# Remote locks for homes/hotels operable by a smartphone.

## Contents

## Problem Statement

Access to buildings has traditionally been protect through the use of a mechanical lock and key. Although time tested, this method creates several problems. The first is need for the key itself which is in itself an additional item which the user must carry. The key also provides a single point of failure when the owner inevitably loses the key.

Additionally mechanical keys and locks lack the intelligence to overcome many of the challenges that are met by people in a modern society.

## Project Overview

To overcome the limitations of a mechanical key/lock system, I have developed a network aware door lock system. The system is composed of four components as shown below in figure one. Three of the components are connected to the Internet and will communicate in a server client relationship consisting of one server, one android based client, and one raspberry pi based client (here to for referred to as Android Client and Raspberry Pi Client respectively) which will control the fourth element of the system which is a mechanical door lock. The primary goal of the project is to allow a user to interface with the android client and determine the status of the lock (locked or unlocked) from any point on earth which has network connectivity. The user may than at their discretion change the state of the lock i.e. remotely lock or unlock the door.

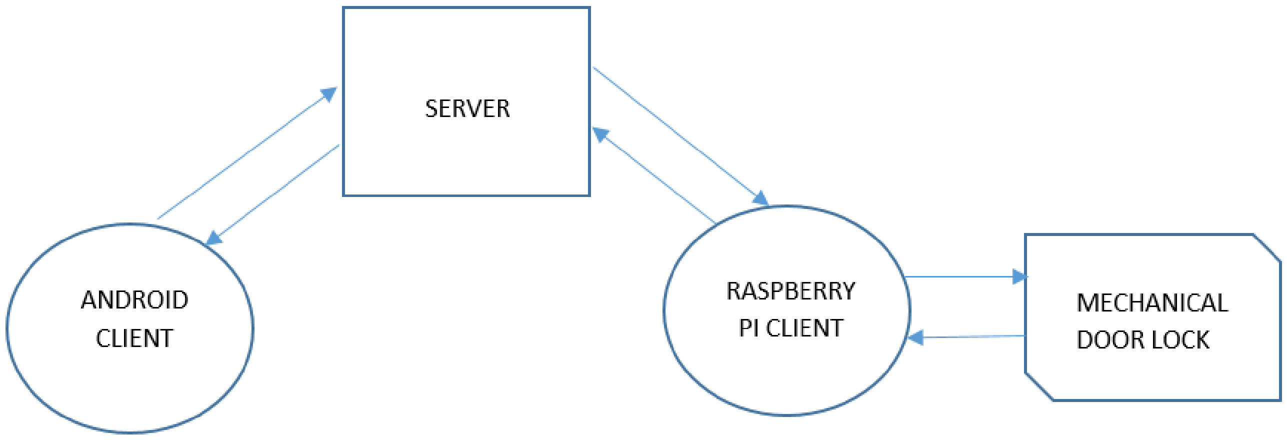


Figure 1.

Additionally, functionality has been added to the Android client to make the device location-wise aware. In this way, at the user's discretion, the device is disabled when the Android client is more than a predetermined distance from the Raspberry Pi client. By including this feature, a user ensures that a stolen Android device will be unable to lock or unlock the target lock unless the malicious user is in possession of the device and knows its location as well, and has moved within the predetermined radius.

## Extra Feature:

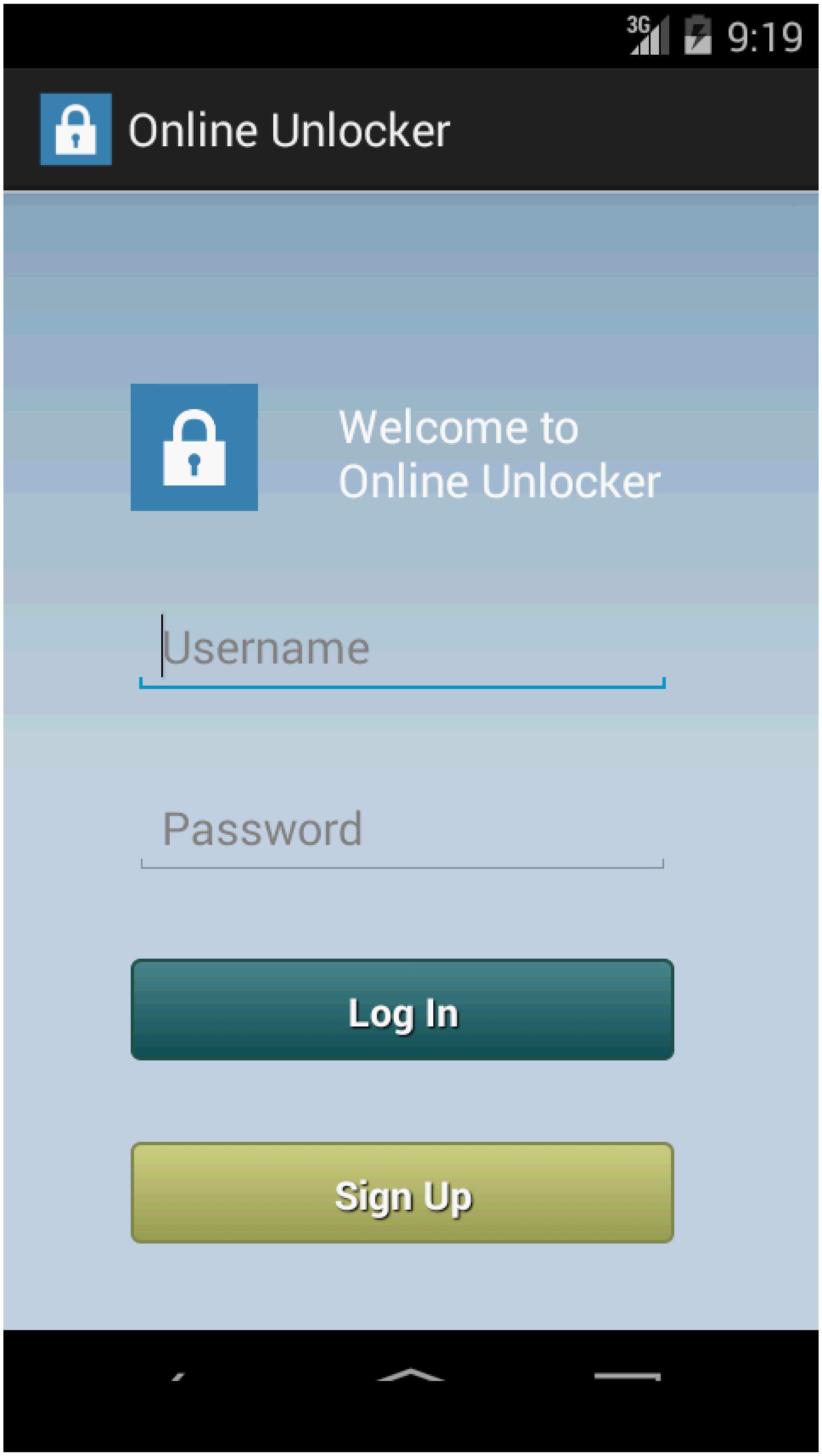
a. Location based unlocking: This feature makes sure that the authorized user (not owner) is at the geographical location near the door so that that person can unlock the door? This way, if a smart phone (belonging to an authorized user) is stolen, it cannot be used to unlock unless it is in the vicinity.

   i. **Details:**

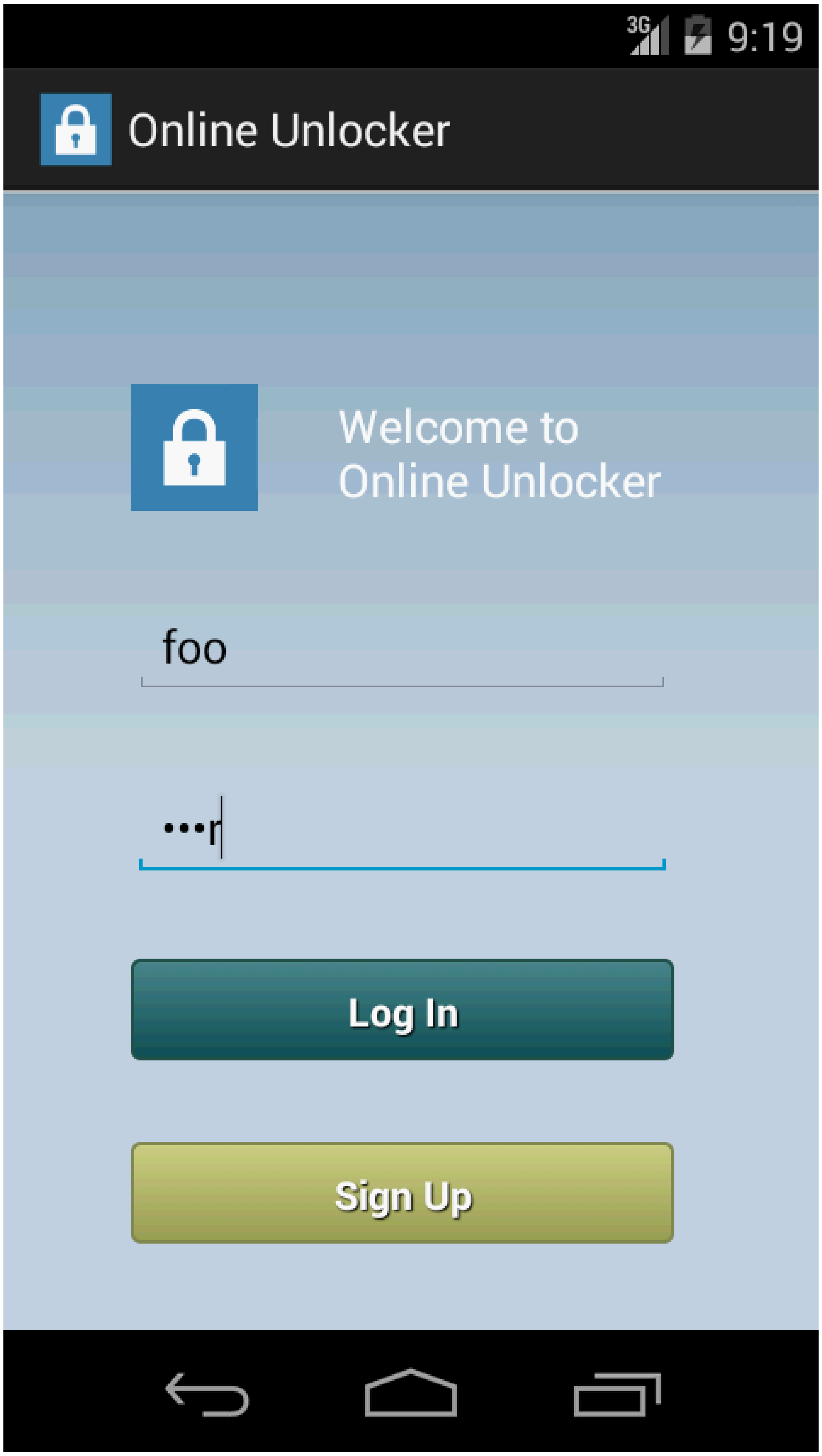
      1. Server : The server gets the GPS location (Co- ordinates ) from the Android Client and checks with the Raspberry PI Client Co-ordinates if the Android Device is in the vicinity of the Raspberry PI then only the Lock is Unlocked else no operation is done.
      2. Android Client: The Android device gets the GPS location of the device and sends it to the Server so that the server can check with the Raspberry PI location and make sure that the device is with in the vicinity.
      3. Raspberry PI: The Co- ordinates for the Raspberry PI or obtained by using the PI as the GPS receiver. So that we can know the exact location of the device (Co- ordinates) thus we can use this co-ordinates to check with the Android Device (Co-ordinates) and unlock the lock.

## Screenshots:

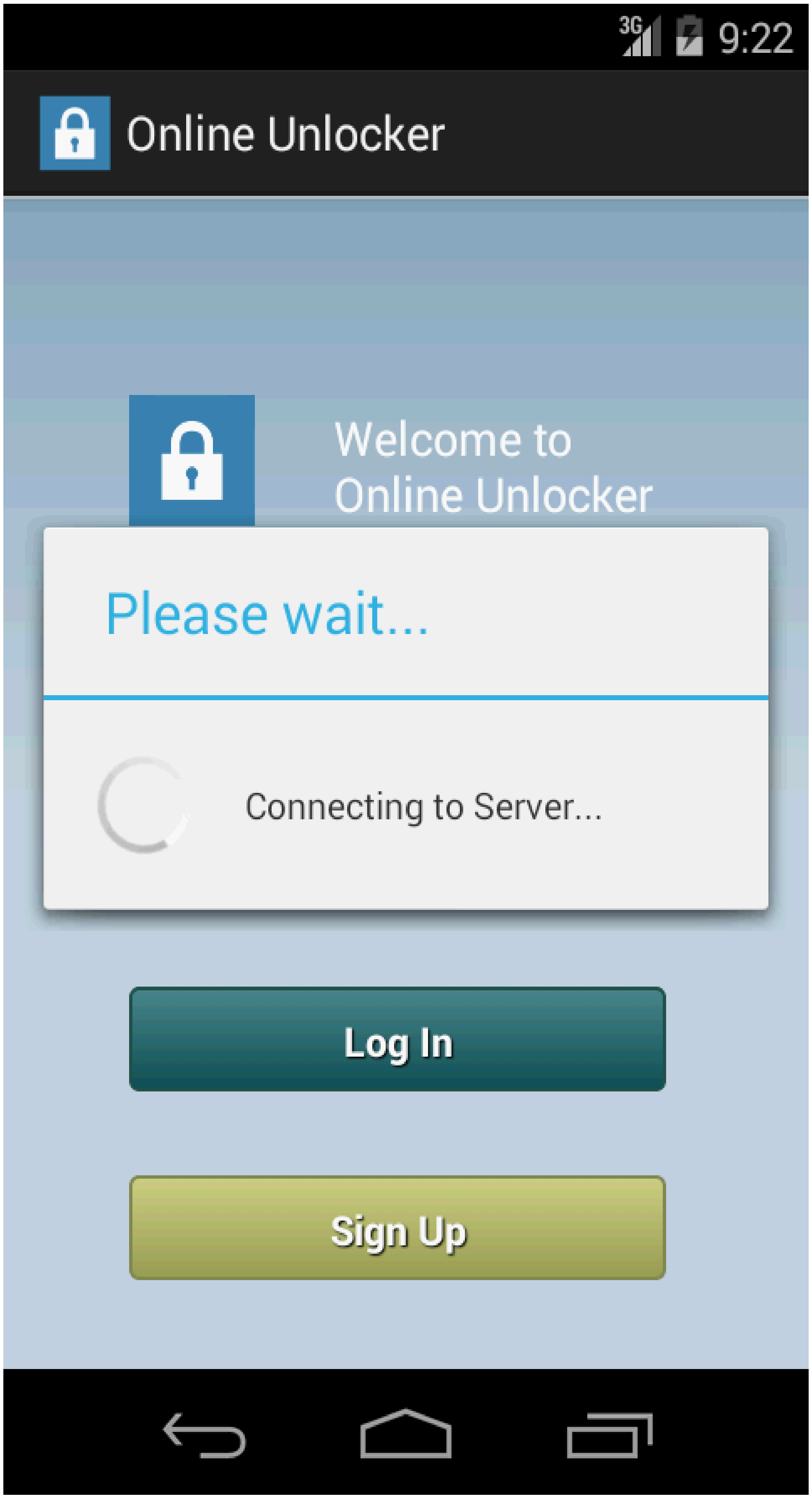
1. Login Screen:



**2.** User authentication is done with a valid username and password combination:



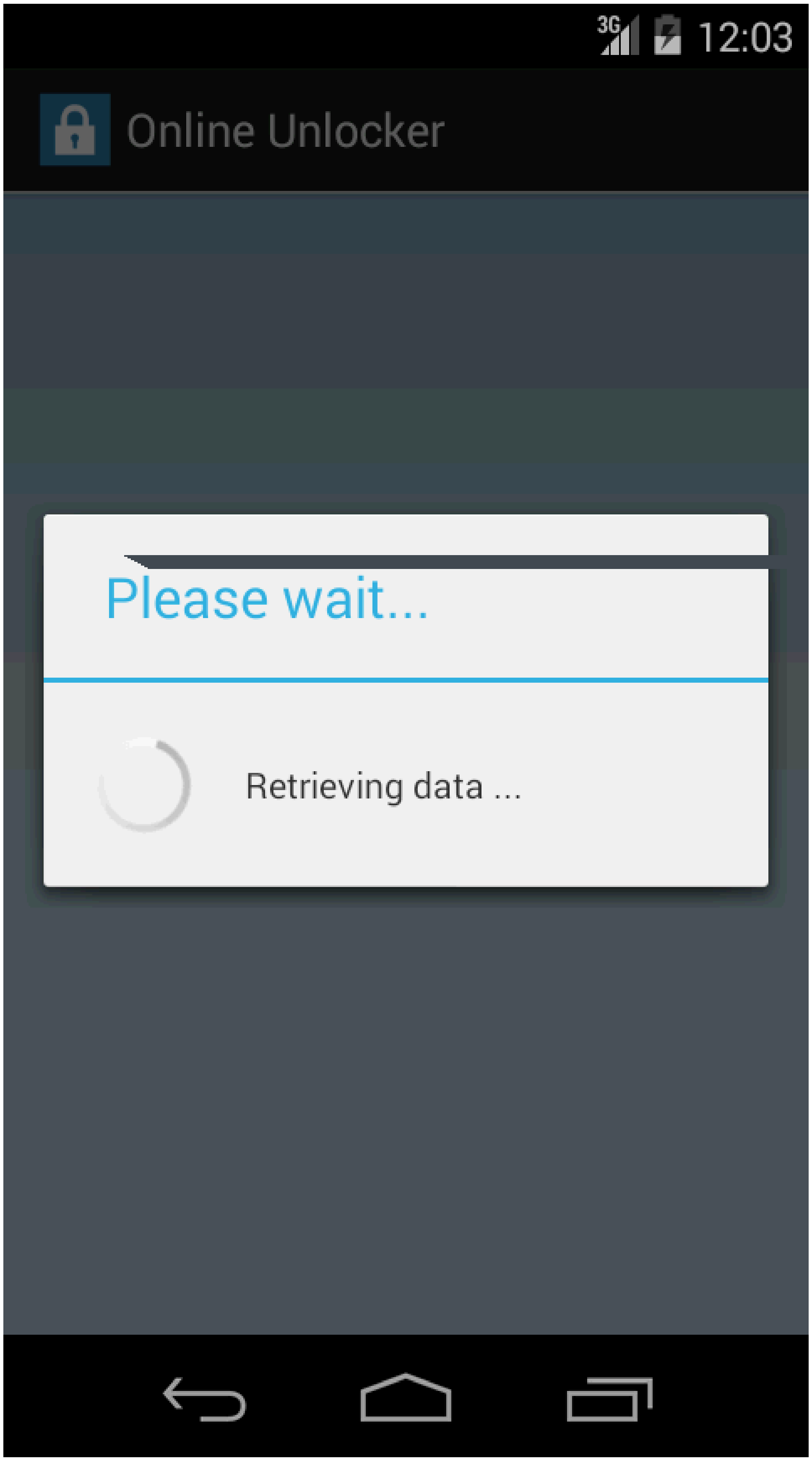
3. Validation process is done by connecting to backend server that stores user credentials:



Please wait...

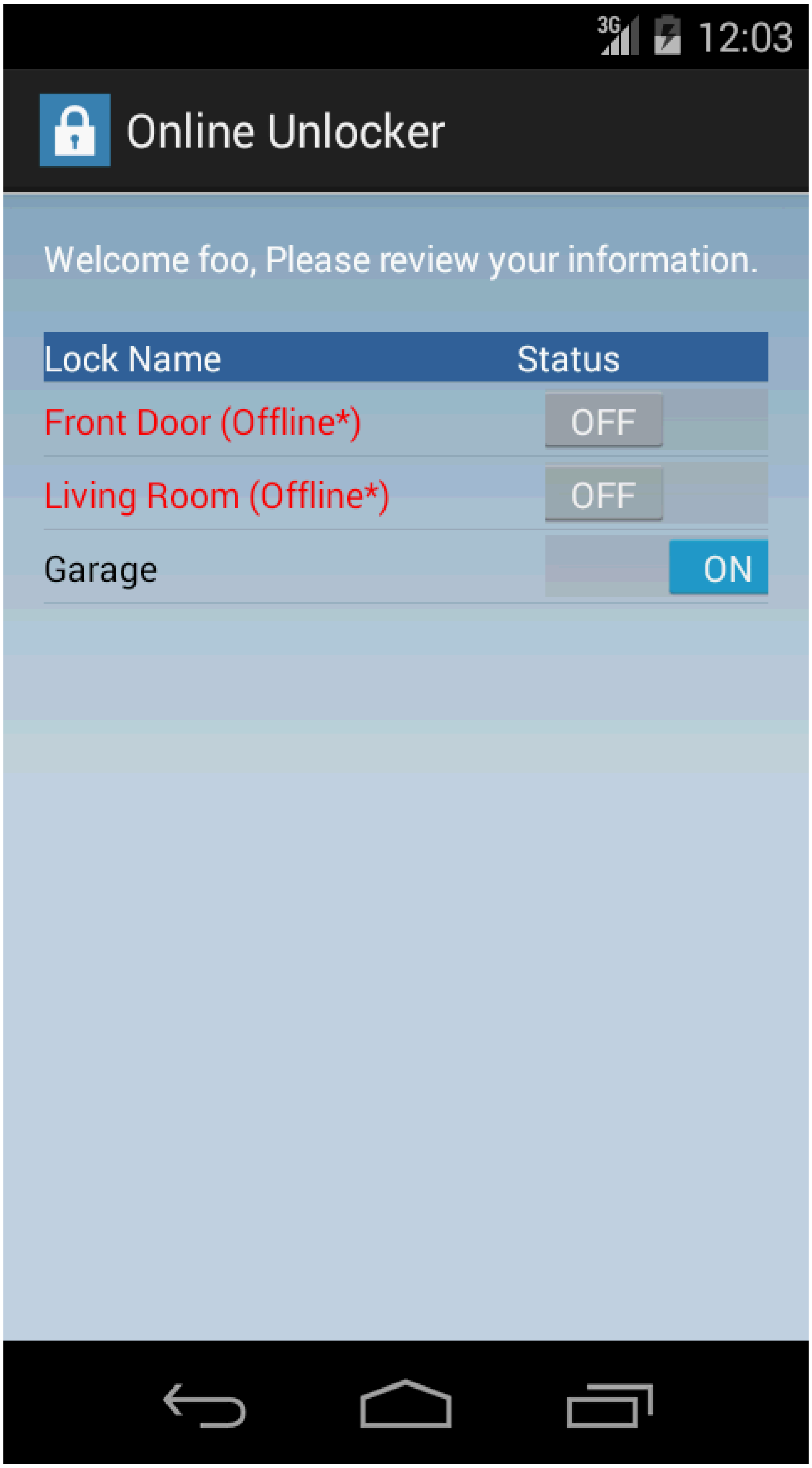
Connecting to Server...

4. After a valid combination of username and password, user is then redirected to a page which displays all of the locks that have been linked to the account:



Retrieving data ...

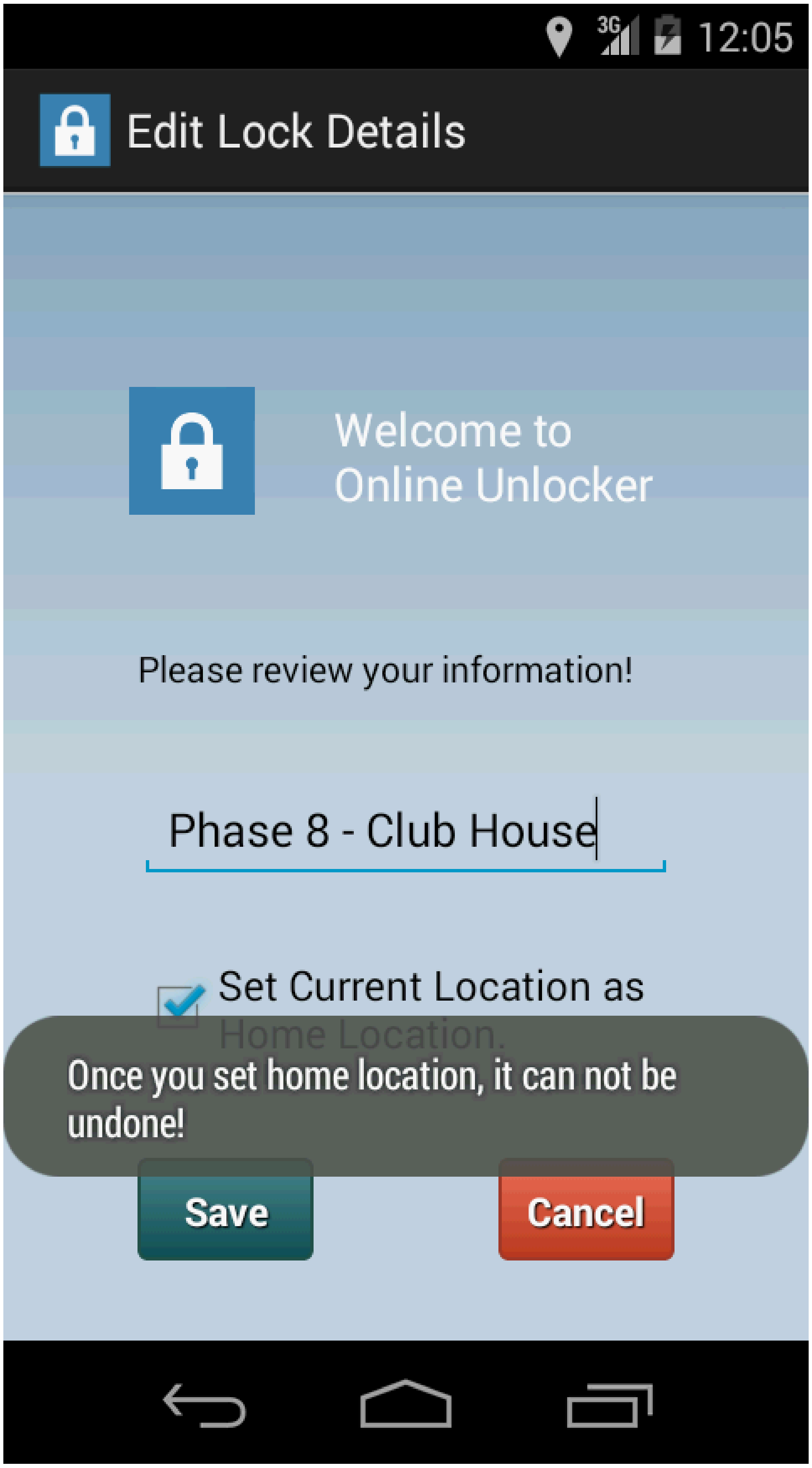
5. Locks can be locked/unlocked using a toggle switch. Application periodically updates the data to stay updated on current status of the lock.



Welcome foo, Please review your information.

| Lock Name | Status |
|---|---|
| Front Door (Offline*) | OFF |
| Living Room (Offline*) | OFF |
| Garage | ON |

6. A quick way to edit a lock details is to touch on its current name. User can set current place as an additional location to be able to operate on a lock from remote place.



Welcome to
Online Unlocker

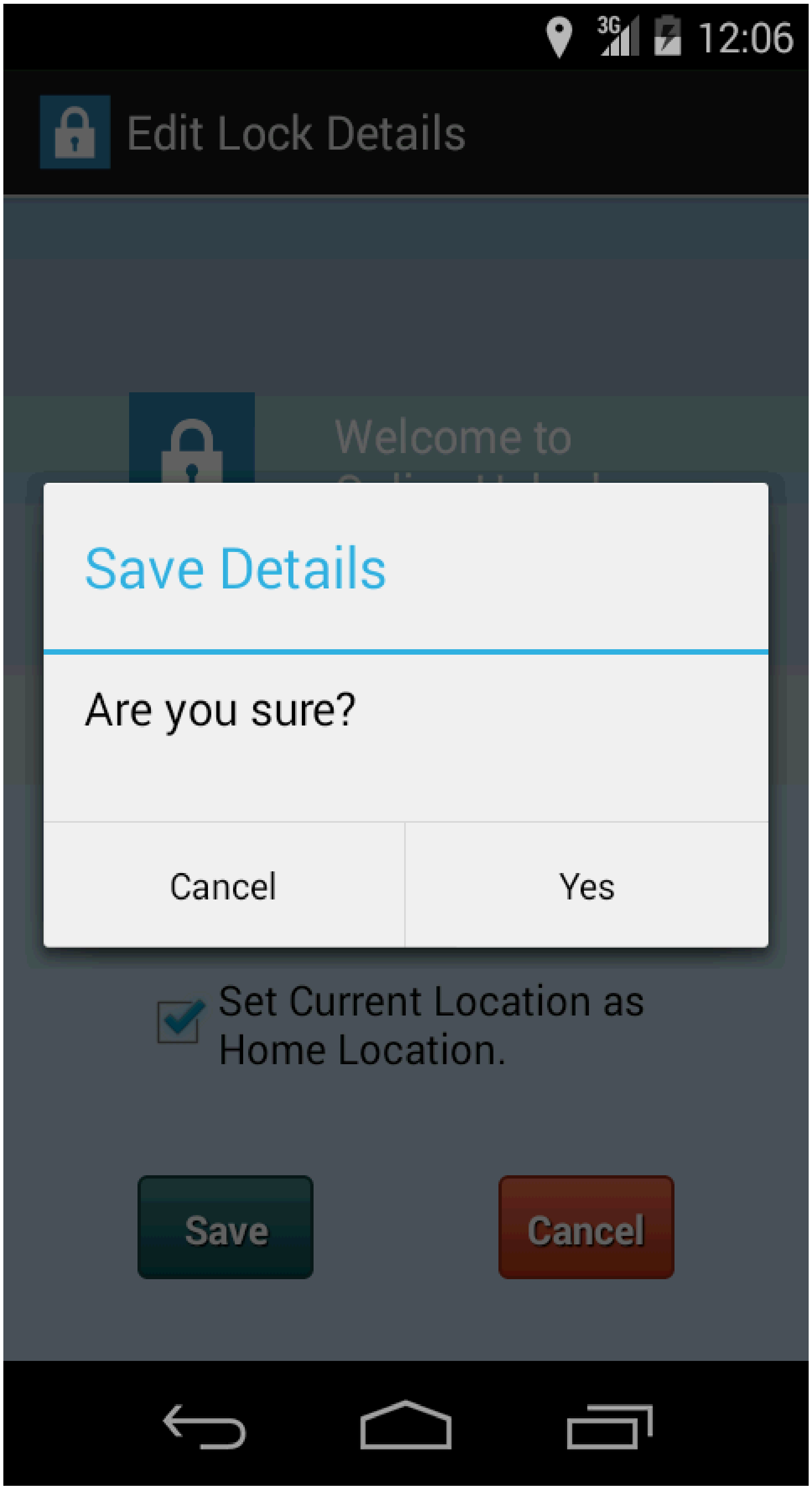
Please review your information!

Phase 8 - Club House

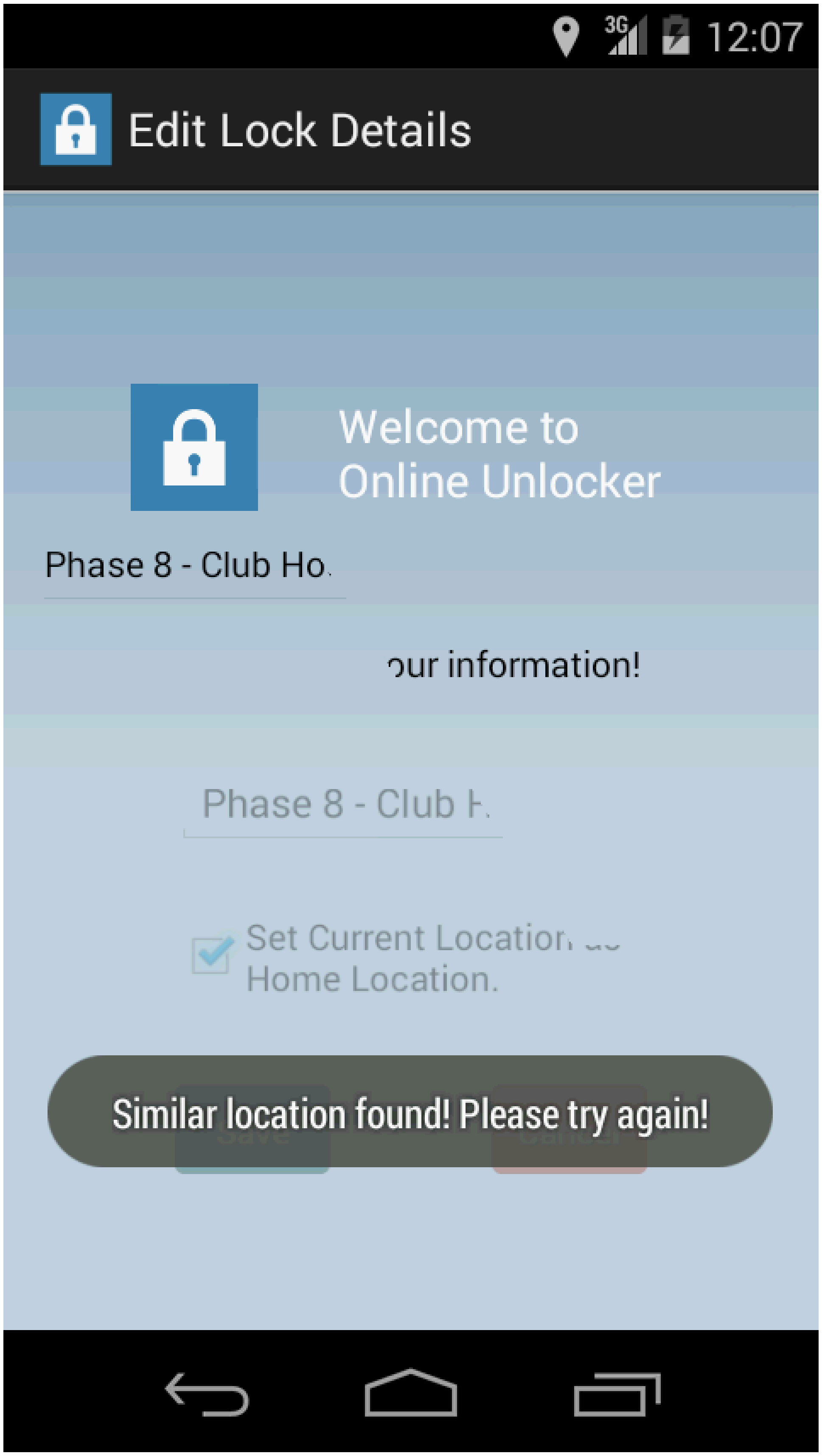
✓ Set Current Location as Home Location.

Once you set home location, it can not be undone!

**7.** A prompt screen to confirm the changes is then displayed.

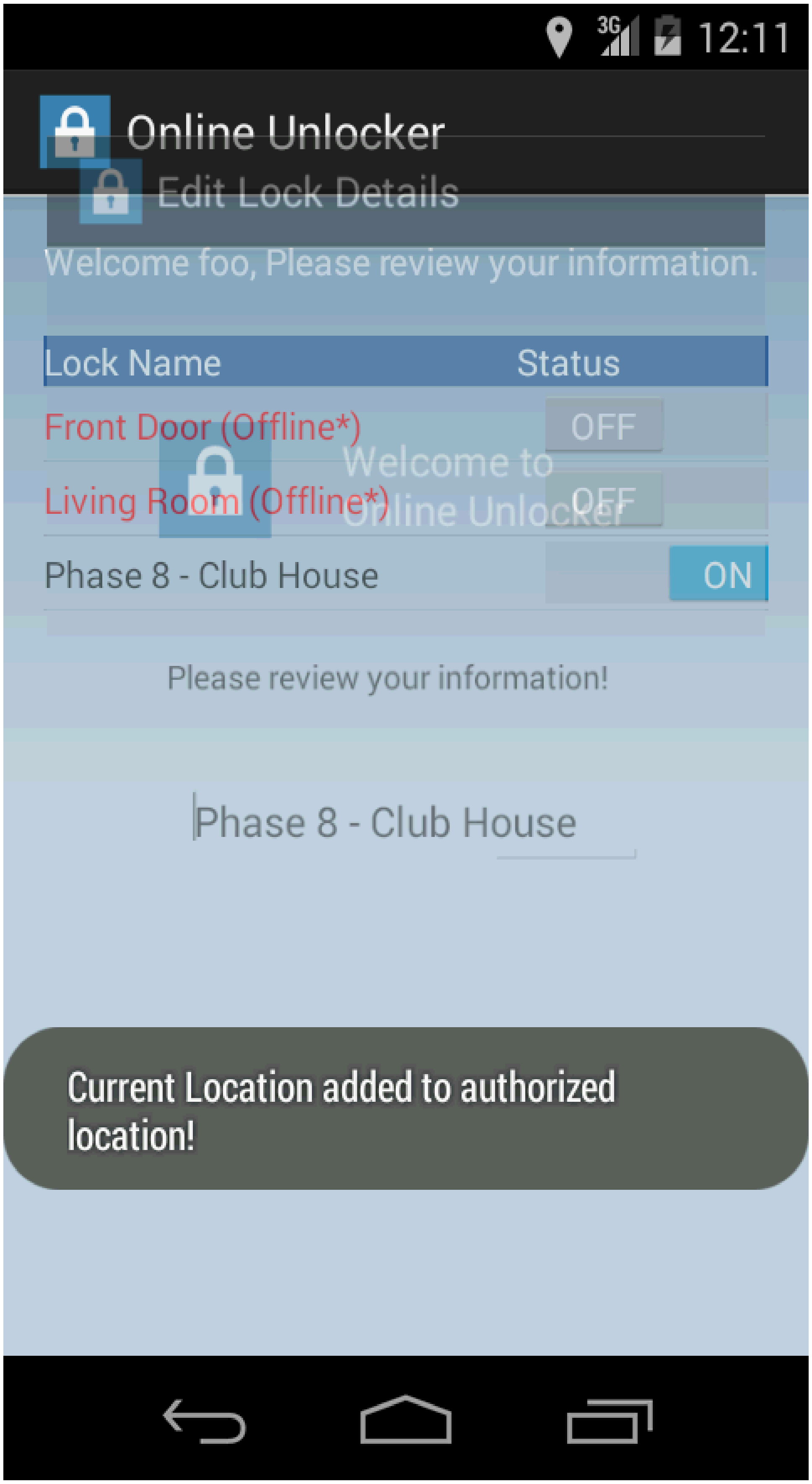


**8.** If the current location is within the proximity of 500ft of physical lock, then error condition will be raised.



Similar location found! Please try again!

**9.** Otherwise, current location can then be used to operate the lock.


12:11

Online Unlocker

Edit Lock Details

Welcome foo, Please review your information.

| Lock Name | Status |
|---|---|
| Front Door (Offline*) | OFF |
| Living Room (Offline*) | OFF |
| Phase 8 - Club House | ON |

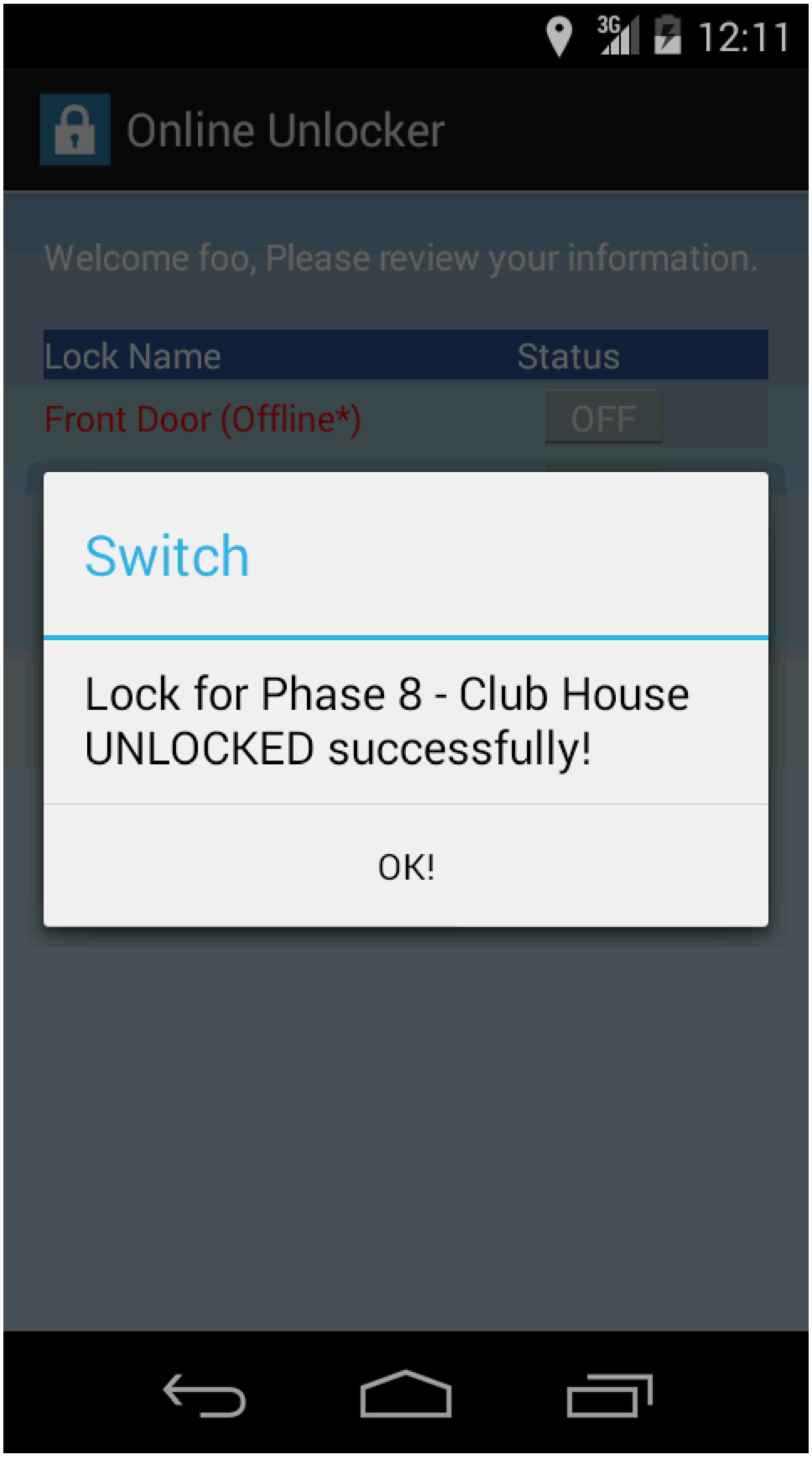
Welcome to Online Unlocker

Please review your information!

Phase 8 - Club House

Current Location added to authorized location!

**10.** If the user is within the vicinity of 500ft of the lock or in the authorized location, lock can be locked/unlocked right from the mobile app.



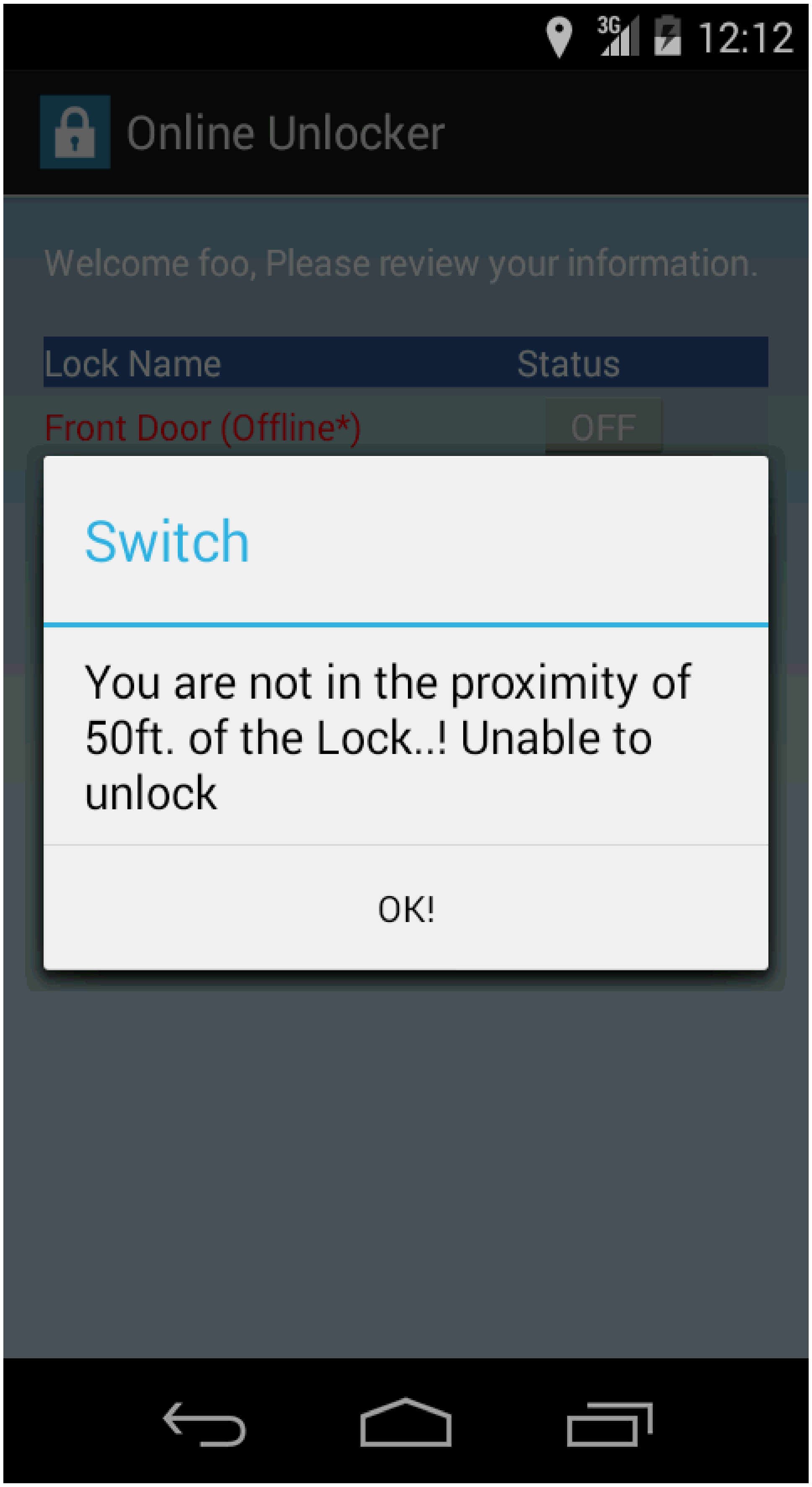
Welcome foo, Please review your information.

| Lock Name | Status |
|---|---|
| Front Door (Offline*) | OFF |

Switch

Lock for Phase 8 - Club House UNLOCKED successfully!

**11.** If the user is trying to operate on the lock from an unknown location.



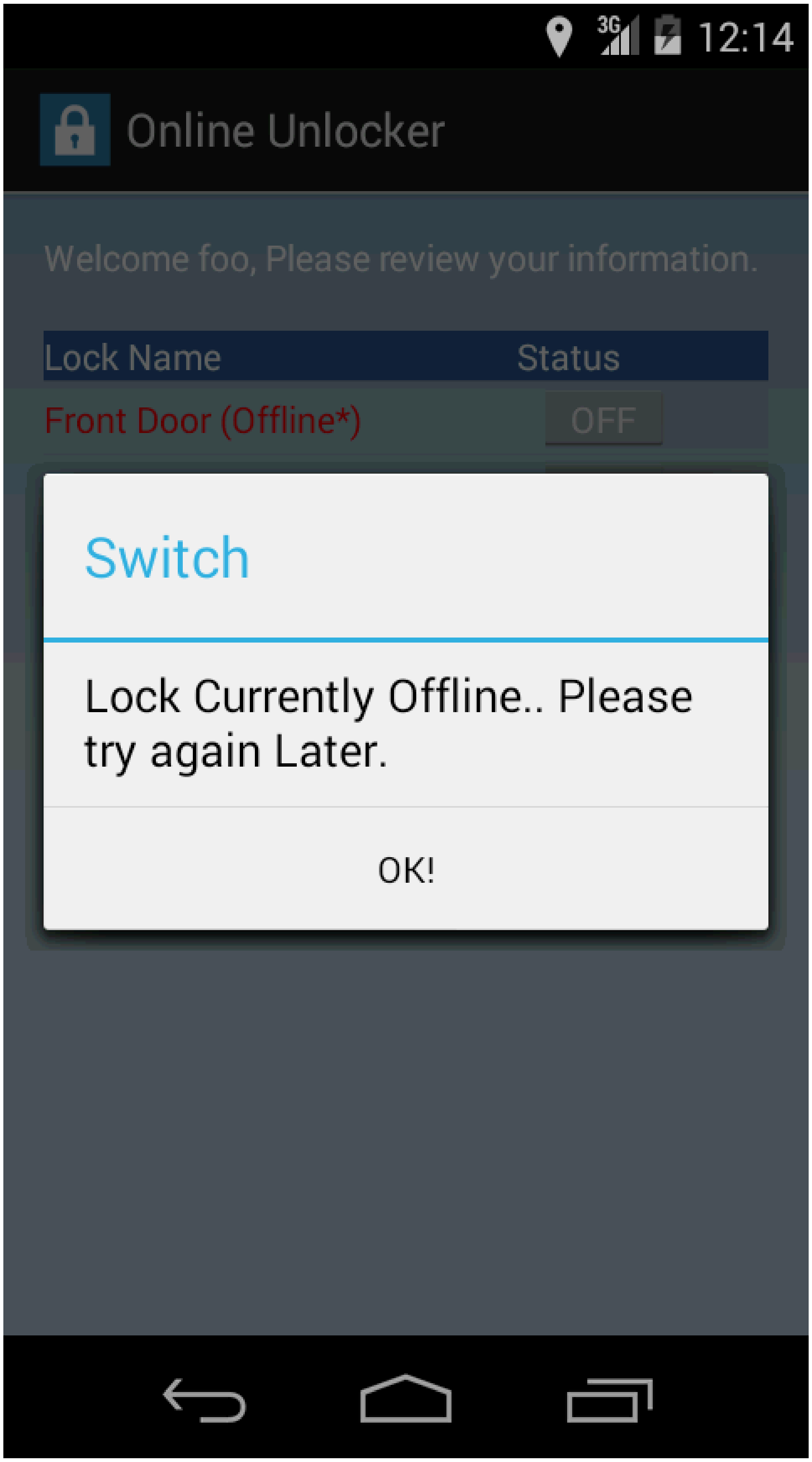
Welcome foo, Please review your information.

| Lock Name | Status |
|---|---|
| Front Door (Offline*) | OFF |

Switch

You are not in the proximity of 50ft. of the Lock..! Unable to unlock

**12.** If the lock is not reachable, then it goes in offline mode if the system cannot receive a heartbeat from it and cannot be operated until it becomes online.



Welcome foo, Please review your information.

| Lock Name | Status |
|---|---|
| Front Door (Offline*) | OFF |

Switch

Lock Currently Offline.. Please try again Later.

## Component Details

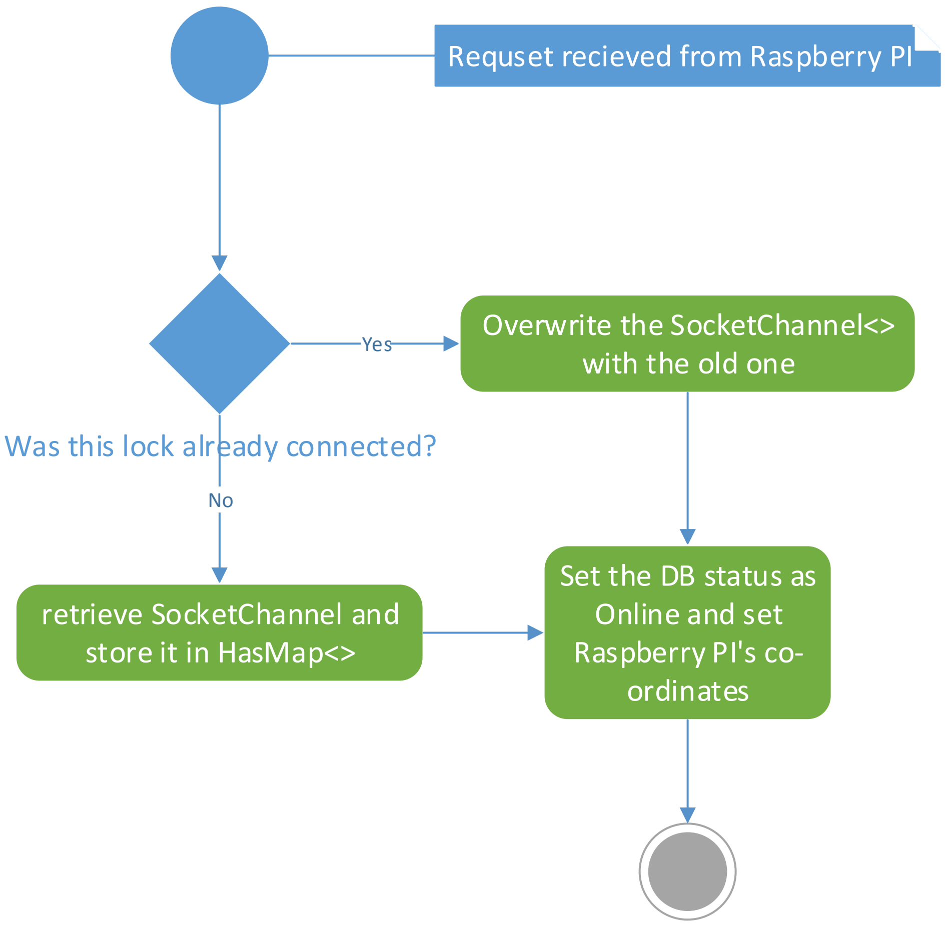
### Server:

The Server will manage requests from both the Android Device and the Raspberry PI to lock/unlock the door-lock. The server will have a back-end database which will store the current state of the lock and will act depending upon the state of the door-lock.

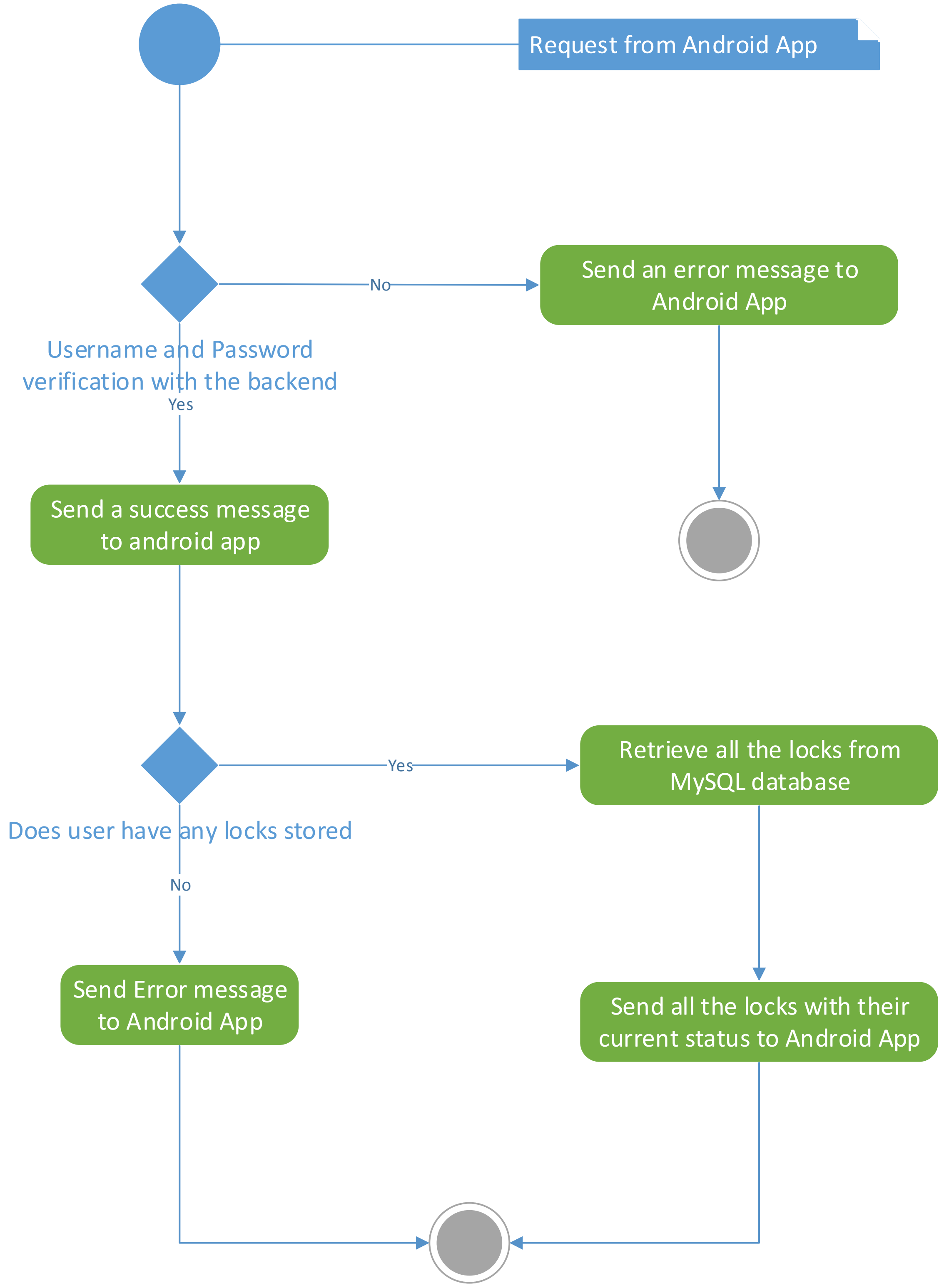
Server will have a multi-threaded environment. Every single request regardless of the source i.e. Android Cellphone or Raspberry PI, a new thread will be created to process the complete request without affecting the listening capability of other requests.

### Server Data Flow Diagrams:

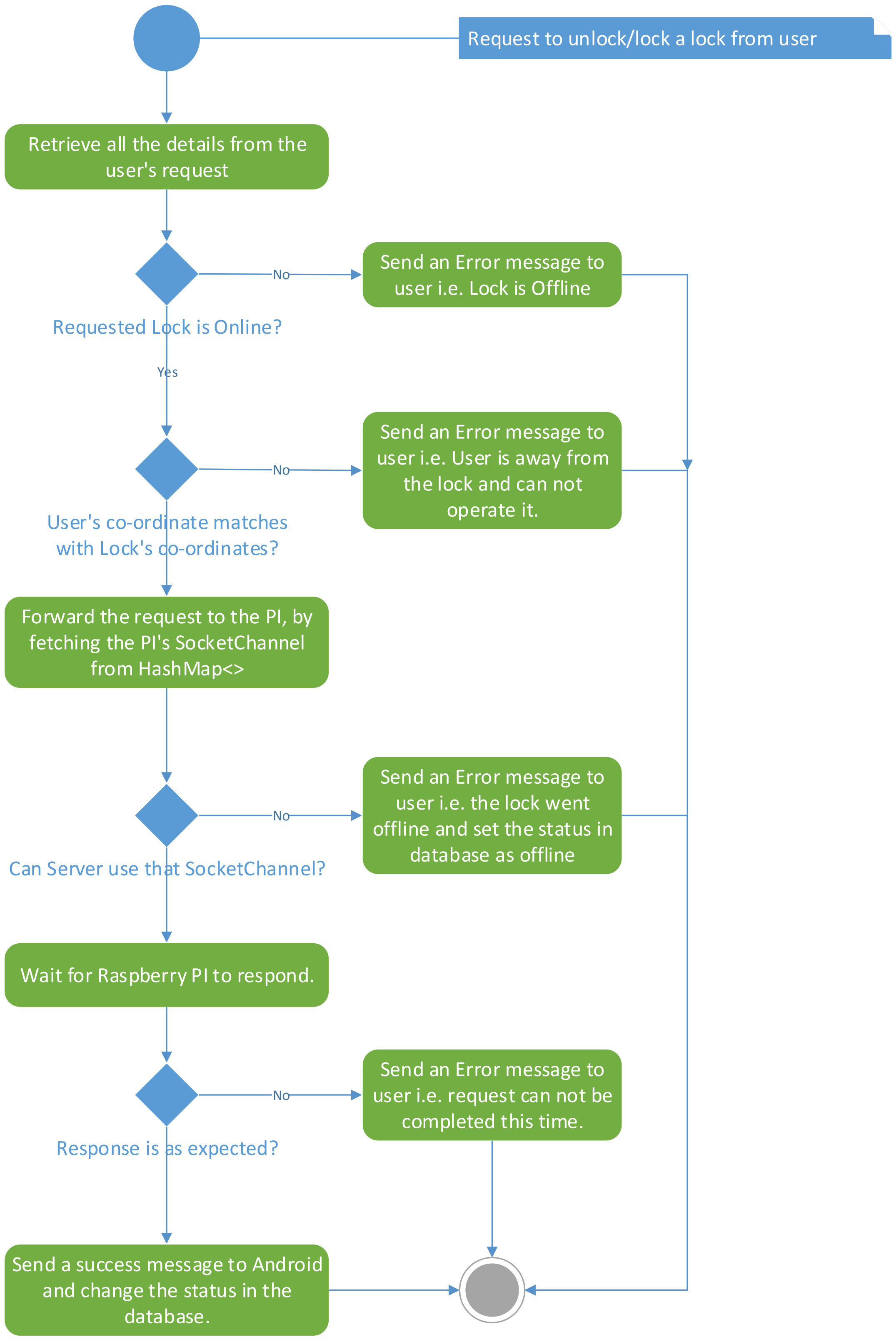
ii. An Initial request from Raspberry PI when it boots up:


Requset recieved from Raspberry PI
Yes
Overwrite the SocketChannel<> with the old one
Was this lock already connected?
No
retrieve SocketChannel and store it in HasMap<>
Set the DB status as Online and set Raspberry PI's co-ordinates

iii. Android app tries to log-in into the system:


Request from Android App
Username and Password verification with the backend
No
Send an error message to Android App
Yes
Send a success message to android app
Does user have any locks stored
Yes
Retrieve all the locks from MySQL database
No
Send Error message to Android App
Send all the locks with their current status to Android App

iv. User tries to operate a lock:


Request to unlock/lock a lock from user
Retrieve all the details from the user's request
Requested Lock is Online?
No
Send an Error message to user i.e. Lock is Offline
Yes
User's co-ordinate matches with Lock's co-ordinates?
No
Send an Error message to user i.e. User is away from the lock and can not operate it.
Forward the request to the PI, by fetching the PI's SocketChannel from HashMap<>
Can Server use that SocketChannel?
No
Send an Error message to user i.e. the lock went offline and set the status in database as offline
Wait for Raspberry PI to respond.
Response is as expected?
No
Send an Error message to user i.e. request can not be completed this time.
Send a success message to Android and change the status in the database.

## Server Classes:

v. **Data-structures:**

1. **Message.java:** This class provides a way to communicate between Android App, Web Server and Raspberry PI. This class contains fields which are then used to extract the information that has been passed from one component to another.
2. **DatabaseConnection.java**: This class is mainly used to communicate with the database. This class also provides a separation of application logic required for database. This way web server can easily pass the information to this class to update the database or to retrieve the information from the database.

vi. **Threads**:

1. **MainServerThread.java**: This class provides a listener's capability for web server. All the incoming requests will be received by this class and then depending upon the type of the request.
2. **MobileRequestThread.java**: Upon receiving a request from Android App, this thread will be invoked and will process the entire request on its own. For every new request from Android App a new thread of this class will be created, thus every thread will work independently.
3. **RaspberryPIRequestThread.java**: When Raspberry PI comes online, it will register itself with the web server. Such registration request will be handled by this thread. It will also store the SocketChannel on which the request has arrived for future PUSH functionality.

vii. **Main**:

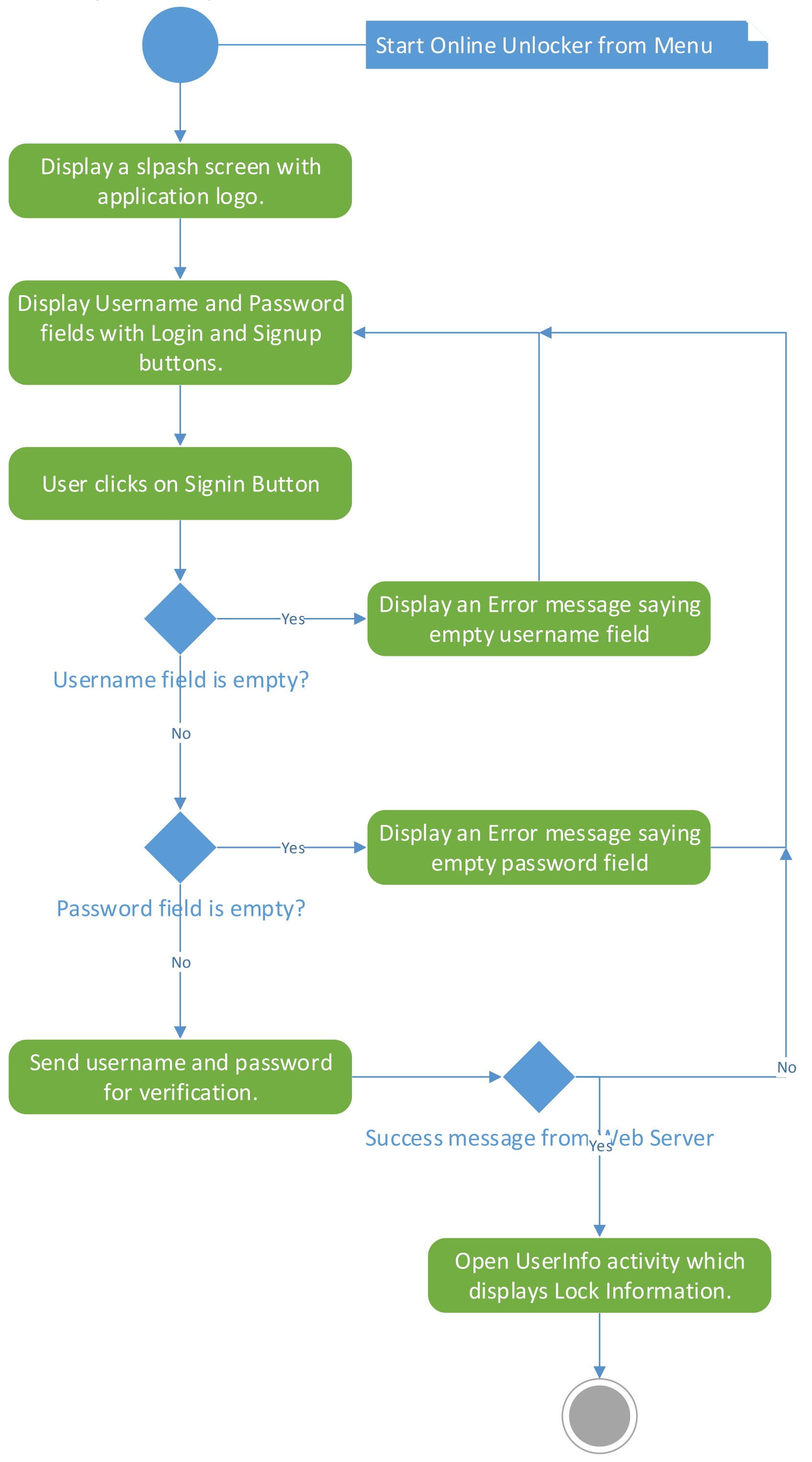
1. **SocketServer.java**: This class will start a listener thread i.e. MainServerThread with the configuration specified for the web server.

## Android Client:

The code for the Android client will be Java based and written using the Eclipse IDE. The minimum API to execute the code will be 14 (Android 4.0 “Sandwich”) which will be required for the switches to function properly. The target API will be 18 (Android 4.3 “Jelly Bean”).

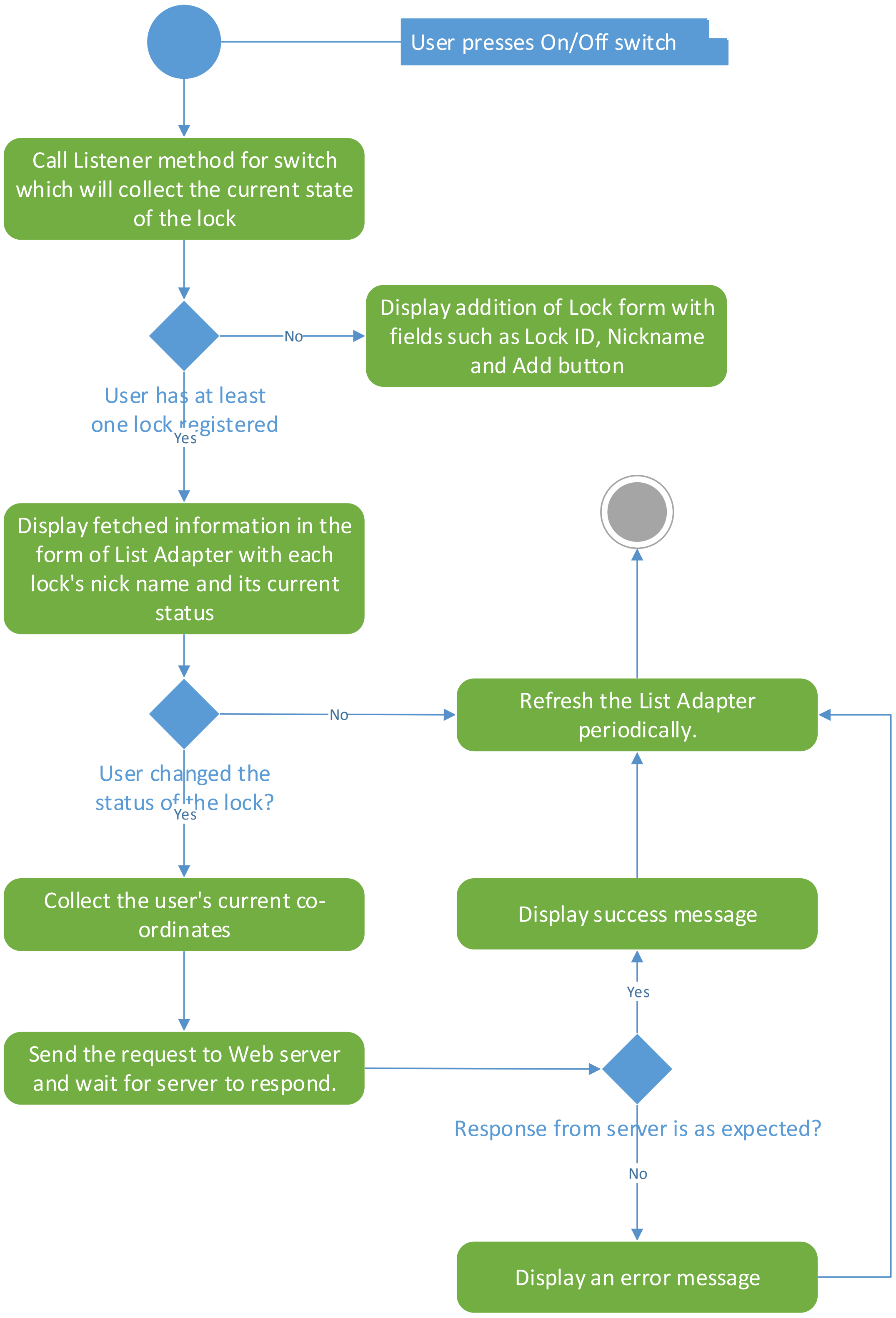
## Android Client’s Data Flow Diagrams:

viii. User Login Activity:

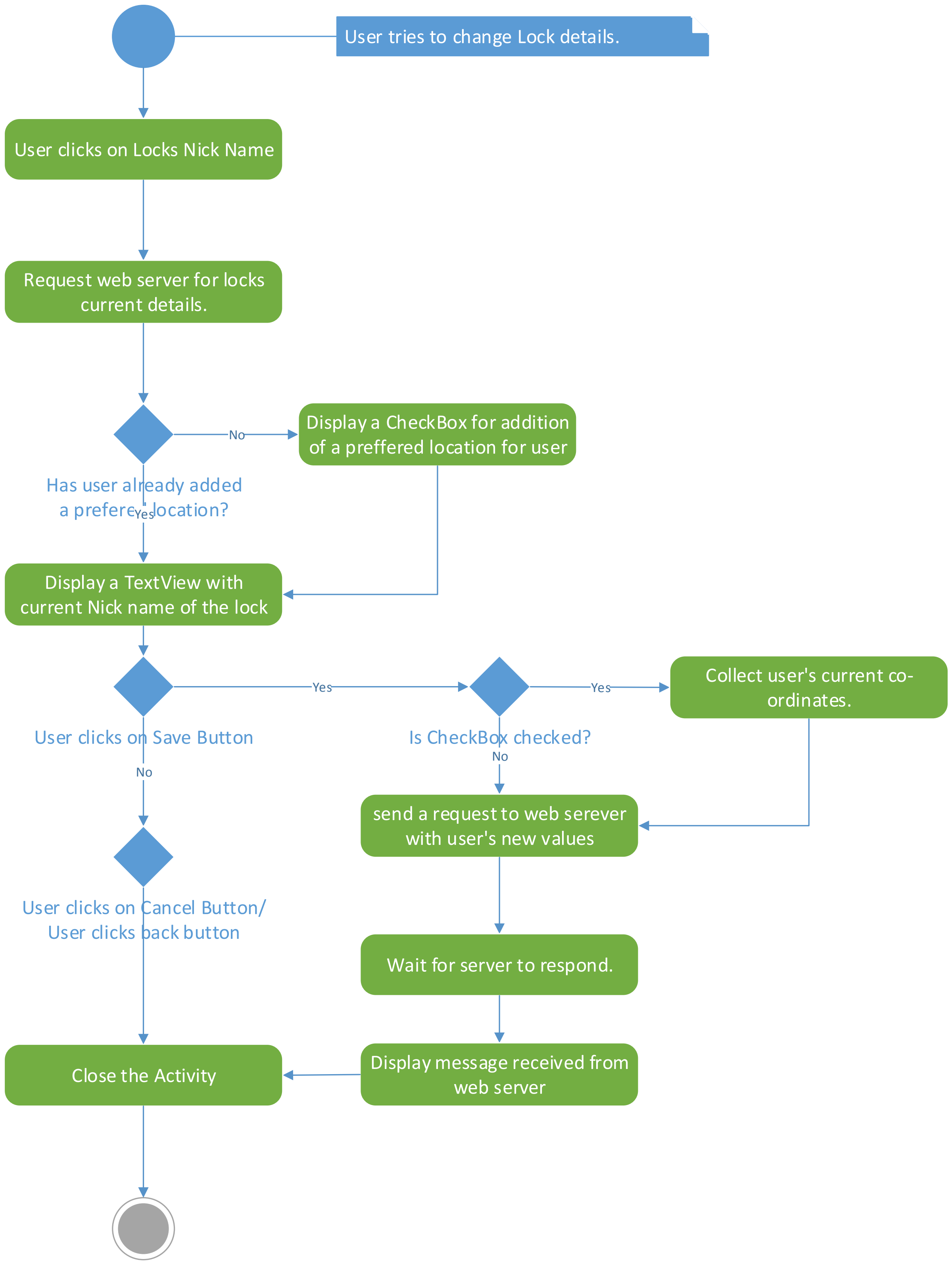


Start Online Unlocker from Menu
Display a slpash screen with application logo.
Display Username and Password fields with Login and Signup buttons.
User clicks on Signin Button
Yes
Display an Error message saying empty username field
Username field is empty?
No
Yes
Display an Error message saying empty password field
Password field is empty?
No
Send username and password for verification.
No
Success message from Web Server
Yes
Open UserInfo activity which displays Lock Information.

ix. User tries to unlock/lock a Lock:


User presses On/Off switch
Call Listener method for switch which will collect the current state of the lock
No
Display addition of Lock form with fields such as Lock ID, Nickname and Add button
User has at least one lock registered
Yes
Display fetched information in the form of List Adapter with each lock's nick name and its current status
No
Refresh the List Adapter periodically.
User changed the status of the lock?
Yes
Collect the user's current co-ordinates
Display success message
Yes
Send the request to Web server and wait for server to respond.
Response from server is as expected?
No
Display an error message

x. User tries to Edit Lock Details:


User tries to change Lock details.
User clicks on Locks Nick Name
Request web server for locks current details.
Has user already added a prefered location?
No
Display a CheckBox for addition of a preffered location for user
Yes
Display a TextView with current Nick name of the lock
User clicks on Save Button
Yes
Is CheckBox checked?
Yes
Collect user's current co-ordinates.
No
send a request to web serever with user's new values
No
User clicks on Cancel Button/ User clicks back button
Wait for server to respond.
Display message received from web server
Close the Activity

## Android Classes:

xi. Data structures:

1. **JSONParsor.java**: This class is used to make the HTTP calls and get the response from the web server. All the responses for the HTTP requests are encoded as a JSON object, hence this class will be used to parse the response which then can be given to Android app for further processing.
2. Message.java: This class provides a way to communicate between Android App, Web Server and Raspberry PI. This class contains fields which are then used to extract the information that has been passed from one component to another.

xii. Main:

1. MainActivity.java: This class will display screen with username and password fields along with Login and sign-up button. Upon providing username and password by the user, this class will verify user's credentials with the web server and will display an error message if there is a mismatch the given credentials. Otherwise it will open UserInfo.class.
2. UserInfo.java: This class will retrieve user's lock information if user has already registered few locks. Otherwise it will display a form asking user to register a new lock.
3. LockInfo.java: An instance of this class will be created for every lock that user has registered. This class will contain information such as unique LOCKID and its current status.
4. LockAdapter.java: This class will create instances of LockInfo and will add it to display screen.
5. LocationInfo.java: The LocationInfo class serves as an abstraction to access the Location class of the Android API. It is included to allow for the user ability to disable locking and unlocking from outside of a predetermined distance from the Raspberry PI.
6. EditLockDetails.java: This class will provide customization of Lock information. If user has not added a preferred location for the lock, this class will display a CheckBox to give user an option to add his current location so that the user can operate the lock from this location along with the 50ft radius of the lock's physical location.
7. AlertDialogSingleButton.java: This class is derived from AlertDialogBox, which is used to display an alert box with a single button along with a custom message.

## Android User Interface:

The user interface has been designed to be both intuitive and seamless to the end user, abstracting the "back end" details outlined in this report. Screen shots for the user interaction with the Android component can be found in appendix 1 of this document.

Android User Interface has following Layouts defined in it;

1. **splash_screen.xml**: This layout will be displayed when user opens up the App from the menu.
2. **welcome_screen.xml**: This layout will have two TextViews i.e. Username and Password fields. Along with which it will have two Buttons i.e. Login and Sign up buttons.
3. **user_info.xml:** This layout will be populated once the information from the web server has been successfully retrieved. This layout will have a ListView which will contain list_view_lock_info.xml and a header as list_view_header.xml.
4. **edit_lock_details.xml**: This layout will provide an option for user to customize its Lock information stored in the backend.

## Raspberry Pi Client:

The Raspberry PI client communicates with the server via the Internet.  It is the responsibility of the Raspberry PI to handle messages sent from the server and respond by performing the necessary changes to the lock (engage / disengage). Upon receiving the respective signal from the server the Raspberry PI triggers a relay which can be attached to an electromechanical device of the users choosing.

If the Lock is manually unlocked the time stamp of the change along with the present status of the lock is sent back to the server which can than update the Raspberry PI of the current status.

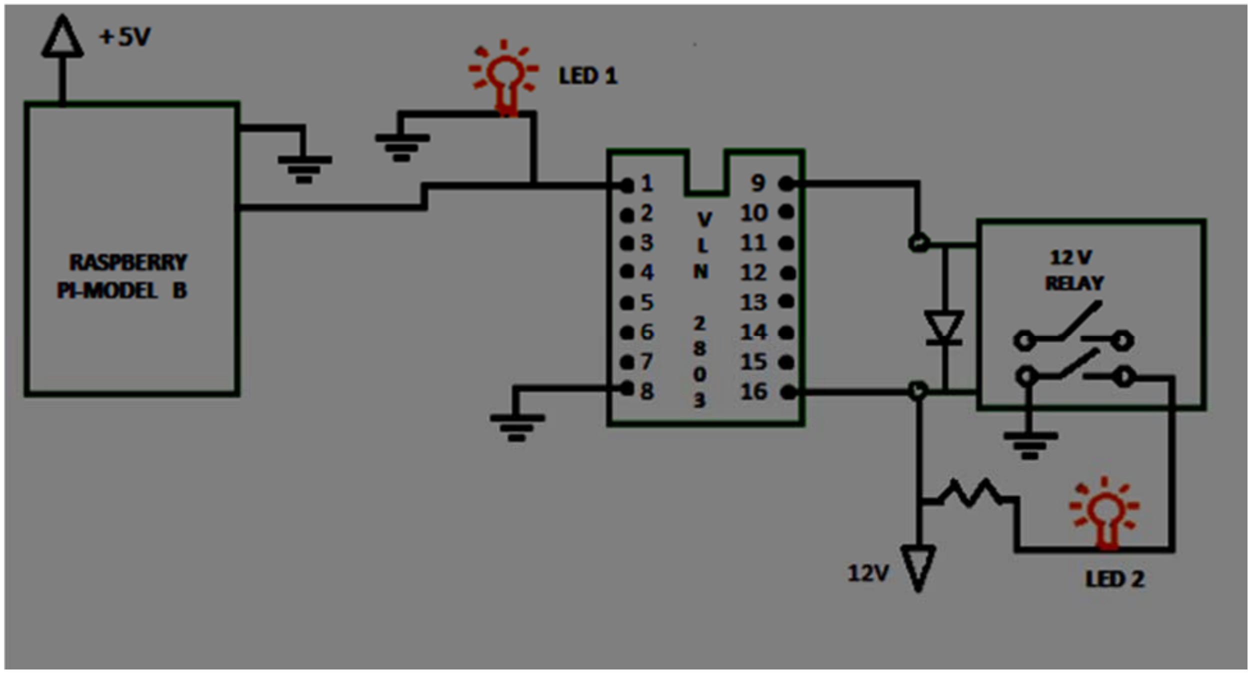
### Raspberry PI Classes:

1. **RaspberryPIServerThread.java**: This class will start the only thread that will be running on Raspberry PI, and with specified configuration it will initially try to register itself with the Web Server. Upon successful registration, it will then wait for server to PUSH the data to it on which it will lock/unlock the lock that has been connected to it. If due to failure it fails to connect with the Web Server, it will try to do so after every 5 seconds.

### Mechanical:

We considered a number of possible interfaces for controlling the GPIO pins on the Raspberry Pi. The pins can be accessed via the file system by adding the appropriate pin numbers to the 'export' file in the /sys/class/gpio directory and then manipulating files in the individual pin directory, such as 'value' and 'direction'. A pin with 'direction' of 'out' and a 'value' of 'HIGH' or '1' will be outputting a 3.3V signal until 'value' is set back to '0'.

There are a number of modules in various programming languages that have been developed for carrying out this functionality. The Python code we started with used the RPi.GPIO package for GPIO controls; the Java code we are currently using has utilized the Pi4J library, which is a JNI wrapper for the WiringPi GPIO access library. In our code, therefore, the 'export' of a pin with direction 'out' is accomplished by the provisionDigitalOutputPin function, and the changes in signal are carried out by setting PinState to 'HIGH' or 'LOW'.



## Mechanical Parts-List:

2. Raspberry PI Model B
3. Led's
4. ULN 2803
5. Relay (12v)
6. Execution