database and send it to the android application.
- 4) Provide security for communication channel.

Figure 3.5 below is the class diagram for the web service.

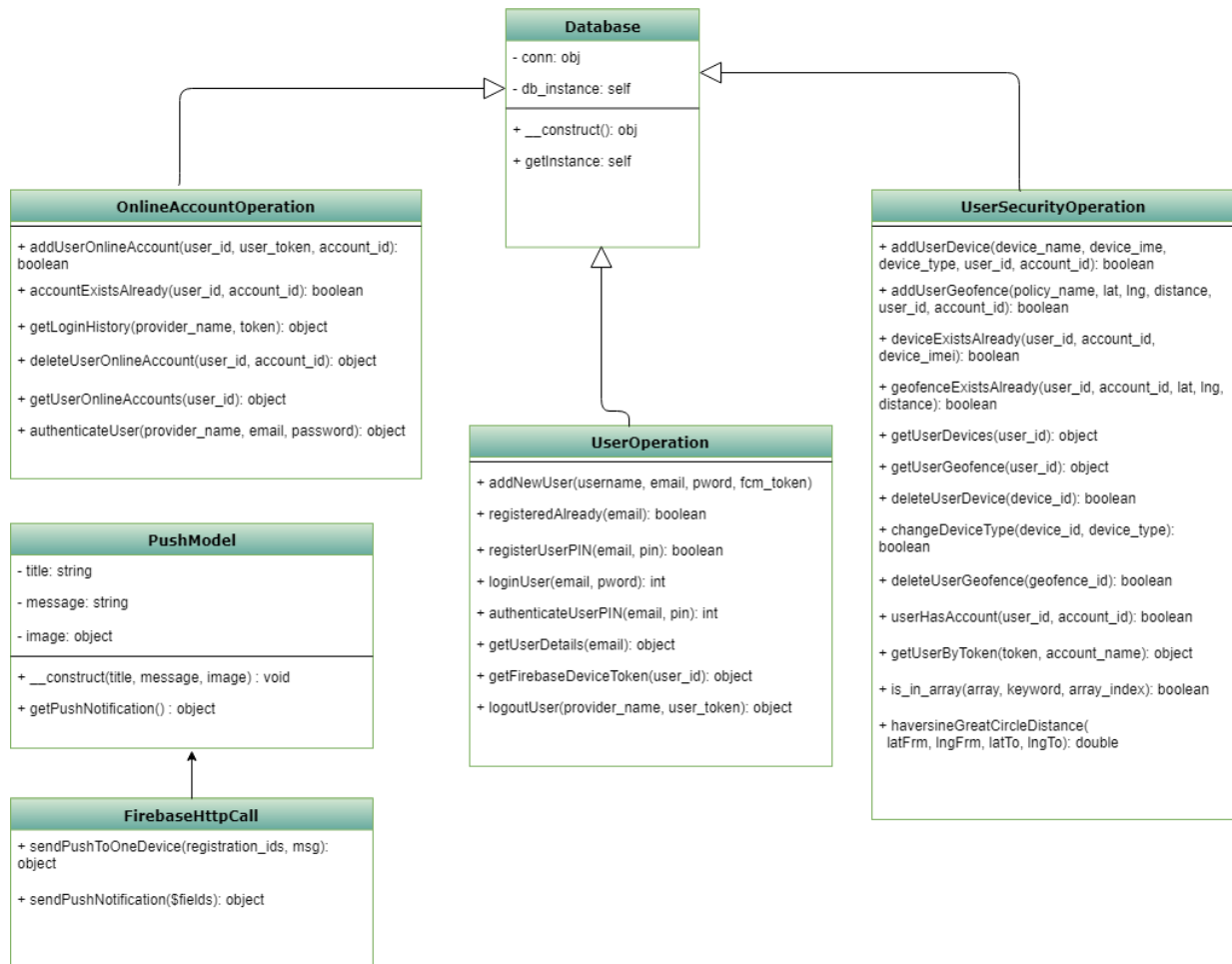


Figure 3.5 Class Diagram of the Application Layer (Web Service)

### 3.5.3 Login Anomaly Detection Module

This module will contain an implementation of algorithms that detects suspicious login attempts. It will mainly be accessed by monitR and service providers. The login anomaly module functions to prevent unauthorized use of a user's online account—the aim of this project.

#### **3.5.4 Service Integration Module**

The Service Integration Module will define set of HTTP protocols and API endpoints that will ensure a seamless communication between the MonitR application and external systems. This module will be implemented using PHP curl. CURL is a default PHP HTTP client manager that helps to make requests to an API endpoint and receive response thereof. There will be an agreement between external systems and monitR with regards to the HTTP method to use (whether GET or POST), and the required parameters that will be passed to the API URL. The agreement will also describe the format of the data returned (whether JSON or String or XML).

## Chapter 4: Implementation

### 4.1 Technologies and Tools Used

This section outlines the set of technologies, tools and development stacks used in implementing the proposed monitR application.

#### 4.1.1 Languages

**PHP:** PHP is the abbreviation for Hypertext Preprocessor. It is a server-side scripting language mostly used in developing web applications. PHP is simple to learn and can be used on all major operating systems. More to this, PHP supports a wide range of databases which can be integrated into web applications using the standard MySQL, PDO or ODBC (PHP-Manual, 2018). It also has a large online community which provides guidelines and support for developers, thereby reducing development time. Considering these features, PHP was used as the backend scripting language in this project to create RESTful APIs.

**Java:** Java is an open source high level object-oriented programming language developed by Sun Microsystems but currently traded under Oracle Corporation. Java is cross platform, meaning the compiled code can run on different operating systems. Currently, Java is one of the languages that powers most mobile devices and software systems. The android mobile application of this project was fully developed using Java. Java was used because of the large online support and rich documentation.

**MySQL:** The database layer of the project was implemented using MySQL. MySQL is a relational database management system It was used because it is simple to use, fast, free and open source.

#### 4.1.2 Libraries

**Volley HTTP Client:** Google's Volley HTTP library was used to handle all HTTP requests from the client side to the PHP web service module. Volley is used to transmit data over a network. This network library was chosen over other libraries like AsyncTask and

Retrofit because of its lightweight design, simplicity and automatic scheduling of request (Android-Developers, 2018). One of the most significant feature that informed my decision is the library's default caching system. Volley caches data unto the memory thereby saving memory and bandwidth. Additionally, this library was chosen because of its support for processing JSON object without the need for any third-party library.

**Android Material Design Library:** The Google's Material design library was used in this project because of its nice user interface and view components. Components like navigation drawer layout, recycler view, action toolbar, swipe to refresh and material colours were used to enhance the user experience of the application.

#### 4.1.3 APIs

**Google Maps Android API:** To provide interactivity based on location data fetched from the server, Google Maps Android API was used to embed a map into the application. This way, users can know their location on the map while interacting with the application. The API is free and has support for geocoding services (Google-Developers, 2018).

**Firebase Cloud Messaging API:** Firebase Cloud Messaging is a cross platform messaging solution that is used to send push notifications to client devices at no cost (Firebase Cloud Messaging, 2018). With firebase cloud messaging, one can send messages to a single device, a group of devices or to devices that have subscribed unto specific topics. Firebase was used in this project to send push notifications to client device anytime a suspicious login attempt is detected. It was chosen over SMS and E-mails because it is free of service.

#### 4.1.4 Tools

**Postman:** Postman is an API development environment. It is a platform used to design, debug, test and monitor APIs. It provides an interactive GUI for making requests to a web service and displaying the response. In this project, Postman was used to test all the RESTful web services developed.

**Android Studio:** Android Studio is the official IDE for developing native android applications. The monitR app was developed using android studio.

**000webhost Apache Server:** To be able to successfully process HTTP requests from the client side, 000webhost server was used to host and process user requests. It is a platform that supports development, hosting and deployment of PHP and MySQL web services.

**PhpMyAdmin:** PhpMyAdmin is a relational database management system used to handle the administration of MySQL over the web. It is a simple lightweight web application that supports wide range of operation on MySQL and MariaDB databases. It was used to manage the SQL operations.

## 4.2 Description of Components

Included in this chapter is an overall description of the components used in the application. These includes:

- i. **Online Account Integration:** This component defines set of interfaces that makes it possible to integrate an online account into monitR. Sub systems that make up this component includes:
  - a. **Service Provider Integration:** This sub component handles all processes that allows a given service provider to integrate its services into the monitR application. For a successful integration, both parties thus, monitR and a service provider, agree to choose a desired means of communication protocols mainly through HTTP. In this project, communication between monitR and a service provider was carried out through API calls. The service provider's account is activated once the integration is successful.
  - b. **Client-Side Integration:** This sub component is responsible for creating an interface for users to synchronize their online account into the monitR platform. The actors involved are mainly monitR and its users. In this component, the user

synchronizes an online account (say a shopping account) unto the platform by providing the same login details in the exact manner the user signs in into his or her online account (say his shopping account). When the correct login credentials are provided, this component connects to the sub component above, which makes an API call to the service provider to authenticate the user. Once authentication is successful, the user is alerted and monitR monitors the synchronized account, which is another component to be discussed below.

- ii. **Monitoring Login Activity:** The success of this project depends on this component—to reliably and efficiently monitor a user’s online account. The login activity monitoring component is responsible for keeping track of a user’s online account, detecting suspicious login attempts and notifying users of their account status. In this component, an effort is provided to safeguard a user’s account from fraudster and unauthorized users. The objective is to mitigate unauthorized login attempts. To effectively monitor a user’s online account, this component employs a combination of algorithms to achieve its objective. The algorithms and techniques used in this module are device identification techniques and Haversine’s great circle distance algorithm.

- a. **The device identification** technique is used to uniquely identify a user’s device being it a mobile phone or a tablet and label it as trusted or untrusted. At the time a user sends a login request to any of the accounts he or she has synchronized with monitR, the service provider of that account, through an API call, redirects the login request to monitR for security checks. monitR then examines the device name, device IMEI, the operating system, the location and time from the API parameters and checks it against the user’s security policies. If the device information does not match the profile or security policies of the user, monitR send a firebase push notification to the user to take remote actions.

Additionally, monitR notifies the service provider with an API call. The service provider either allows or denies the login request from the user after cross checking the security policy of the user with monitR. Through this, the unauthorized login can be detected even though the attacker provides the correct login credential.

- b. **Geofencing Detection using Haversine's Algorithm.** In addition to the device identification technique, geofencing policy was used to provide an extra security layer to the system. In this sub component, Haversine's great circle distance was used to detect whether the login request is within or outside the specified geofencing policy set by the user. The Haversine algorithm calculates the great circle distance between two points on a sphere given their longitudes and latitudes. The result from Haversine's formula is compared with the geofence policy set by the user. If the login request is within any of the geofence set by the user, it is tagged as a trusted login else a suspicious login attempt.

A combination of the two techniques highlighted in this component is used to detect anomaly login attempts. Though they are both used in the system, the device identification technique however takes precedence over the geofencing technique.

- iii. **Taking Remote Action:** This component handles all processes that enable a user to take remote actions anytime a suspicious login is detected by the system. The component is broken down into the sub systems as follows.

- a. **Remote Logout:** This sub component allows a user to remotely terminate the session or log the unauthorized user out of the compromised account. The action taken by the user is sent to the service provider of that account through an API



call. The service provider then in real time, terminates the session and the attacker will be logged out wherever he or she is.

- b. **Remote Blocking of Devices:** Aside terminating the session, users can still block the unauthorized device from accessing their account. When this action is taken, monitR will flag a device as untrusted and will be added to the list of user's blocked devices. Any device in the blocked list will no longer have access to the user's online account.
- iv. **Support for User Security Policies:** One of the key objectives of the project is to provide support for users to manage their security policies for an account. This component achieves such an objective. In this module, users can add their device into their trusted devices list, which will later be used by monitR during login anomaly detection. Additionally, users can set a geofence to secure their online accounts. The geofencing feature takes in the GPS location and the distance of that place and sends it to the web service for further processing.
- v. **Notification services:** This component is responsible for sending push notifications to users anytime a suspicious login attempt is detected. It allows users to know the status of their online accounts. When a user is registering with monitR, the device he or she uses is identified, and a firebase unique token is generated for that device, which is sent to the server. Subsequent notifications will be sent to the user through this device.

#### **4.3.1 Implementation Techniques and Process**

MonitR is a third-party security app that monitors online account of users. This suggests that its security implementation is non-negotiable. That said, security has played integral role at each layer of the application's design and development.

At the design stage, the system was divided into three main layers namely the front-end layer, application layer and database layer. Each layer comprises a number of components, which were further broken down into sub tasks. Overall, the software development practice that was used in this project is the test-driven approach. This is when the developer interleaves testing whiles developing at the same time.

The system was developed using a bottom-up approach, when considering the three layers. The database layer was thus the first layer implemented. The layer used MySQL database to manage user data. Because security is keen to this project, all database operations were implemented using prepared statements (Listing 4.1), which guards against security breaches and SQL injections. Unit and component testing were carried out to ensure the functions are working as expected.

*Listing 4.1 MySQL Prepared Statement Usage*

```
/**
 *a function to add new user into app_user table (registering a user)
 *@param $username user first name
 *@param $email user email
 *@param $pword user password
 *@param $fcm_token firebase device token
 *@return retruns true if registration was success and false otherwise
 */
public function addNewUser($username, $email, $pword, $fcm_token){
    $db = Database::getInstance();
    //hash password for security purpose
    $hashedpword = password_hash($pword, PASSWORD_DEFAULT);

    $status = 'Active';
    try{
```

```

        $stmt = $db->prepare("INSERT INTO app_users (Username, Email,
        Password, Status, Firebase_Device_Token) values (:uname, :uemail,
        :upass, :status, :fcm_token)");

//bind params to data
$stmt->bindParam(":uname",$username);
$stmt->bindParam(":uemail",$email);
$stmt->bindParam(":upass",$hashedpassword);
$stmt->bindParam(":status",$status);
$stmt->bindParam(":fcm_token",$fcm_token);
$stmt->execute(); // execute query
if($stmt->rowCount() > 0){
    return TRUE;
}
} catch (PDOException $e) {
    return FALSE;
}
}

```

After the database layer, the application layer (web service) was the next layer implemented. This layer is made up of RESTful APIs developed with PHP. The function of the layer is to accept requests from users, process the requests and give response thereof. Again, because of security reasons, all API endpoints can be accessed only through the HTTP POST method instead of the GET method. The POST method is more secured than the GET method. Listing 4.2 shows the PHP code for accepting POST requests. APIs were tested using postman to ensure the correct JSON data is received.

*Listing 4.2 Usage of API Accepting Only POST Requests.*

```

<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if(isset($_POST['email']) && isset($_POST['password'])) {
        $email = $_POST['email'];
        $password = $_POST['password'];
    }
}

```

```

        //create new UserOperation object
        $loginObj = new UserOperation();
        $loginResult = $loginObj->loginUser($email, $password);
    }
    // when GET server request is made
    else{
        $response['error'] = true;
        $response['message'] = 'Cannot Handle GET Request';
    }
    ?>

```

The last layer implemented was the front-end layer, which is the monitR mobile app. The android operating system was chosen because there is a direct access to the device's operating system's resources. Stated differently, there are no plugins to be installed before one can access some resources say camera, as it is in the case of hybrid mobile applications. At the front-end layer, all major components were identified and broken into sub tasks. Still on security, a user is required to provide password or 6-digit PIN in order to access the system.

#### **4.3.2 Implementation Challenge**

The greatest challenge of this project was the inability to get a service provider to integrate their system unto our platform. Efforts were made to even get their sandbox API for testing purposes but as it stands, no company was willing to give out their APIs. This was primarily because they do not want to give out their resources for school project and, they wanted the app to be fully functional before they could even consider giving their sandbox APIs for testing. To overcome this challenge, sample mobile applications were created that could feed monitR with live information just as any service provider would do. Two (2) sample applications were developed to that effect: were Shoppn – a shopping application and BankeX, an online a banking application.

One other major challenge was the choice of technologies. This project requires functionalities such as push notifications which implementation was first time and as such

presented a hurdle. However, tutorials from YouTube videos and Simplified Coding proved very helpful (Khan, 2018).

## 4.4 Evidence of Implementation

This section includes snapshots of the implemented system.

### 4.4.1 User Registration

Figure 4.1 shows the registration screen, which allows a user to sign up or register unto the application. The registration page requires users to provide a username, email address and password. The details are processed at the server and the user is given response whether registration was successful or unsuccessful.

### 4.4.2 User Login

Figure 4.2 shows the login screen that allows a user to login to the monitR application. The user provides his or her email address and password, and when successfully authenticated, is granted access to the application.



Figure 4.1 Sign-Up Screen of monitR

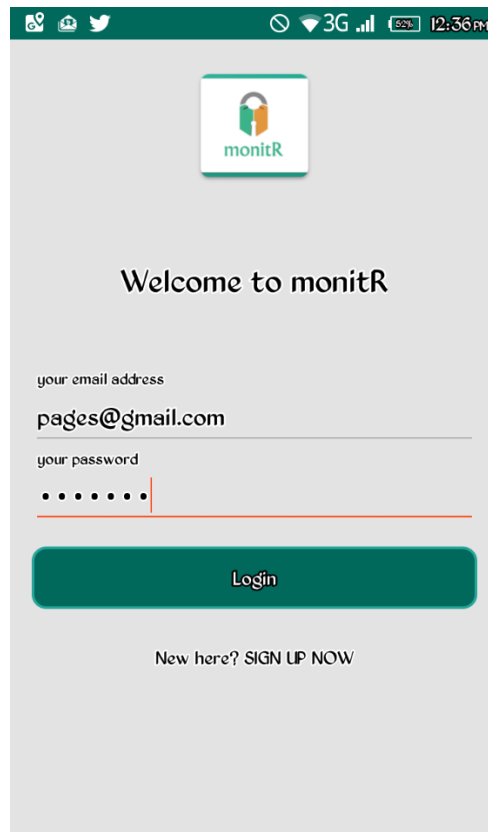


Figure 4.2 Login Screen of monitR

#### 4.4.3 User Dashboard

This screen displays all the accounts a user has integrated with the monitR. Accounts are wrapped in a card view to enhance user experience. If no account is synchronized, a default page is displayed asking users to synch their account(s). Figure 4.3 below shows the dashboard screen, displaying BankX and Shoppn accounts synchronized with monitR.

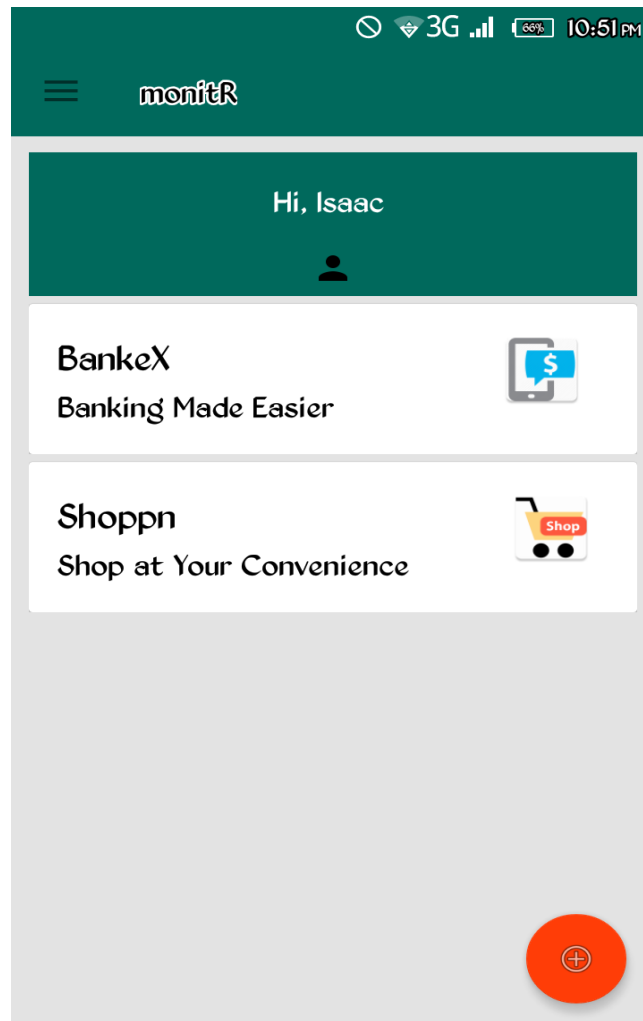


Figure 4.3 Dashboard Screen

#### 4.4.4 Service Providers and Account Sync

Figures 4.4a show a list of service providers whose applications are integrated with monitR. Once a user selects a service provider, they are redirected to the service provider's login engine to be authenticated and validated as the legitimate user.



Figure 4.4(a) List of Integrated Applications.

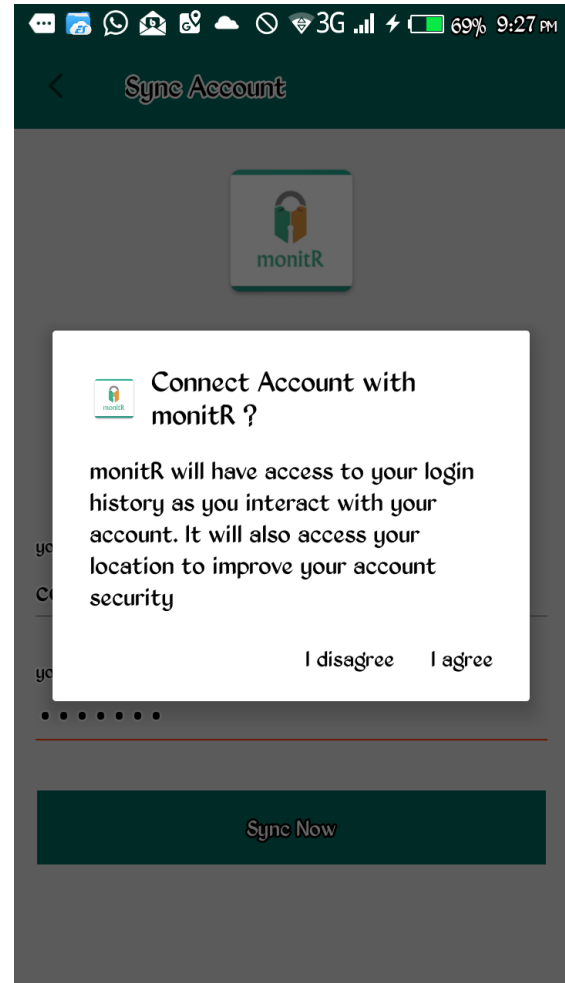


Figure 4.4(b) Rules for Syncing Accounts.

#### 4.4.5 Login History Map View

Figure 4.5a displays an account login activity in an interactive map. The user is informed of where his or her account is logged in from. Details of the login activity can be displayed by clicking on the red markers on the Map. The details are shown in Figure 4.5b.

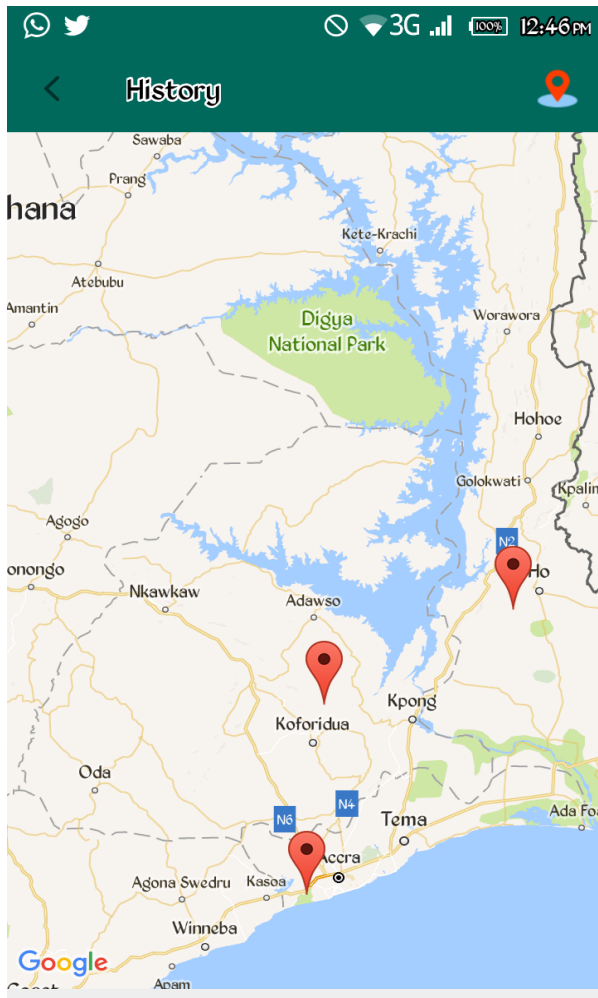
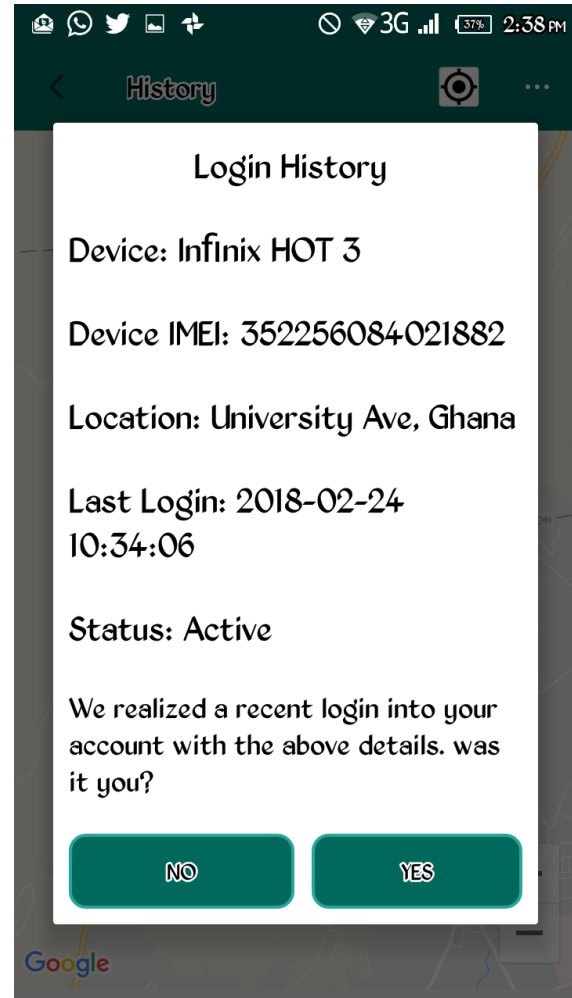


Figure 4.5 (a) Account Activity Map.



4.5 (b) Details of Login Activity.

#### 4.4.6 Remote Action View

Figures 4.6a and b allow a user to take remote action when there is a suspicious login activity based on the information from figure 4.5. The user can then add the device to his or her white listed devices if he recognizes the login activity, remotely logout the device or blacklist that device from obtaining subsequent access to the account.



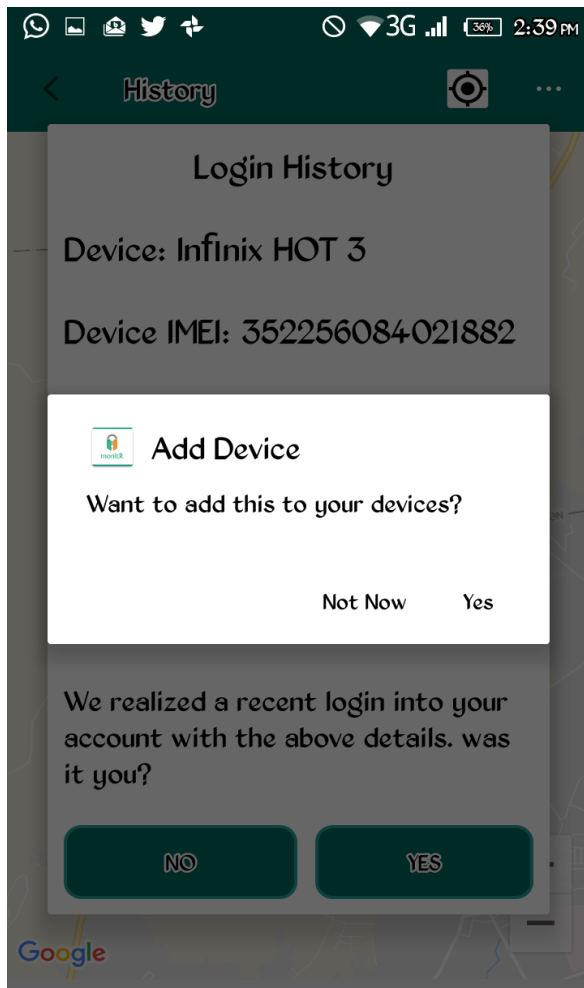
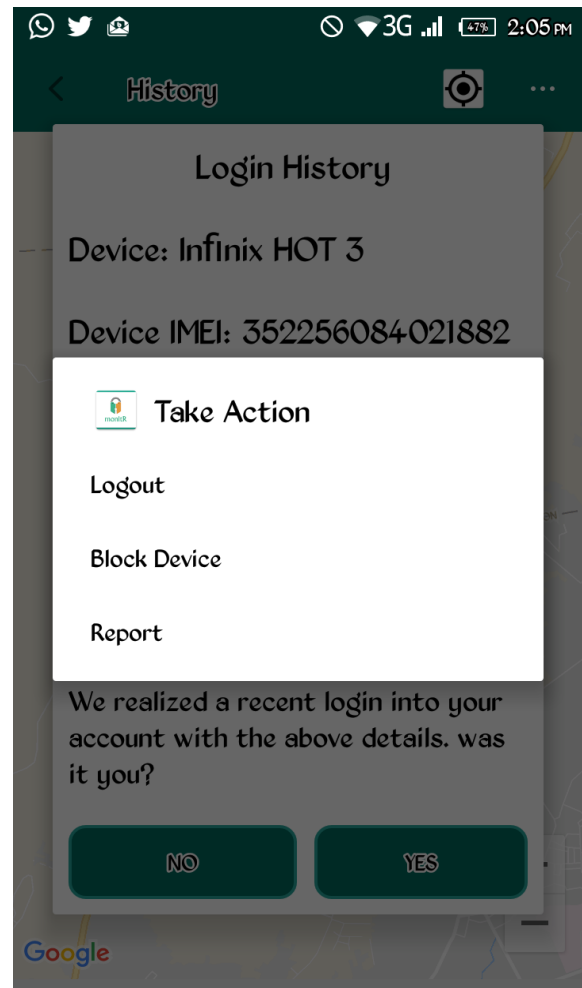


Figure 4.6(a) Dialog for Adding a Device



(b) Dialog for Taking Action Remotely

#### 4.4.7 Geofence View

To restrict login access to specific geographical locations, the geofence feature displayed in Figure 4.7 is implemented. This security feature will enable a user to set a geofence around an area of interest. Upon setting a geofence, the GPS location of that area is sent to the server and stored for forensic analysis.

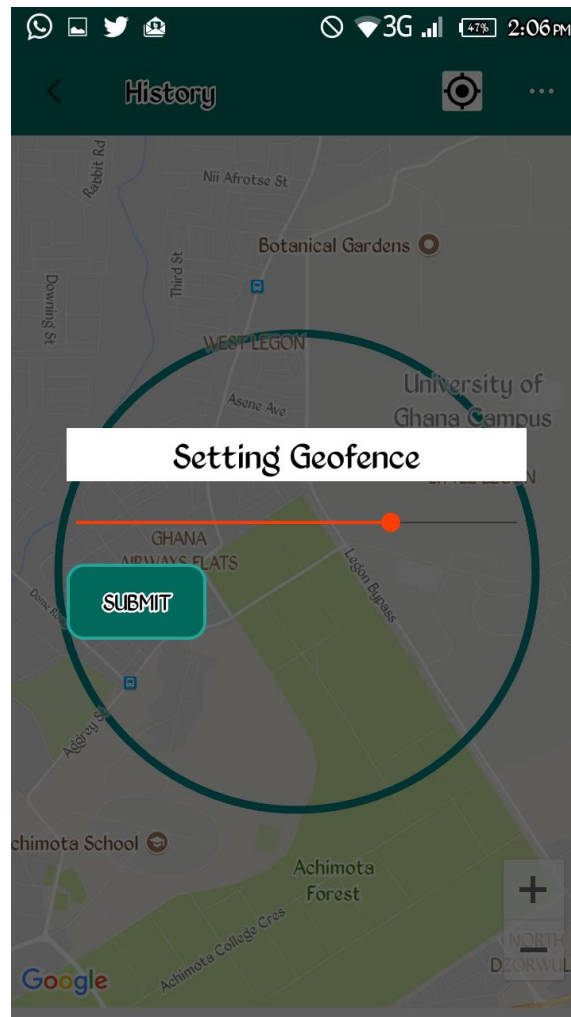


Figure 4.7 Adding a Geofence

#### 4.4.8 Manage Devices View

Figure 4.8 shows the screen for managing what device has access to the user's account. All will be used by the user to manage any previously added or blocked device; by toggling between disabling a whitelisted device or enabling a previously blacklisted device. The user can switch between unblocking a blocked device to blocking a trusted device.

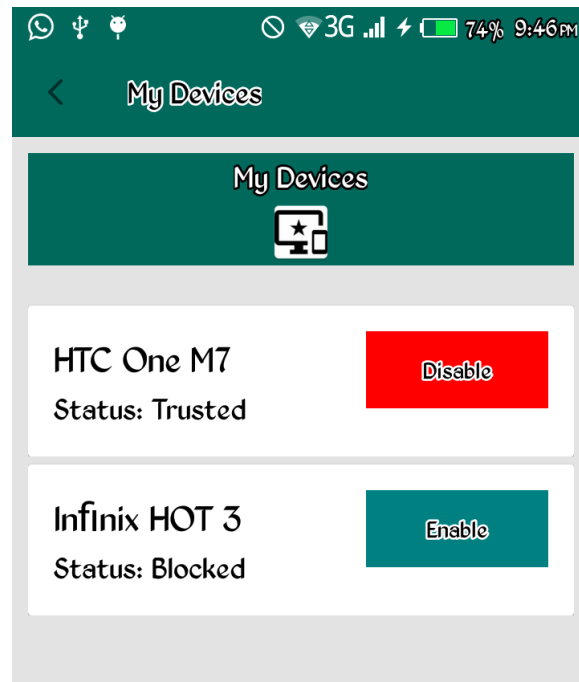


Figure 4.8 Screen for Managing Synched Devices

#### 4.4.9 Push Notification Alert

When the system detects a suspicious login attempt, a push notification is sent to the user with information about which account is being compromised. The notification message prompts the user to take remote action when a suspicious login attempt is encountered. Figure 4.9 below shows a notification received when HTC One M7 tried logging into Bankex.

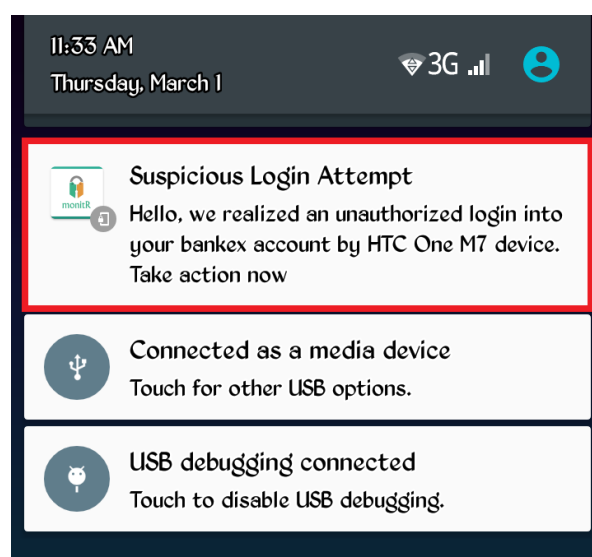


Figure 4.9 Push Notification Message Received

## **Chapter 5: Testing and Result**

This chapter highlights the techniques and processes involved in testing the monitR software. The essence of testing is to validate the functional and non-functional requirements as described in chapter 2 of this software document. In this project, two major tests were carried out: development test and user acceptance test. Included in the subsequent section of the chapter, is a detailed report of test cases, test results and analysis of test results.

### **5.1 Development Test**

Development testing is a type of testing that is done at the development stage of the software development life cycle to discover bugs and defects. This testing is performed to ensure the software is correct, meets standards, and free from errors and bugs. Development testing is sub-divided into three categories: unit testing, component testing and system testing. Below provides the test cases and results for each of the test types.

#### **5.1.1 Unit Testing**

Unit testing is a way of testing individual components of the system such as a function, method or a class. Unit testing helps to check the functionality of individual objects and functions of monitR to ensure user operations associated with each object produce the required results when given valid, invalid and null parameters. PHPUnit testing framework was used to conduct unit testing. Below is a code snippet of the unit test for user registration and login functionality.

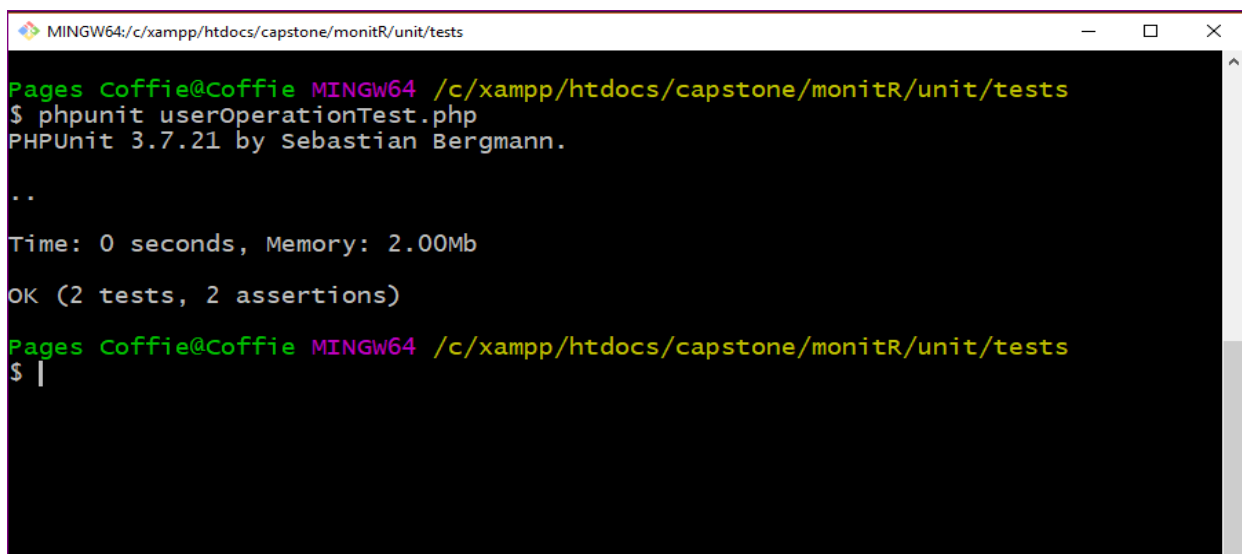
```

1 <?php
2 include_once("C:/xampp/htdocs/capstone/monitR/model/userOperation.php");
3 /**
4  * @Author Isaac Coffie
5  * Date: 19/03/2018
6  */
7 class UnitTests extends \PHPUnit_Framework_TestCase{
8
9     //test registration function
10    public function testUserRegistrationReturnsTrue()
11    {
12        $user = new UserOperation();
13
14        $this->assertTrue($user->addNewUser("Pages", "pagescoffy@gmail.com", "mypass999", "
15            someLooongFirebaseDeviceToken"));
16    }
17    //test login function
18    public function testUserLogin()
19    {
20        $user = new UserOperation();
21        $email = "pagescoffy@gmail.com";
22        $password = "mypass999";
23        $this->assertEquals(1, $user->loginUser($email, $password));
24    }
25
26 }

```

Figure 5.1 PHPUnit test of the UserOperation model.

This code snippet tested the user registration and login function. Figure 5.2 below indicates the two test cases both passed testing.



```

MINGW64:/c:/xampp/htdocs/capstone/monitR/unit/tests
Pages Coffie@Coffie MINGW64 /c:/xampp/htdocs/capstone/monitR/unit/tests
$ phpunit userOperationTest.php
PHPUnit 3.7.21 by Sebastian Bergmann.

..
Time: 0 seconds, Memory: 2.00Mb
OK (2 tests, 2 assertions)
Pages Coffie@Coffie MINGW64 /c:/xampp/htdocs/capstone/monitR/unit/tests
$ |

```

### 5.1.2 Component Testing

This is a method of testing where individual component or module of the software system is tested separately. The aim is to find defects in the module and verify the functionalities of the software component. Additionally, component testing plays a key role in detecting bugs in software component before integrating it with other components. The following are examples of test cases for some selected components of the software as explained in chapter 3 and 4.

#### 5.1.2.1 Test Case: Add user online account into monitR.

**Precondition:** The online account that the user wishes to integrate into monitR must exist. This is done during system integration between the service provider and monitR. Additionally, the user's account must be active.

**Expected Result:** A user online account record will be created in the database for the user. A JSON response will be sent with the required message. Upon a successful response, the user can now receive notifications and can take remote actions on that account.

#### **Test Result:**

Figure 5.3 shows a test result for adding a user's online account with a valid data (userId, accountId, providerName, email and password). The result was a JSON data with the following content (Listing 5.1):

```
{  
  "error": false,  
  "message": "Account Successfully Synchronized"  
}
```

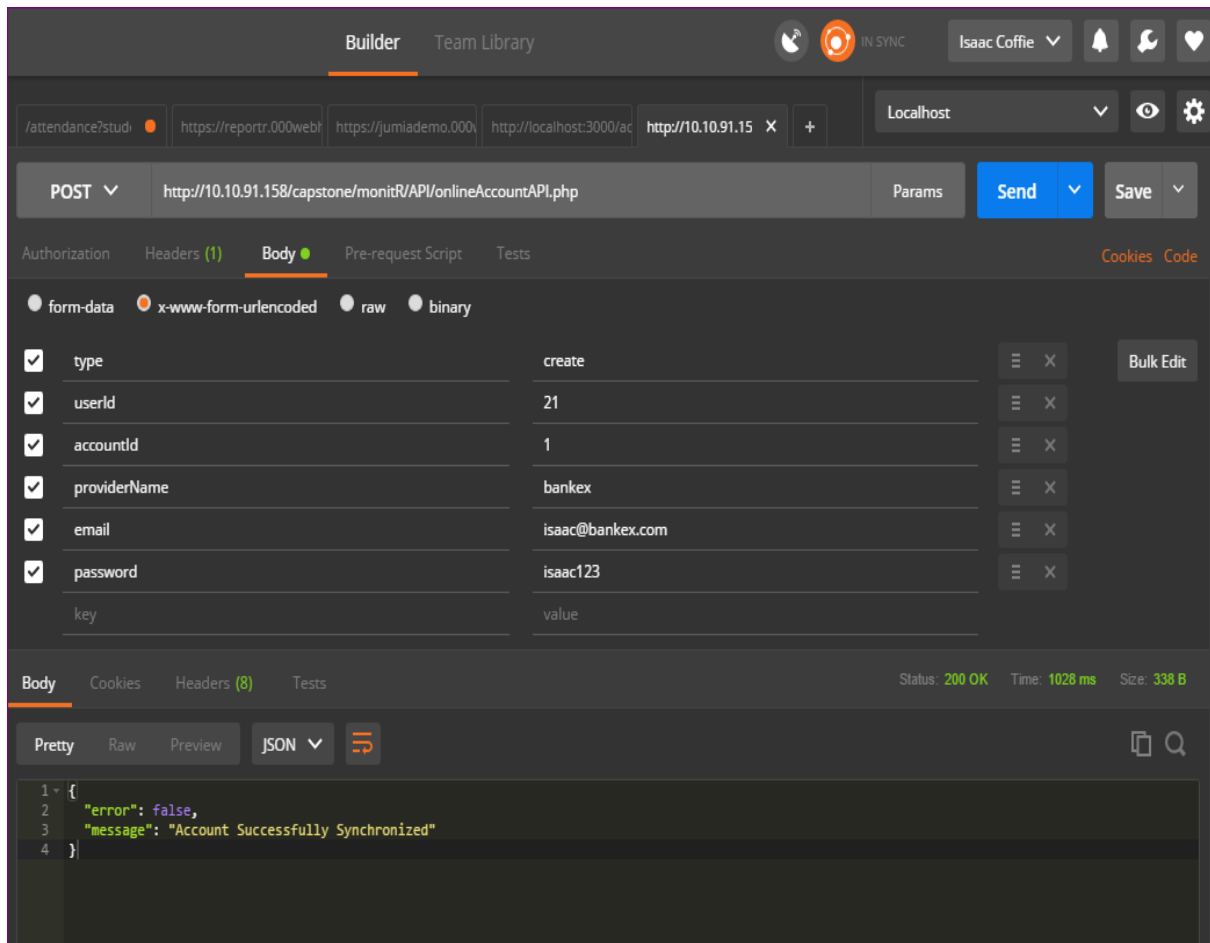


Figure 5.3 Postman test of the synchronize user online account component.

Listing 5.2 shows a test result for adding a user online account with an invalid data. In this scenario, the user with unique identifier of 21 tried to synch his “Shoppn” account with monitR. However, he entered the wrong password to his account (coffie@shoppn.com) and the result was a JSON data with the following content:

```

{
  "error": true,
  "message": "Incorrect Password"
}

```

### 5.1.2.2 Test Case: View user online account.

**Precondition:** The online account that the user wishes to integrate into monitR must exist and the user's account must be active.

**Expected Result:** A JSON data of all online accounts associated with a particular user.

#### **Test Result:**

Listing 5.3 as shown below is a test result for viewing a user's synchronized account with a valid data (userId = 21). The result was a JSON data with the following content:

```
{
  "myaccounts": [
    {
      "Account_Name": "BankeX",
      "Account_Tagline": "Banking Made Easier",
      "User_Id": "21",
      "User_Token": "9",
      "Online_Account_Id": "1",
      "User_Online_Account_Id": "22",
      "Status": "Active",
      "Date_Created": "2018-03-20 08:35:41"
    }
  ],
  "error": false,
  "message": "Online Account(s) Found"
}
```

Listing 5.4 shows a test result for viewing a user's synchronized online account with an invalid data. In this case the userId was incorrect and the result was a JSON data with the following content:



```
{  
  "error": true,  
  "message": "No Account(s) Found"  
}
```

### 5.1.3 Integration Testing

Integration test focus on testing the integration of interface between software components. It helps to ensure a seamless interface between one software component and the other(s). Below shows a test cases between a service provider (Bankex in this case) and monitR. This scenario tests the interface between the login module of the service provider and the anomaly detection engine of monitR. The test is simulated on postman test environment.

#### 5.1.3.1 Test Case: Login anomaly detection from Bankex

This determines if a login request from a user should be granted or denied by Bankex.

**Precondition:** The user has an account created with bankex. The user has synchronized his or her Bankex account with monitR.

**Expected Result:** A JSON response from monitR indicating whether the login is trusted or fraudulent after checking the security policy of that user who sent the login request.

**Test Result:**

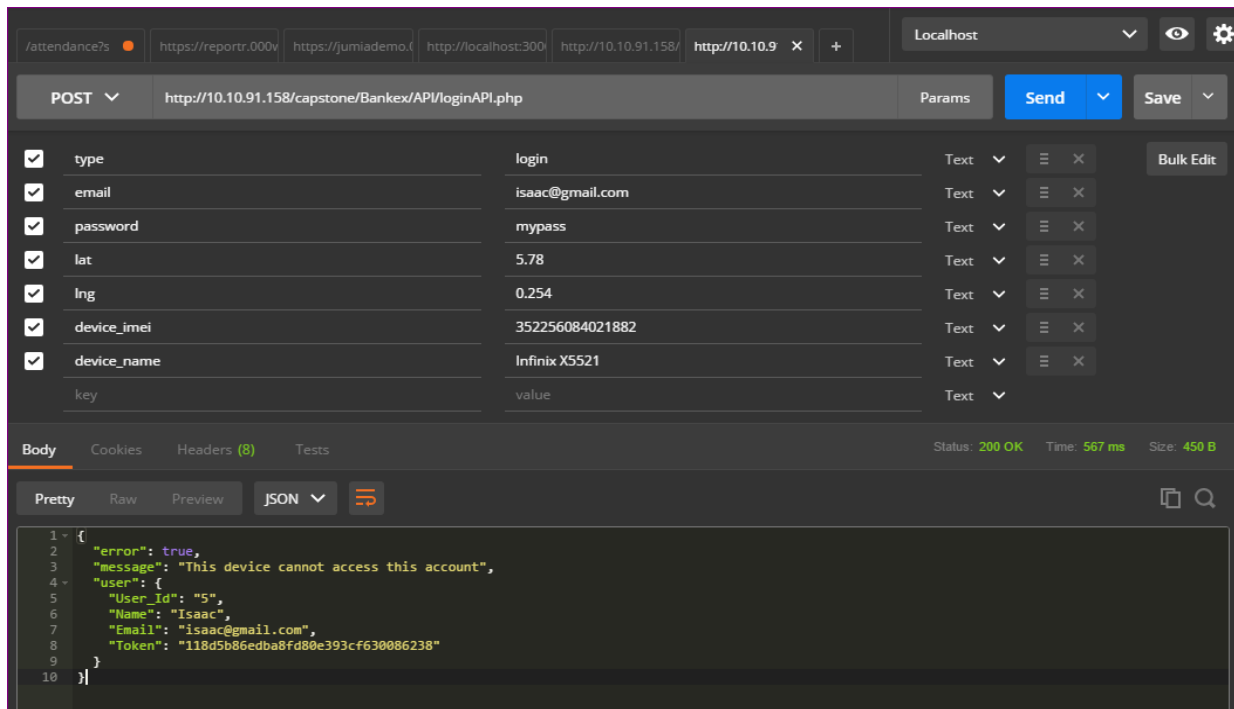


Figure 5.4 Test result of checking login request with valid data (email, password).

The result was a JSON data (Listing 5.5) with the following content:

```
{
  "error": true,
  "message": "This device cannot access this account",
  "user": {
    "User_Id": "5",
    "Name": "Isaac",
    "Email": "isaac@gmail.com",
    "Token": "118d5b86edba8fd80e393cf630086238"
  }
}
```

This test case passed because Bankex’s login module successfully interfaced with monitR’s login anomaly detection module. From the response above, this login request will be flagged as suspicious as the security policy (device identification and geofencing) of the user did not match the login request.

#### **5.1.4 System Testing**

This type of testing involves integrating the entire software components to create a system. The objective is to verify that the developed software meets specifications and purpose. It also investigates both the functional and non-functional requirements specified in chapter 2. To test the monitR application, all the components of the software were integrated to form a software application. The testing was done at the user interface layer, where the monitR mobile application was deployed for use. As stated in chapter 4, there was a challenge of getting APIs from a real service provider. To overcome that challenge, two mobile applications namely Bankex and Shoppn were created to feed data to the monitR software. Ashesi students installed and tested the mobile applications on their device. The results from the test cases indicated that the applications were able to communicate among each other. This means a logout operation from monitR was in real time, reflected in the other applications (Bankex and Shoppn).

## **Chapter 6: Conclusion and Recommendation**

### **6.1 Conclusion**

This paper contributed towards the body of information security by providing a revolutionary solution, yet a proactive approach to monitoring user's online account in wake of internet insecurity. To this end, a native Android mobile application, monitR, was developed to help online users take control over their accounts. To make this possible, the monitR application defines a seamless interface for service providers to integrate their systems with the platform. With this solution, a user synchronizes his or her online account(s) unto the monitR mobile application, sets basic security policy and the system in return monitors the online account(s).

The research question, "how to detect and prevent unauthorized login attempts?" was answered by using two categories of security policies: device identification techniques and geofencing techniques. The device identification technique defines which device(s) should have access to an online account, and the geofencing techniques utilize Haversine's algorithm to determine whether a login request should be granted or rejected at a particular location.

This solution will not only detect suspicious login attempts but will mitigate it, notify online users and safeguard online users especially banking customers from hackers.

### **6.2 Recommendation**

Readers can in future implement the following features to improve the efficiency of the system.

- I. The system's ability to view all online accounts in one map interface. Currently, the system allows login activity of only one online account (say Bankex) to be viewed at a time. To make the system more robust and avoid multiple click events, all online accounts owned by a user which has previously been synchronized with monitR should be displayed in one big map with color codes.

- II. The system should allow users to decide whether a geofencing policy should be out or in. With the current implementation, anytime a user sets a geofence, the system assumes the geofence is to block login attempts from outside that area. However, there should be an option for the user to decide if he or she wants to allow login inside that geofence or outside that geofence.
- III. The system should be able to learn the login activity of a user. This can be done by utilizing machine learning algorithms to make the prediction more accurate and with ease.
- IV. The system should be extended to other platforms like iOS and the web. Currently, the application runs only on the Android operating system. It will be best if an iOS version or a web version is built.
- V. Improved user interface. The user interface as at the finish of this project fits well for the purpose. However, more works can be done to improve upon it. A user interface design that utilizes material design patterns is highly recommended.

## References

- Aiman, M. O., Dafa-Allah, A., & Elhag, A. A. (2017). Proposed security model for web based applications and services. *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)* (pp. 1-13). Khartoum, Sudan: IEEE.
- Android-Developers. (2018, February 24). *Transmitting Network Data Using Volley*. Retrieved from Android Developers Console: <https://developer.android.com/training/volley/index.html>
- Awiah, D. M. (2017, November 23). *Ghana loses \$50m to cyber crimes*. Retrieved from Graphic Online Newspaper: <https://www.graphic.com.gh/news/general-news/ghana-loses-50m-to-cyber-crimes.html>
- Axten, S. (2017, October 7). *Staying in Control of Your Facebook Logins*. Retrieved from Facebook Website: <https://www.facebook.com/notes/facebook/staying-in-control-of-your-facebook-logins/389991097130/>
- BelitSoft. (2017, November 1). *Software Requirements Specification Document Example International Standard*. Retrieved from Belit Soft Website: <https://belitsoft.com/php-development-services/software-requirements-specification-document-example-international-standard>
- DiGiacomo, J. (2017, November 29). *2017 Security Breaches: Frequency and Severity on the Rise* . Retrieved from Revision Legal: <https://revisionlegal.com/data-breach/2017-security-breaches/>
- Diwanji, P. (2017, October 7). *Detecting suspicious account activity*. Retrieved from Official Gmail Blog: <https://gmail.googleblog.com/2010/03/detecting-suspicious-account-activity.html>

Firebase Authentication . (2017, October 30). Retrieved from Google Firebase Website:

<https://firebase.google.com/docs/auth/>

Firebase Cloud Messaging. (2018, February 24). Retrieved from Firebase Console:

<https://firebase.google.com/docs/cloud-messaging/>

Google-Developers. (2018, February 24). *MapFragment*. Retrieved from Google Developers

Console:<https://developers.google.com/android/reference/com/google/android/gms/maps/MapFragment>

Hewamadduma, S. I. (2017). Detection and prevention of possible unauthorized login attempts

through stolen credentials from a phishing attack in an online banking system. *Research and Innovation in Information Systems (ICRIIS)* (pp. 1-6). Langkawi, Malaysia: IEEE.

Khan, B. (2018, February 11). *Firebase Cloud Messaging for Android using PHP and MySQL*.

Retrieved from Simplified Coding Website:

<https://www.simplifiedcoding.net/firebase-cloud-messaging-android/>

Khrais, L. T. (2015). Highlighting the Vulnerabilities of Online Banking System. *Journal of*

*Internet Banking and Commerce*, 1-13.

Morgan, S. (2017, September 16). *Top 2016 Cybersecurity Reports Out From AT&T, Cisco,*

*Dell, Google, IBM, McAfee, Symantec And Verizon*. Retrieved from Forbes :

[https://www.forbes.com/sites/stevemorgan/2016/05/09/top-2016-cybersecurity-](https://www.forbes.com/sites/stevemorgan/2016/05/09/top-2016-cybersecurity-reports-out-from-att-cisco-dell-google-ibm-mcafee-symantec-and-verizon/#38f509311caf)

[reports-out-from-att-cisco-dell-google-ibm-mcafee-symantec-and-](https://www.forbes.com/sites/stevemorgan/2016/05/09/top-2016-cybersecurity-reports-out-from-att-cisco-dell-google-ibm-mcafee-symantec-and-verizon/#38f509311caf)

[verizon/#38f509311caf](https://www.forbes.com/sites/stevemorgan/2016/05/09/top-2016-cybersecurity-reports-out-from-att-cisco-dell-google-ibm-mcafee-symantec-and-verizon/#38f509311caf)

PHP-Manual. (2018, February 24). *What can PHP do?* Retrieved from PHP Manual:

<http://php.net/manual/en/intro-whatcando.php>

Venter, H., & Eloff, P. (2003). A taxonomy for information security technologies. *Computers & Security*, 299-307 .