SMART LOCK SCREEN FOR ANDROID

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Aravind Sagar, 1100104,

Discipline of Computer Science and Engineering,

Indian Institute of Technology, Indore

Guided by:

Dr. Gourinath Banda,

Assistant Professor,

Computer Science and Engineering,

IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2014

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Smart Lock Screen for Android"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Dr. Gourinath Banda, Assistant Professor, Computer Science and Engineering**, IIT Indore is an authentic work. Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Aravind Sagar

CERTIFICATE by BTP Guide

It is certified that the above statement made by the students is correct to the best of my knowledge.

Dr. Gourinath Banda,
Assistant Professor,
Discipline of Computer Science and Engineering,
IIT Indore

Preface

This report on "Smart Lock Screen for Android" is prepared under the guidance of Dr. Gourinath Banda, Assistant Professor, Computer Science and Engineering, IIT Indore.

Through this report, we have tried to provide a detailed description of our approach, design and implementation of an innovative method to balance security and convenience of modern day smart phones. We also try to cover additional aspects of the developed app, which provide even more convenience and security, in certain situations, for smart phones and the personal data contained within.

We have tried our best to explain the proposed solution using lucid examples, and added screen-shots of our implementation for added clarity. Analysis of the ability of the implemented app to add convenience and save time, if used in certain configurations, is also included.

Acknowledgements

I wish to thank Dr. Gourinath Banda for his kind support and valuable guidance throughout the duration of the project, giving me an opportunity to work at my own pace along my own lines, while providing me with very useful directions whenever necessary. I would also like to thank all the faculty members of Discipline of Computer Science and Engineering for their invaluable support and constructive feedback during the presentations. I would also like to thank my project partner (V. Priyan, CSE1100136) for his constructive support throughout.

I wish to express my sincere gratitude for XDA Forums for providing a great platform to gain testing audience for the implemented app, and all the members there who downloaded and provided very encouraging feedback about the app. Special thanks to Udacity.com for providing an excellent Android App Development course for free, and to Github.com, for providing their micro plan for students for free.

Finally, I offer my sincere thanks to everyone else who knowingly or unknowingly helped me complete this project.

Contribution

This project was done over the course of six months starting from mid-May 2014. The first month was mainly used for survey of relevant and related solutions which exist for the problem at hand, namely creating a balance of security and convenience for smartphone users. I, together with my project partner(V. Priyan, CSE1100136), analysed these solutions and realized the advantages and drawbacks of them. Then we started studying the Android framework(target platform), and Android-app-development, along with implementing some sample applications (apps) which helped us in getting familiar with the APIs required to realize the app.

By mid-July, we started implementing the app which helps in securing the phone and relaxing security in certain environments. I started with the database back-end and the core service which runs in background to detect sensor value changes. Then I started with the core-app-logic which detects the current environment based on the sensor values. After making sure that the detection runs robustly in all scenarios, I started with implementing administrative actions in the app, that is changing the password in the background according to current environment. Around this time, we got the idea of showing notifications in lock-screen. Later I changed some initial implementation to work around some inherent platform limitations of displaying an overlay over the standard lock-screen. Meanwhile I also got an idea to overcome a challenge we were facing in making pattern unlock available for users. A pattern view library was developed, and used in the app to provide the functionality. I constantly modified the UI of the app in accordance with 'Material Design' guidelines and also improved the lock-screen overlay, providing additional functionality like short-cuts, and enhancing the animations. I then implemented back-end and User-interface (UI) for different User-profiles, which helps in securely sharing phones with others. In the end, we analysed the possibility of reducing phone unlock time when our app is used in practice. We also published (available for download by Android-based smart-phone users) the app as alpha version, in XDA forums, for gaining a wider test audience and feedback about usability and effectiveness of the app. I made several modifications to the app according to the feedback received, and 3 more updates were published since, with each version having over 1000 downloads in the span of a week. The task allocation between the project members was done by our project guide Dr. Gourinath Banda.

Abstract

Smart phones are becoming very popular, and even replacing traditional computers for a variety of day-to-day tasks. A lot of personal data is contained in smart phones, but the portability of these devices makes them vulnerable to privacy issues in a variety of situations. A strong unlock password can help in these situations, but more often than not, it becomes a hindrance in accessing the phone. In this project, we propose and analyse a potential solution for creating a balance between convenience and security, which is, varying the security level of the smart phone according to the environment it is in. We also developed an app for Android based smart phones which gives users the power to define environments and set a different password for each environment. Additional features to enhance convenience and security were also included in the app. Finally, the possibility to reduce unlock times on an average were analysed using unlock time data from different phones, for different password types. We can see that the implementation can certainly reduce the effort in accessing the phone most of the times, while maintaining strong security levels when it is most required.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Mobile industry is one of the fastest growing industries in recent times, and smartphones are more common than ever before. We carry it everywhere we go, and use it for common tasks without bothering to open a traditional computer, and hence today's smartphones carry a lot of personal data, which makes it a very convenient device.

But the worrying fact is that mobile phone theft rate is increasing alarmingly in many countries. We have to ensure that our data doesn't fall into the wrong hands when we are out and about. Keeping our data private to ourselves is required even in other conditions, and while software and hardware manufacturers are doing their roles by making it almost impossible to use a stolen device, all that is useless unless we set a strong password. Strong passwords are also required to prevent unauthorized access to data in other environments as well.

Having a strong password has its benefits, but it comes at the cost of convenience. On an average a typical smartphone user unlocks the device 110 times a day [1]. It can quickly become tedious to type out a strong password this often. There is a clear trade-off between security and the effort taken to unlock your device.

The motivation for our project is to develop an app which can arrive at a good balance of convenience and security. Modern smart phones have a variety of sensors and radios in them, and these can be used to determine the current environment of the phone, according to which security of the phone can be adjusted.

1.2 Problem Definition

The core problem that we have at hand is providing convenience to smartphone users to unlock and use the phone but maintaining a high security level in the phone at the same time. To achieve this, the problem has been subdivided into 3 parts:

1. To make an intelligent application for Android devices (primarily phones), which will detect the current environment and secure the phone accordingly by varying security levels.

2. To provide glance-able information about notifications to the user, right from the lock-screen.

3. To enable secure sharing of the device with others, without compromising user privacy.

For a detailed list of use cases see section 3.1.

1.3 Existing Solutions

1.3.1 Related Solutions

Several solutions exist, which attempt to provide convenient security measures for portable devices. Here we explore few such solutions.

1. **Pattern Unlock:** A grid (typically 3x3) of dots is shown to the user. A specific connection of 4 or more dots will be stored as reference password, and user has to draw the same "pattern" by connecting the same dots in the same order to unlock the phone[2].

    - Advantages: Simple, easy to understand, faster than password/pin
    - Disadvantages: Lesser security than a password or pin. It's possible to understand the pattern by simply observing the person drawing it, or carefully making out the residues left on the screen by user's finger.

2. **Face Unlock:** Face recognition is employed here. Users face is studied using front camera of the phone, and key information is stored. To unlock the phone, simply make sure that camera can capture your full face in the middle of its view[3].

    - Advantages: Easy to use, can be faster than passwords or pins.
    - Disadvantages: Similar faces can confuse the algorithms used, powerful hardware required to quickly find match between current face and stored one, does not work well in low light situations.

3. **Fingerprint unlock:** Some phones come equipped with Fingerprint sensors which can be used for securing the device. Fingerprint data is acquired and stored using fingerprint sensors, and users can swipe their finger across the sensor to unlock the device[4].

    - Advantages: Secure and easy to use, fast.

- Disadvantages: Specialized hardware is required, typically found only in a few high-end phones. Sharing the phone becomes more difficult.

4. **Dynamic Pin:** Dynamic pin is an application which changes the pin dynamically. User has to calculate the pin using a simple calculation on 2 numbers predefined positions of a grid of numbers[5].

5. **App lock:** There are app locking mechanisms available, which locks only a specific set of apps, letting the rest of the phone accessible easily[6].

   - Advantages: Accessing basic functionality is convenient.
   - Disadvantages: Much less secure than setting a global password.

1.3.2 Relevant Solutions

Here, we present some solutions which take a similar approach to that of ours, to provide a balance of convenience and security.

1. **Android Wear unlock:** Android wear powered smart watches can unlock the device with which they are paired to, by simply bringing them together (Upcoming Android Lollipop feature). This is similar to the context sensitiveness in our app. Disadvantage is that its use is limited, as not everyone has a smart watch, and the watch might not be with users always[7].

2. **Laptop-phone bonding:** Similar solutions exist to unlock a laptop whenever a trusted mobile phone comes within its range[8].

3. **Motorola's Trusted Devices:** In Motorola's Moto X, some Bluetooth devices can be set as trusted devices, and whenever these devices are in the range of the phone, lock screen is not shown[9].

1.4 Our Solution

1.4.1 Explanation

Present day smart-phones have many sensors compared to other consumer computing devices, hence we take advantage of that to understand the kind of environment (or context) the user is and decide the level of security required. For this we give the phone the intelligence to detect the current environment of the user and his security preference for that environment. The environment (or context) the user is present in is determined by environment variables. The presently used environment variables are (See Figure 1.1):

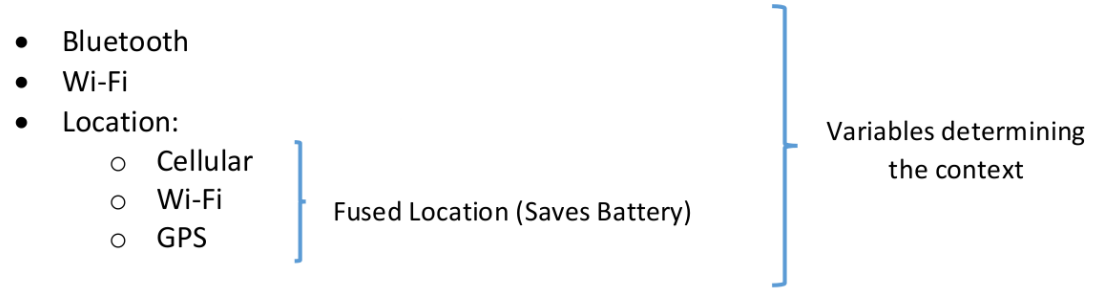


Figure 1.1: Environment Variables presently used in the app

1. Bluetooth devices
2. Wi-Fi networks
3. Location

The security methods available for unlocking the device are called pass-phrases. There are four different types of Pass-phrases (in decreasing order of security, increasing order of convenience):

1. Password
2. Pin
3. Pattern
4. None

Every environment is mapped to a pass-phrase. The type of pass-phrase is dependent on the level of security required, and the level of security required depends on the environment. The example provided in the following section provides a simple illustration.

Notifications are an important part of modern day smartphone experience. All modern mobile operating system platforms have this feature in one way or another[10][11]. Many a time, giving information about pending notifications to the user right from the lock-screen may even lead to the user not requiring to unlock the phone. Clearly, this is an added convenience for the user and hence our solution also incorporates making notifications more accessible, by displaying them in the lock-screen. Users can open the corresponding application related to the notification by double tapping it (they will be asked to input the current reference password before taken to the app), or dismiss it by simply swiping it away.

We share our phones with others many a time, for a variety of reasons, like allowing a friend to make a call, or browse the web etc. These situations can place the user privacy at risk, since the user data is exposed to whoever is using the phone, without any restrictions. To ensure security in this scenario, multiple user profiles can be used.

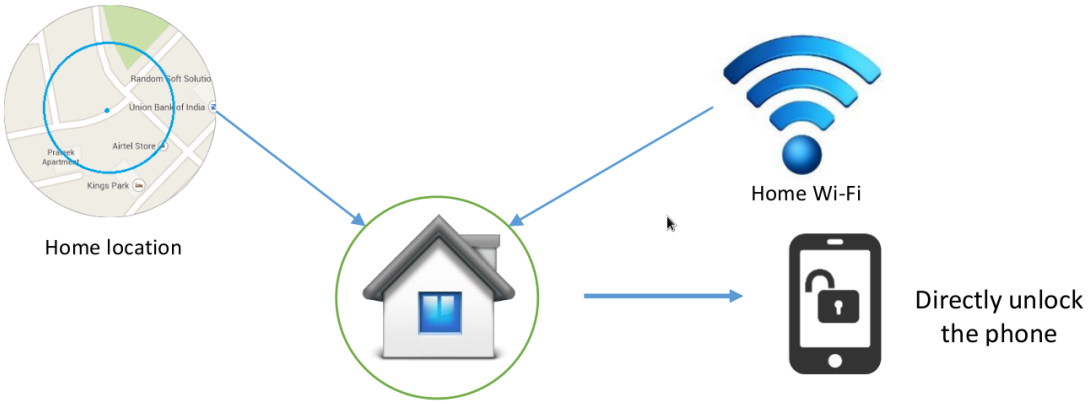


Figure 1.2: An example environment

The device owner can set up different restricted user profiles, and whenever they share their phone, those profiles can be used, which limit access to only certain apps. The ability to switch between users is given in the lock-screen. There are two kinds of profiles: a device owner profile, set up automatically by the app when it runs for the first time, and restricted user profiles, which allows only certain apps to be accessed.

1.4.2 Example

Here is a simple example which demonstrates a simple use of the app. A user wants no password set in her phone when she is in her house. But whenever she is outside her home, she wants her phone to be locked, to keep her sensitive data from prying eyes. To achieve this, she can define an environment in our app(Figure 1.2):

Environment : Home

Security level : Minimal

Environment Detection : The user is at home if he/she is approximately at location (22.659086, 75.907903) and is connected to the home Wi-Fi.

Pass-phrase : None (security level is minimal)

And for unknown environment, pass-phrase can be set as a secure password or pin.

Detailed use cases can be found in section 3.1.

1.4.3 Mathematical Formulation

1. **Defining Environment**

   Let the number of sensors used for defining an environment be n_s. Let us call the sensors $s_1, s_2, \ldots, s_{n_s}$.

Then, environments are defined by the function f_{se}, where

$$f_{se} : D_1 \times D_2 \times \cdots \times D_{n_s} \rightarrow E$$

where D_i is the value domain for sensor $s_i, \forall i = 1, 2, \ldots, n_s$, and E is the set of environments.

The task of evaluating and assigning a value to each sensor, is the job of the sensor. This can be modelled as a set of valuation functions. Each sensor has an evaluation function:

$$v_i(s_i) \in D_i$$

2. **Associating pass-phrases to Environments** This can be modelled using another function f_{ep}, where

$$f_{ep} : E \rightarrow \mathbb{P}$$

where $\mathbb{P}$ is the multiset of Pass-phrases. This mapping is one-one and onto. Thus, if n_e is the number of environments,

$$\mathbb{P} = \{p_1, p_2, \ldots, p_{n_e}\}$$

2. Introduction to Android

2.1 The Android platform

Android is a mobile operating system based on the Linux kernel and currently developed by Google. It's been built from the ground up, keeping in mind the strengths and limitations of mobile devices. The Android stack is succinctly portrayed by Figure 2.1. A brief introduction to each of the layers, with special emphasis to the components

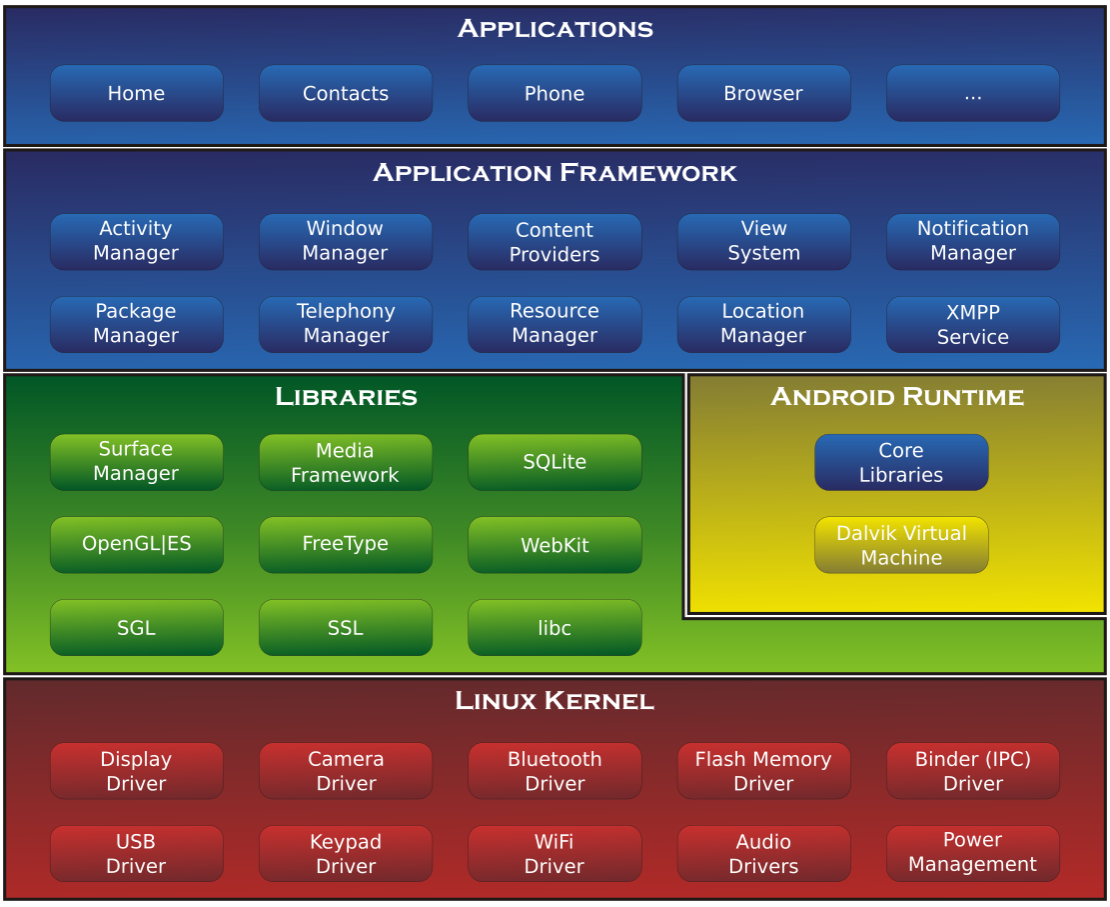


Figure 2.1: Android's architecture diagram. [12]

which we're using, are detailed in sections 2.1.1 to 2.1.4.

2.1.1 Linux Kernel

At the core of Android, is a modified Linux kernel. Various enhancements are patched to it to support Android, like better power management, alarms, better inter process communication through Binder etc.

2.1.2 Libraries

Libraries include native libraries like bionic libc(custom libc implementation, optimized for embedded use), Webkit, media framework, SQLite(light-weight transactional data store), the Hardware Abstraction Libraries etc.

In our project, we use SQLite for handling all the data storage. The schema and other details related to the database can be found in section 3.3.

2.1.3 Runtime

Till Android KitKat(4.4.x), Dalvik was the default runtime in Android. Dalvik Virtual Machine is Android's custom clean-room implementation virtual machine. It provides application portability and runtime consistency, and runs optimized file format (.dex) and Dalvik bytecode. But with Android Lollipop(5.0), Android RunTime(ART) has replaced Dalvik as the runtime in Android. ART has several enhancements over Dalvik like Ahead-of-time (AOT) compilation and better garbage collection. Details of compiling apps to run in this runtime are explained in section 2.3. The core APIs are in Java language and provides a powerful and familiar development platform.

2.1.4 Application Framework

Application Framework contains core platform services that are essential to the Android platform. Some key services used by our app are discussed below:

Activity Manager Used directly for implementing restricted access to apps.

Window Manager Used to draw overlay over the lock-screen to show custom contents

Content Providers Content providers are used to abstract away direct access to the database, and even allows data to be effectively shared between different app components.

2.1.5 Android SDK and APIs

The Android SDK provides API libraries and developer tools necessary to build, test, and debug apps for Android. The main app components which are used are detailed below.

Activities and Services

In Android, an application component with a screen is called an Activity. Activities are the primary way for users to interact with any application in android. An activity typically takes up the entire working space in the screen when it is in the foreground, and allows the user to interact with it, primarily via touch. Each activity has its own lifecycle, and it might cycle through various states during its life time. An activity is stopped immediately when user navigates to another activity, meaning that activities cannot run in the background. However, it could still be kept in memory, though such activities which are not in the foreground are usually the first candidates to be killed, if the system is running low in memory and requires some memory to be freed up. Activities are used in our app to display many user visible windows in our app. Activities to see an overview of added environments, add/edit environments, set master password, resolve environment overlap etc are available.

Long running background tasks require Services in Android. A service does not have a visible component, and hence its memory footprint is lighter than that of an activity. Services are essential when an app must continue to run in background and perform tasks. Our app contains a central service which accepts requests from various components and keeps the app running in background, while doing environment detection and password change.

Broadcast Receivers

Broadcast Receivers are used to receive broadcasts from the system as well as other apps, so that appropriate actions can be taken. For example, a Wi-Fi network change or a bluetooth device connection or even battery level getting low are broadcast throughout the system, and broadcast listeners get to know when these events happen, with possible extra information added to the broadcasts.

Intents

Intents are convenient inter process communication provided in Android. They are used in starting activities, services, and getting other information.

Google Play Services

Google Play Services breaks away some core APIs from the platform and makes it available for all devices running Android 2.3 and above. The advantage is that these APIs can be updated without pushing out a full platform upgrade, and any device will have the same set of APIs at any point of time. Google Play Services is distributed using Play Store. We use the location services provided by Google Play services, since it will be more efficient and battery saving to have a central point for location access, instead of making use of GPS directly. Additionally,

it provides a 'fused location', as in we can get a rough estimate of the location using Wi-Fi and or cell towers even when GPS is not available.

Wi-Fi, Bluetooth and Sensor Framework
APIs are provided by Android to access paired Bluetooth devices and saved Wi-Fi networks. Sensor framework is used for obtaining other sensor data.

Device Admin
An app can request Device Admin permissions. This is required to change password in the background without user interaction.

2.2 Why Android?

Popularity Android is the most popular OS (in new devices). As of 2013, Android devices sell more than Windows, iOS, and Mac OS X devices combined[13].

SDK Android has a powerful Software Development Kit(SDK), and lot of tutorials available for beginners, making it an ideal platform to start mobile app development.

Open Sourced The open-sourced nature of Android makes it possible to delve into its code and understand the workings of the system.

2.3 Developing Android Apps

Android SDK is provided by Google for developing Android apps. It contains various tools, resources, documentation and samples to build and debug apps.

Apart from Java source files, an Android project typically contains other directories and files. The essential ones are discussed below.

res folder Contains various application resources like layouts (specified in XML), images, strings, dimensions, etc.

Android Manifest Every android app should have a manifest. It is an XML file which declares the various app components (Services, Activities, Content Providers, Broadcast Receivers) along with their properties, the Permissions used by our app, etc.

R.java It is an auto generated Java file which is used for referencing resources from within Java code

2.3.1 Android application build process

The build process can be summarized using Figure 2.2. The major steps involved are

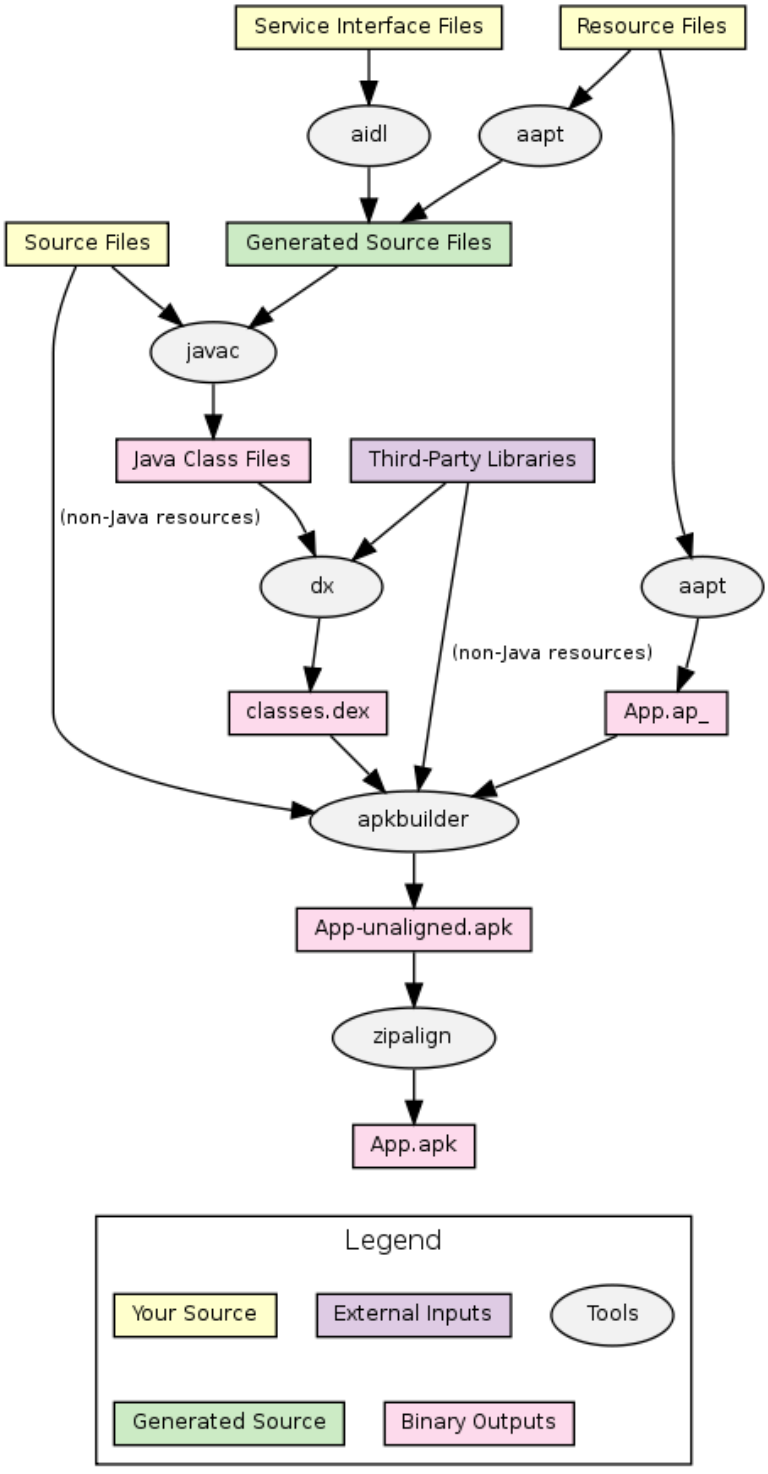


Figure 2.2: Android application build process. [14]

detailed below. [15]

1. **Resource Pre-compilation:** The first step in the build process involves genera-

tion of Java source files from your Android resources. The resources, stored in the res subdirectory, include such things as icons, layouts and strings. These are compiled using the aapt tool into a file named R.java.

2. **Java Compilation:** This step is a standard Java compilation from .java source files (both hand-crafted and generated) to .class bytecode files. One thing to be aware of is the classpath used to compile source. This includes:

   (a) The android.jar file for the target Android platform. This jar includes class and method stubs for all of the Android APIs.

   (b) External library jars you have added to the project (all .jar files in the libs/ subdirectory).

3. **Bytecode Translation:** Android runtime requires a different bytecode format from normal Java bytecode. Thus, after compilation, the dx tool is used to translate class files into a Dalvik executable or .dex file. This includes the class files stored in any external library jars which were added to the project. All classes are package up in a single output file, named classes.dex.

4. **Resource Packaging:** Next, the resources are compiled into a partial Android package file. This is done by the same aapt tool that generates Java source corresponding to the resources.

5. **Packaging and Signing:** Now all of the components required for the final Android package are ready to be bundled up into an apk file named after the application. In the default debug mode, this build step also includes signing of the package with a debug key. Note that for release, signing is a separate step that requires access to a key (and may prompt for a password). Android packages are assembled with the apkbuidler tool, which takes input from several sources:

   (a) The Dalvik executable file classes.dex

   (b) All non-Java resources from the source directory

   (c) All non-Java resources from the external libraries (found by searching all .jar files in the libs/ subdirectory)

   (d) Any native code shared-libraries included by the project

   (e) The resource package built in the previous step

6. **Alignment:** As a final optimisation step, the package file is aligned using the zipalign tool. This step ensures that resources in the package file are aligned on 4-byte word boundaries. This allows the runtime to memory-map those parts of the file for more efficient access.

Thus the final result of this build process is an apk file, which can be installed on an android device using package manager.

2.3.2 ADB and Logcat

Android Device Bridge (ADB) is a tool using which we can issue shell commands to an android device, and this can also be used to install and run apps in the device. Logcat is the log monitoring tool in Android, and comes in handy to identify and isolate bugs.

2.3.3 Android Studio

Android Studio is an IDE based on IntelliJ IDEA, for developing android apps. It uses gradle build system to automate android app build process, and provides many advanced features like powerful code refactoring capabilities, integrated log monitoring, version control system integration, etc. so that the entire development cycle can be managed from within [16]. We are using Git for version control and collaboration between the team members.

3. Specification of the system

3.1 Client Specification - Use cases

1. **App First Use**: When the app is opened for the first time, the app should introduce the user about the basic interface, and ask the user to set two basic environments: Home and Office, and automatically set the password for both the environments. The app should also request the user to set a master password.

2. **Adding Environments**: There should be an interface for adding new environments and rules associated with it. The interface should automatically detect invalid data and warn the user before creating the environment. The interface should also allow the user to choose the pass-phrase type and also set the pass-phrase.

3. **Managing Environment**: There should be an interface for the user to manage already created environments. The user should be able to edit the rules, switch on/off environments and also delete environments. The interface should allow the user to modify the pass-phrase type and also modify the pass-phrase. Option to set pass-phrase for unknown environment should also be present.

4. **Overlap resolution**: The app should have a suitable environment overlap resolution method, and also provide an option to the user to override the choice made by this algorithm.

5. **Smart Overlay**: The 'Smart Overlay' should be shown on top of the lock-screen when the user switches on the screen. This overlay should contain the notification which will have different access level corresponding to the current environment, and should also display an option to switch between restricted profiles (additional) and also swiping away the screen should lead to the Lock screen.

6. **Lock screen security**: The app should be able to detect transitions from one environment to another. When the user chooses to unlock the phone, the app should notify the user about the current environment and the unlock method should be changed corresponding to the present environment and the pass-phrase should be matched corresponding to the present environment.

7. **Settings**: The settings interface should have the option to start the 'Add' and 'Manage' environment interface, and should also allow the user to edit other app preferences like global on-off switch, the feedback option etc.

8. **Restricted Profiles**: The user should be able to add restricted profiles which will enable restricted access to phone's applications. The user should be able to whitelist the apps for the profile and also select the type of pass-phrase and set the pass-phrase for the profile.

9. **Notification setting**: The user should be able to set the notification access in lock-screen overlay preference corresponding to different environments.

3.2 Developer Specification

3.2.1 Event Flow

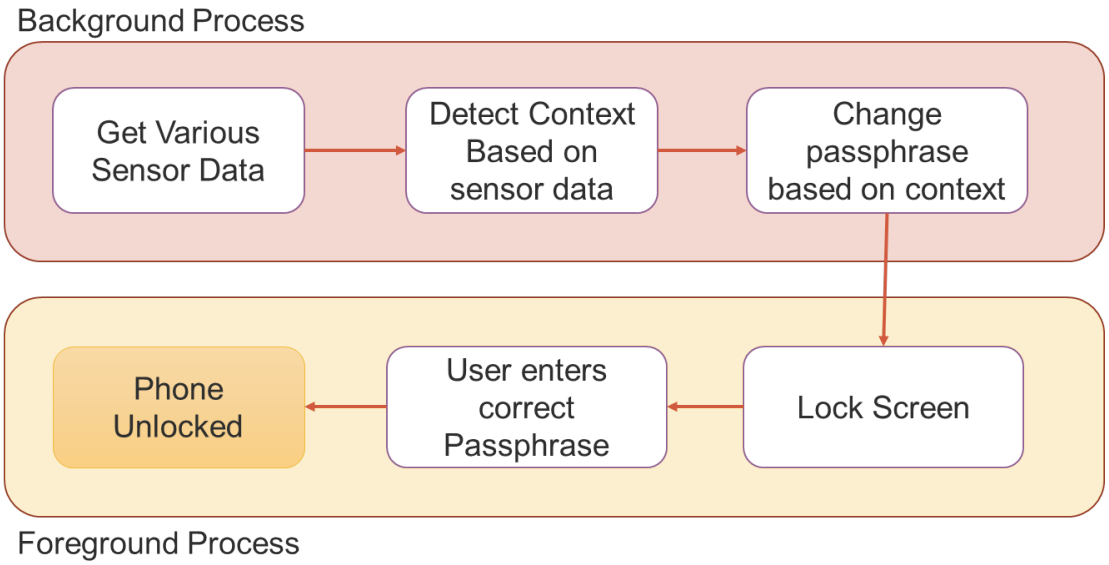


Figure 3.1: Event flow in the app

The event flow of the app to detect environment and accept the correct password is shown in Figure 3.1.

3.2.2 Package Hierarchy

The package hierarchy is shown in Figure 3.2. For links to complete source code and documentation, see Appendix A.

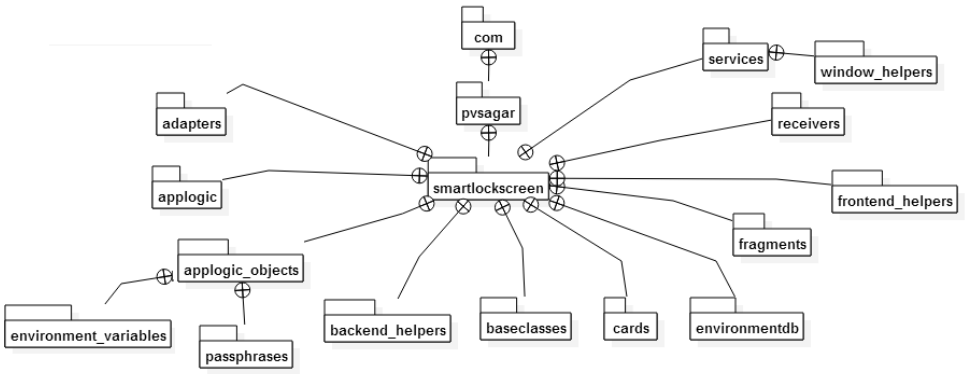


Figure 3.2: Package Hierarchy

3.2.3 Class Hierarchy

The class hierarchy is defined in Figures 3.3 to 3.11.

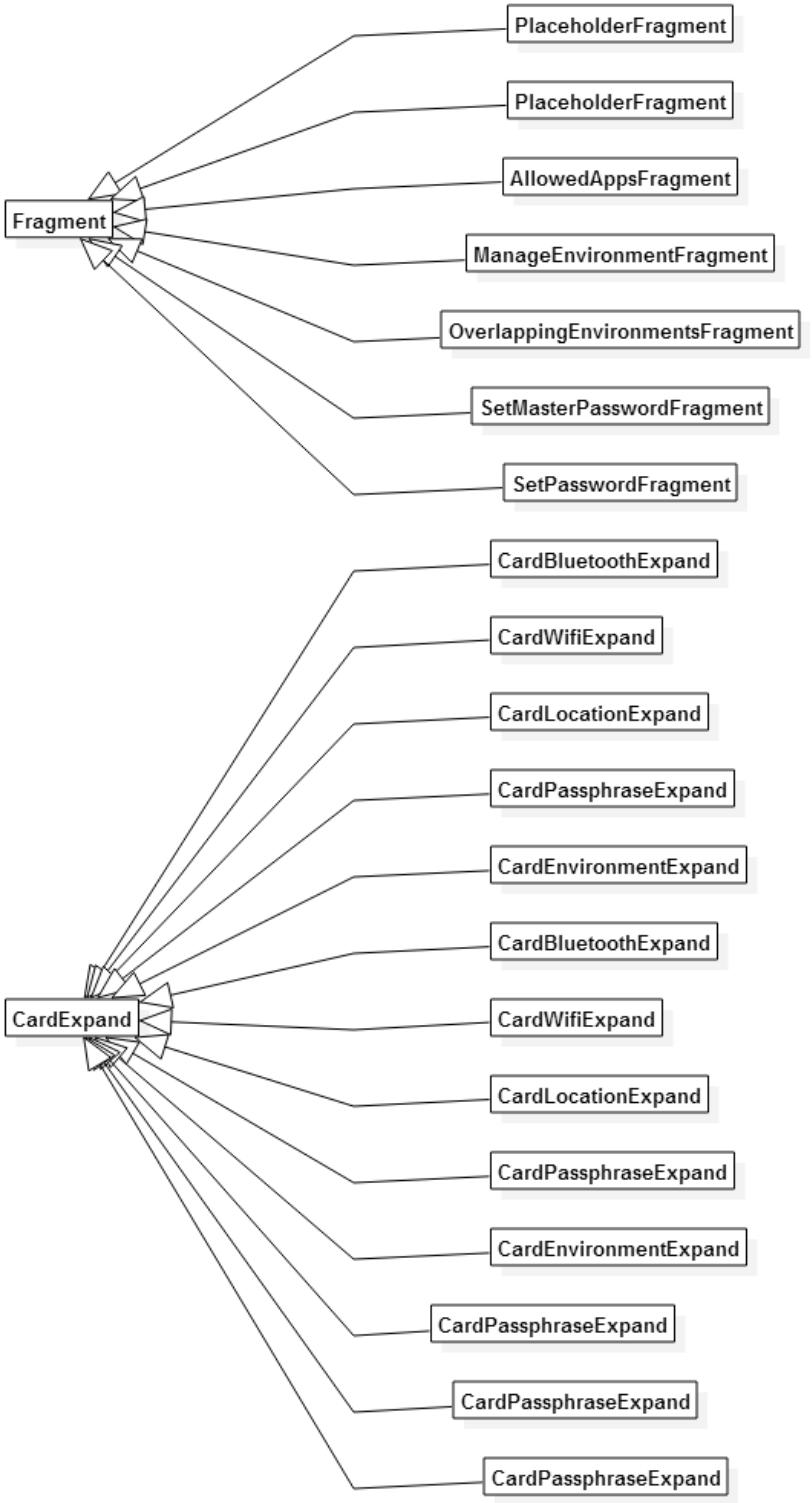


Figure 3.3: Class Hierarchy - I

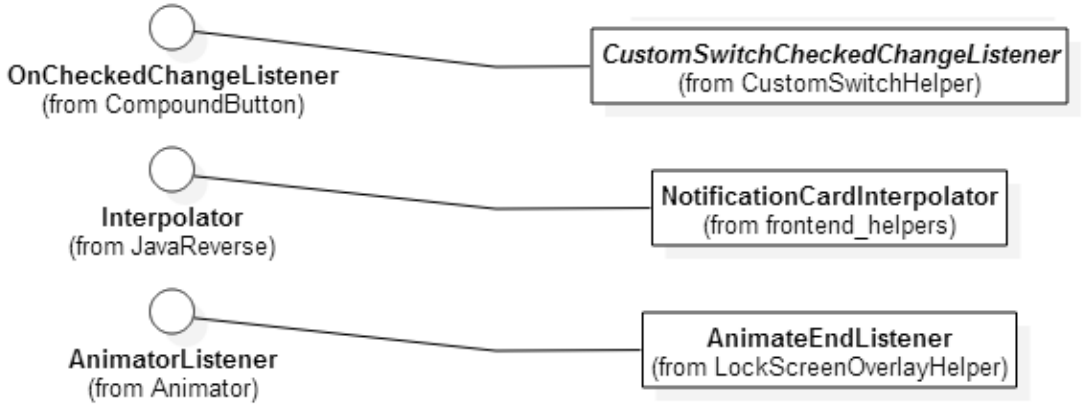


Figure 3.4: Class Hierarchy - II

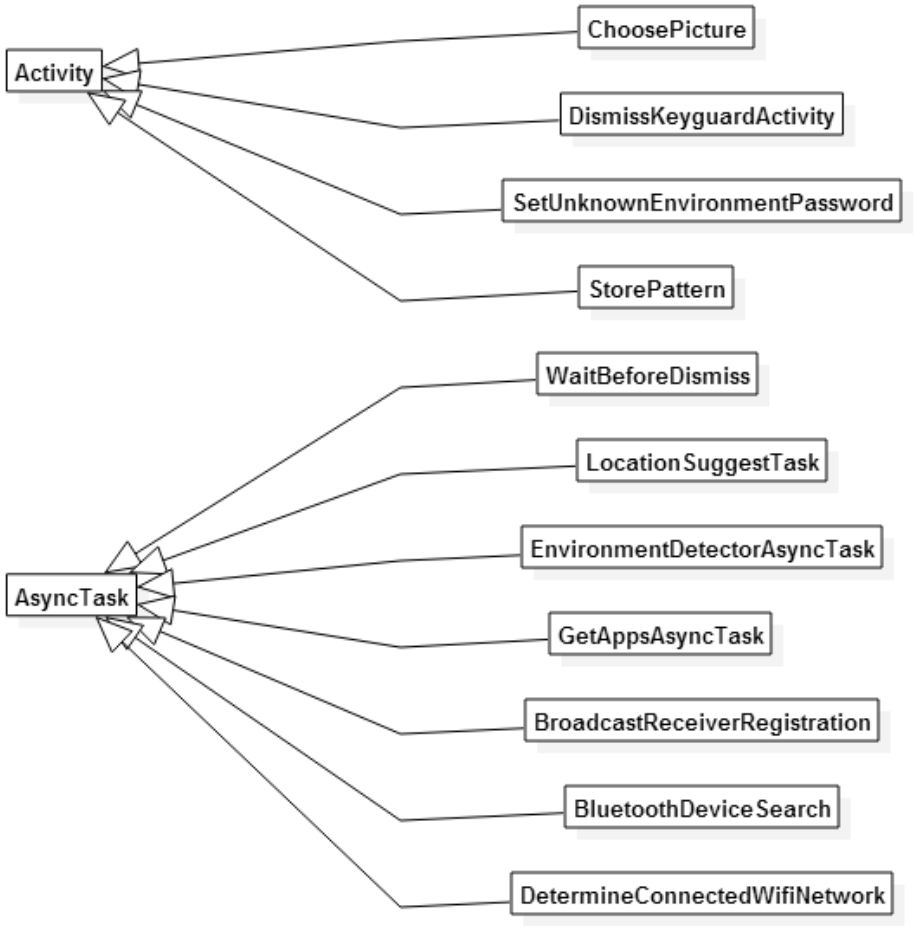


Figure 3.5: Class Hierarchy - III

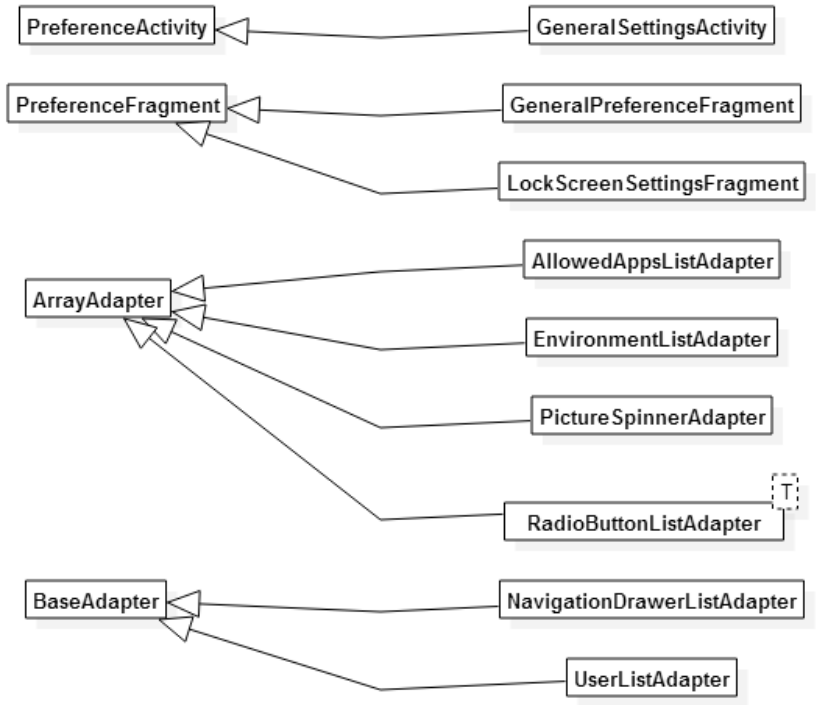


Figure 3.6: Class Hierarchy - IV

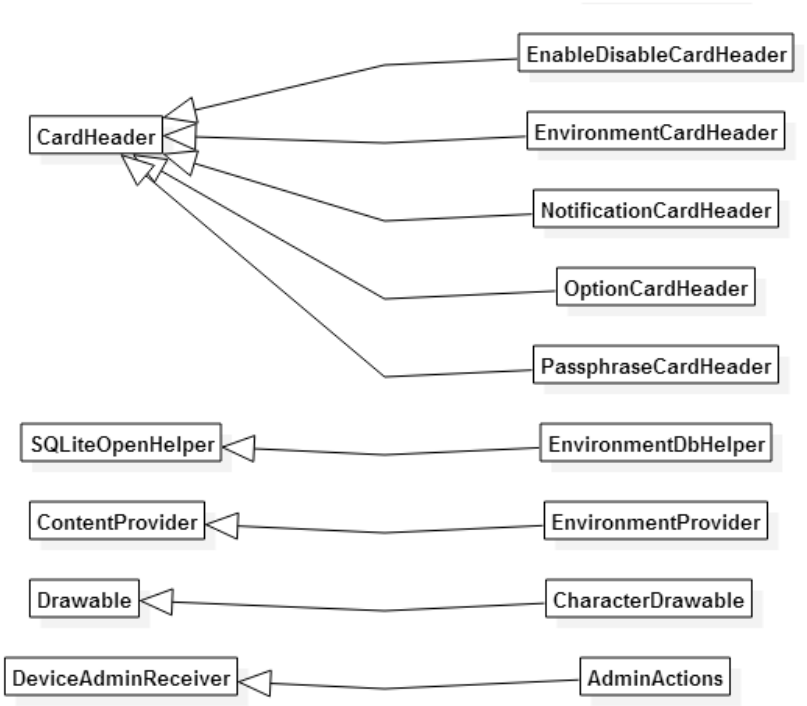


Figure 3.7: Class Hierarchy - V

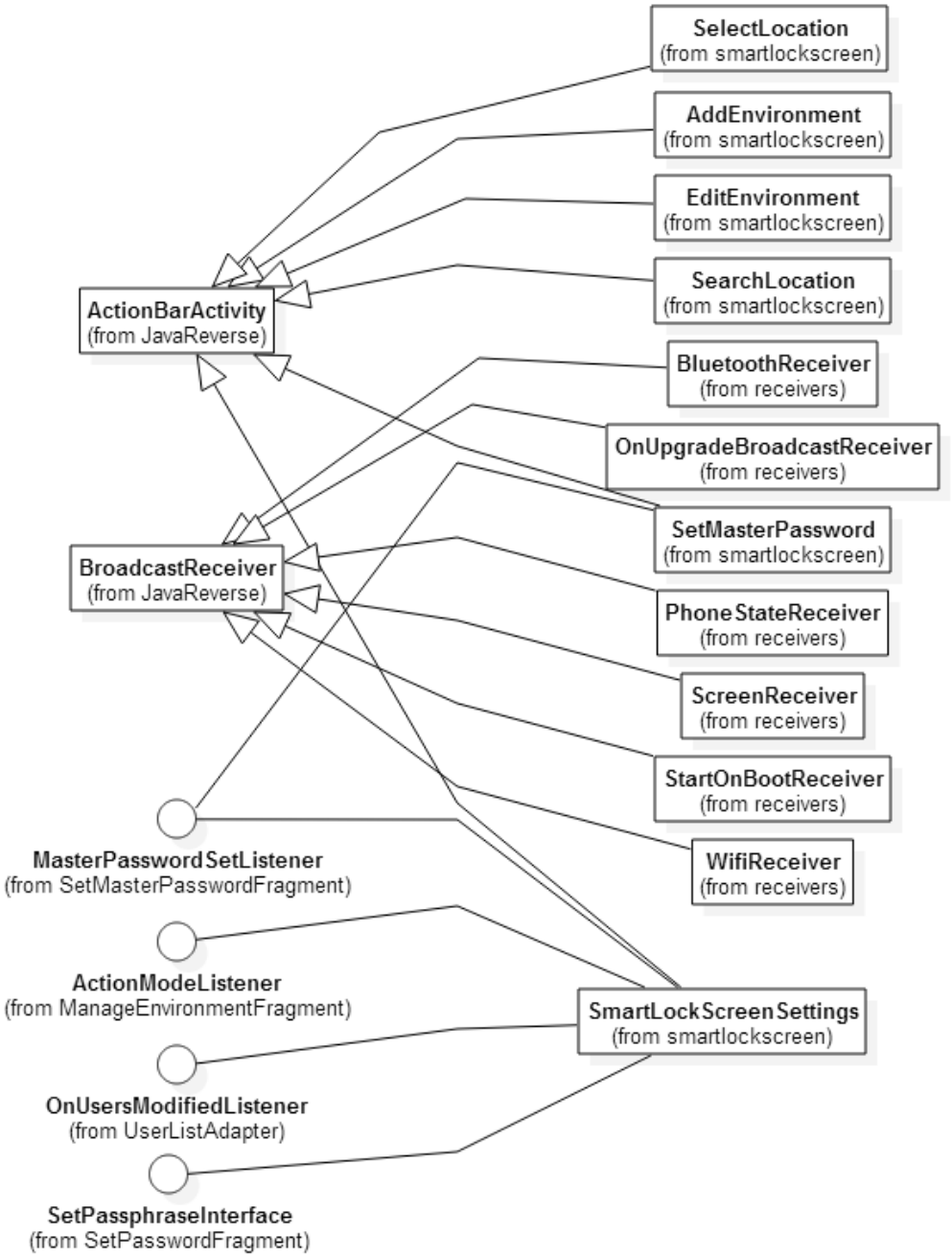


Figure 3.8: Class Hierarchy - VI

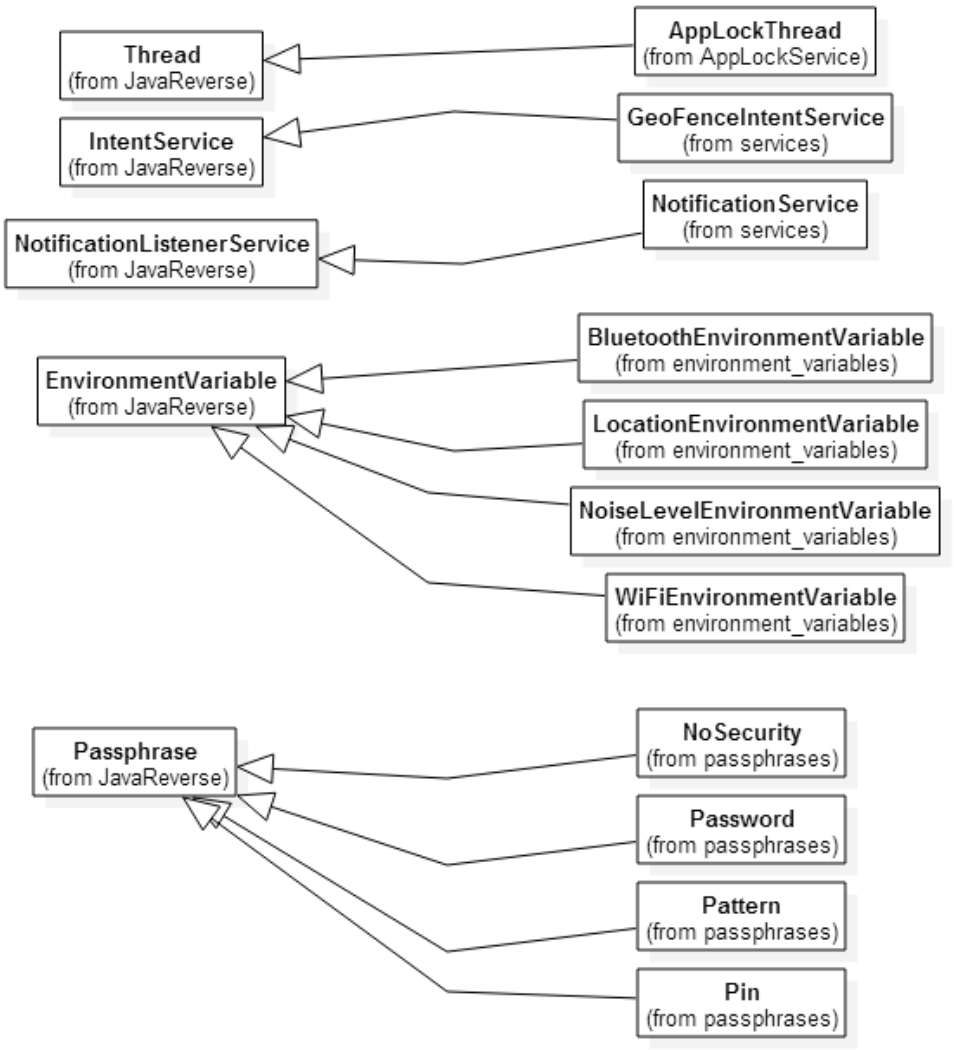


Figure 3.9: Class Hierarchy - VII

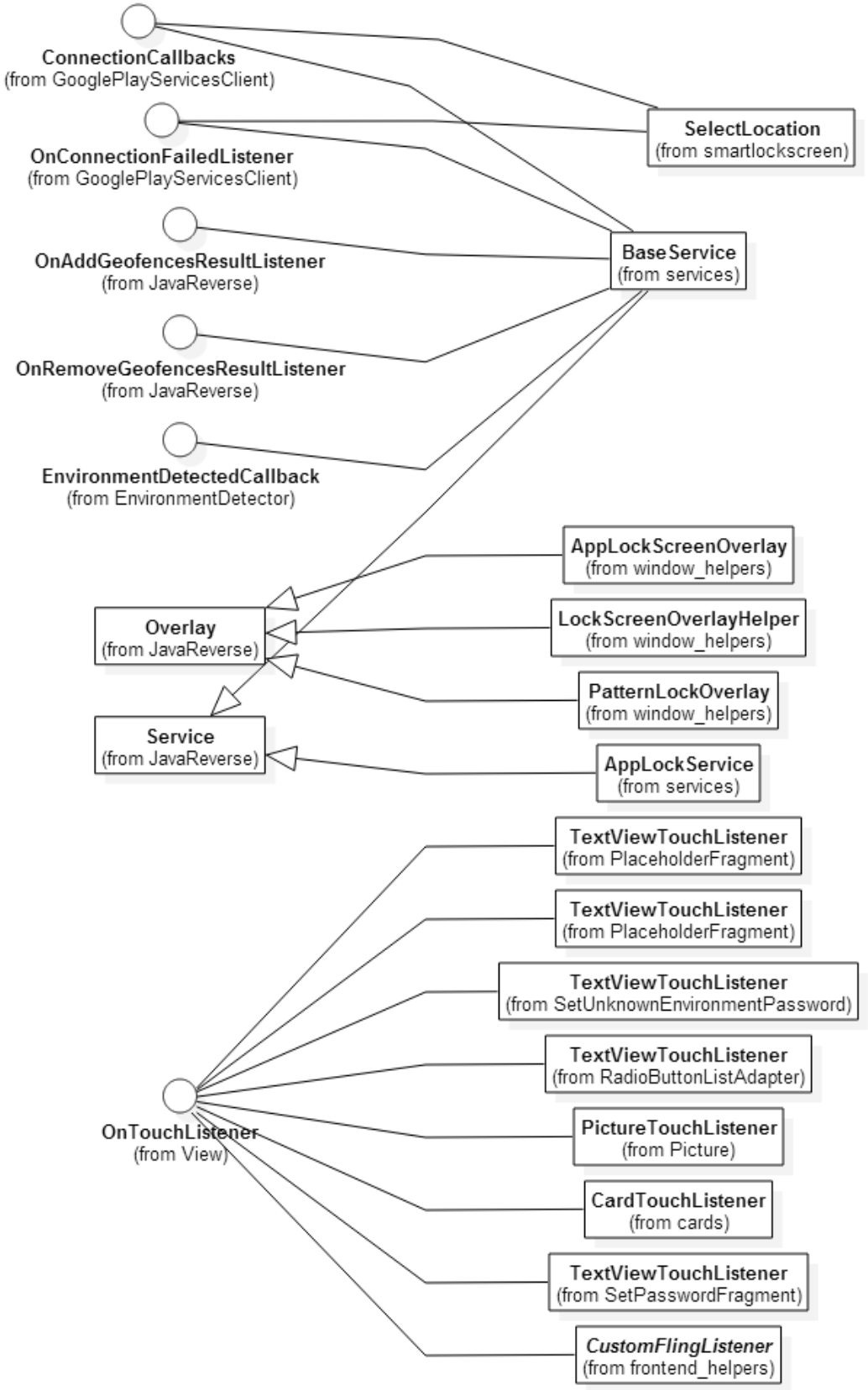


Figure 3.10: Class Hierarchy - VIII

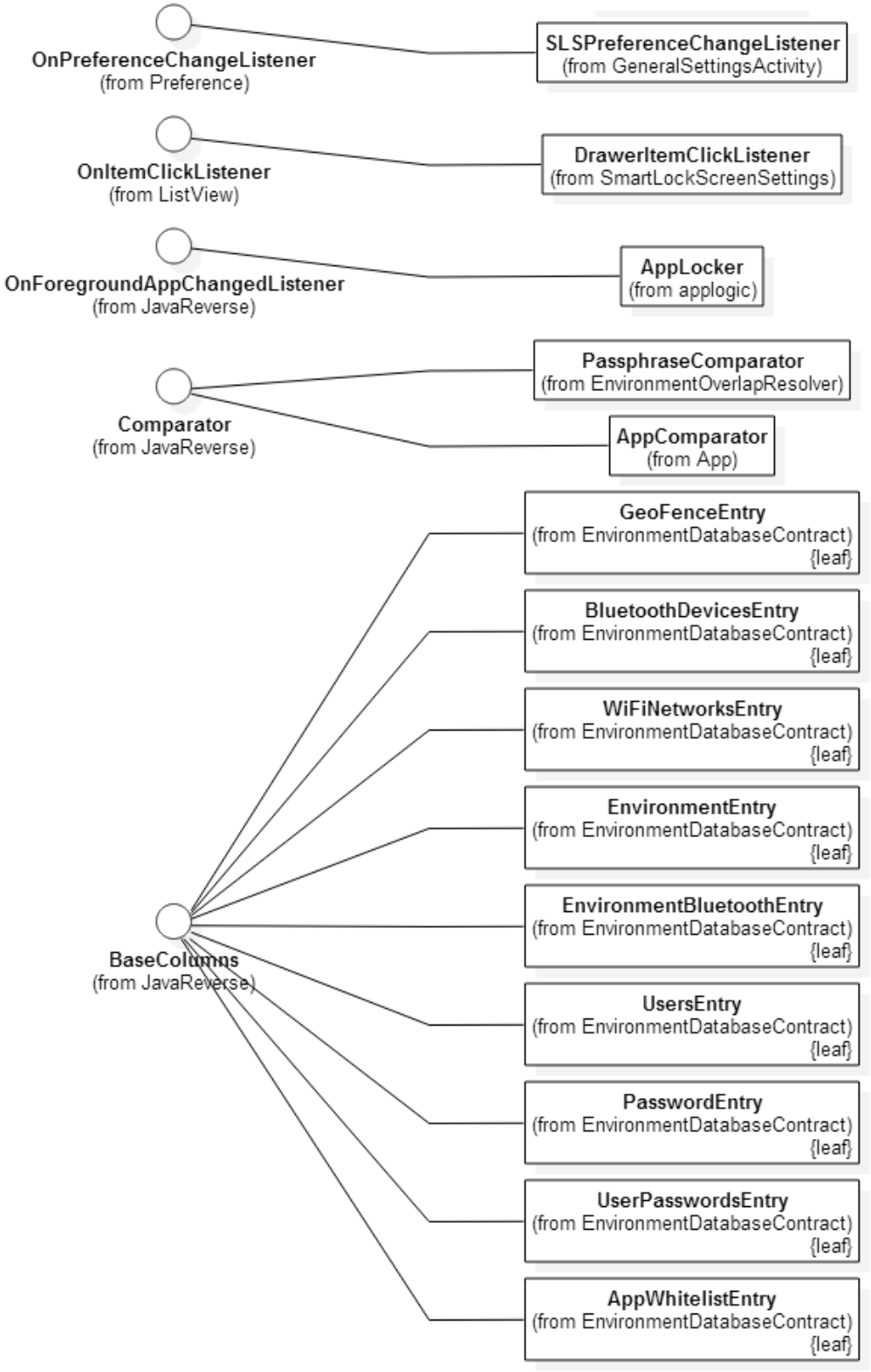


Figure 3.11: Class Hierarchy - IX

3.3 Database schema

The database schema is shown in Figure 3.12.

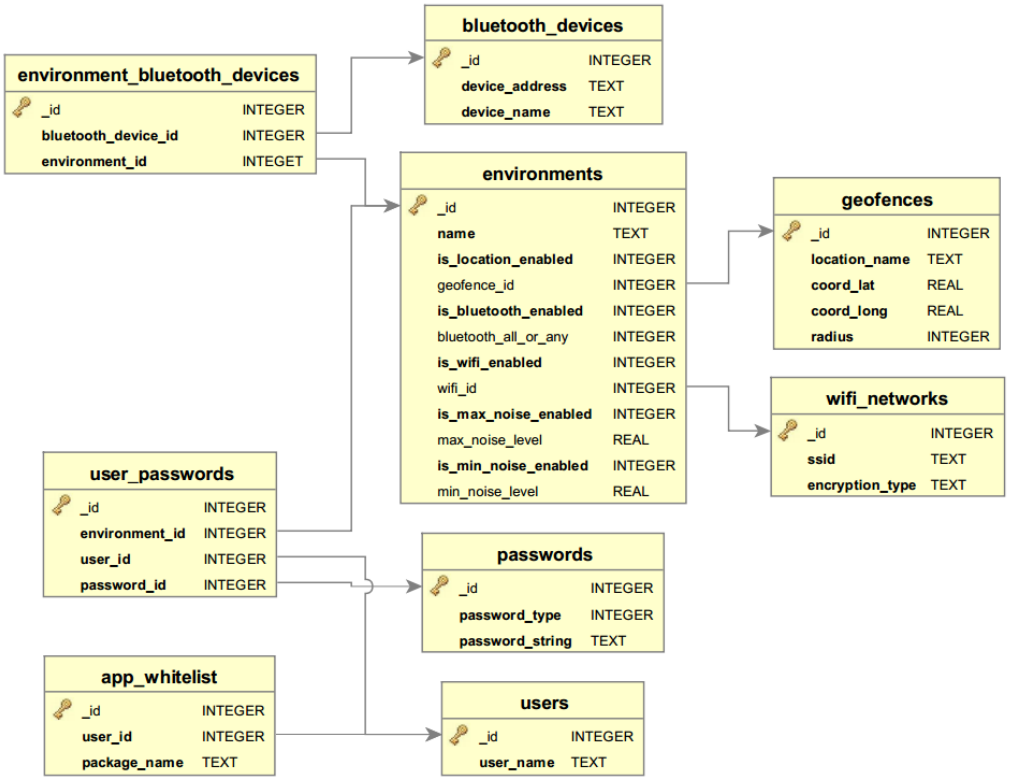


Figure 3.12: Database Schema

3.4 Package diagrams

Package diagrams which shows sub-packages and classes can be found in figures 3.13 to 3.27. Explanations for some of the packages can be found in section 4.2. For links to complete source code and documentation, see Appendix A.

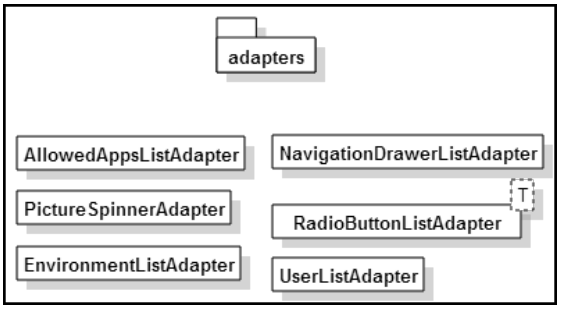


Figure 3.13: adapters package

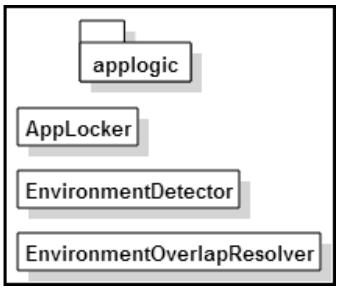


Figure 3.14: applogic package

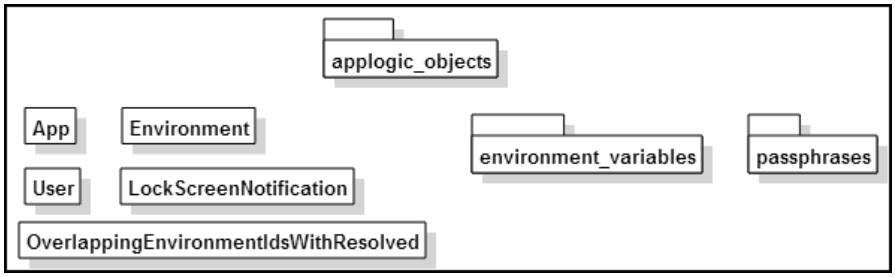


Figure 3.15: applogic_objects package

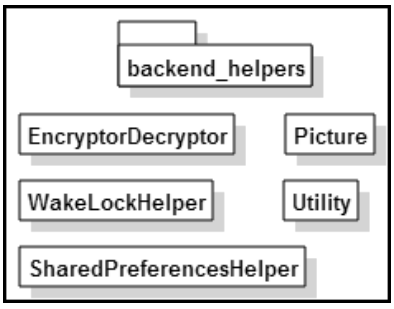


Figure 3.16: backend_helpers package

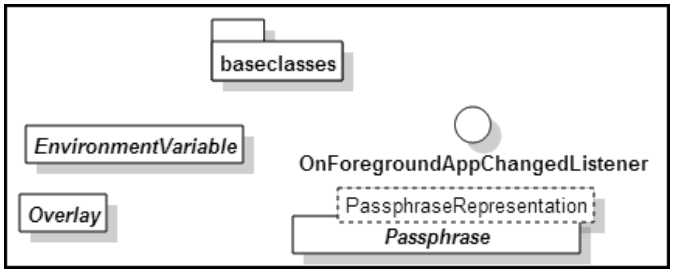


Figure 3.17: baseclasses package

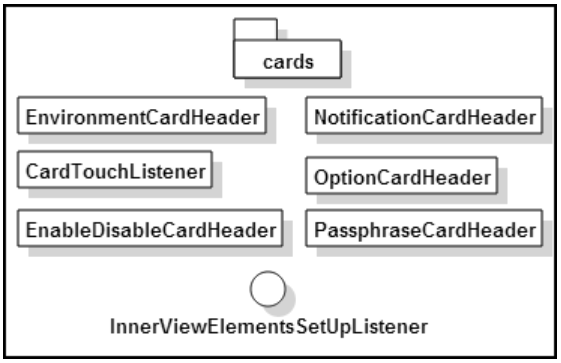


Figure 3.18: cards package

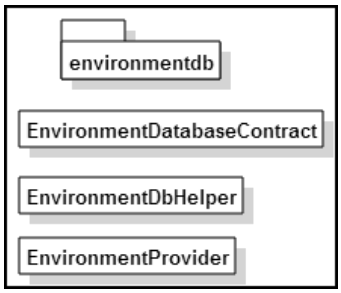


Figure 3.19: environmentdb package

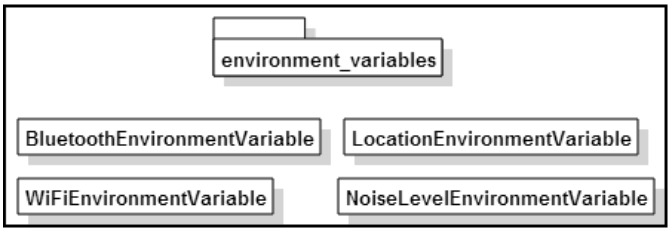


Figure 3.20: environment_variables package

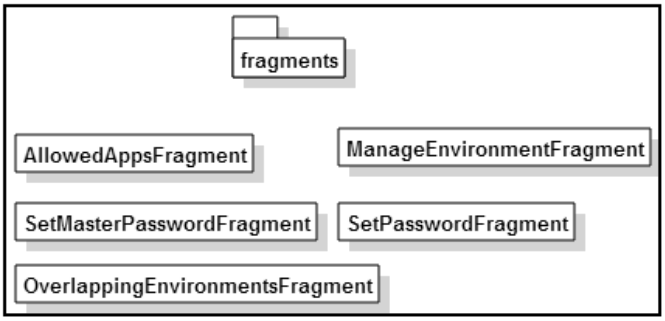


Figure 3.21: fragments package

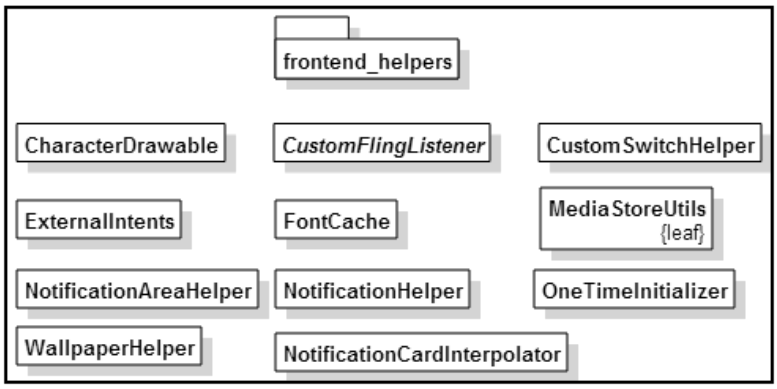


Figure 3.22: frontend_helpers package

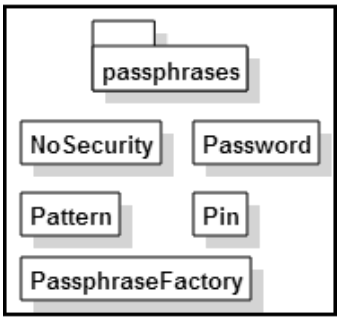


Figure 3.23: passphrases package

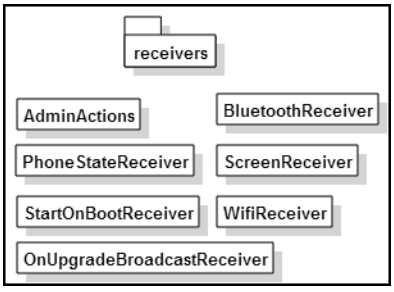


Figure 3.24: receivers package

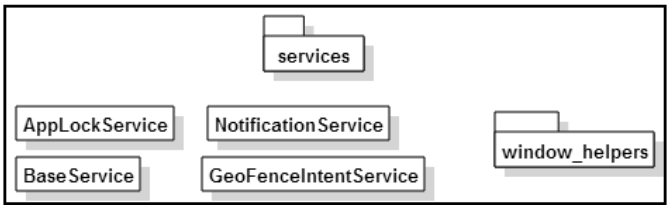


Figure 3.25: services package

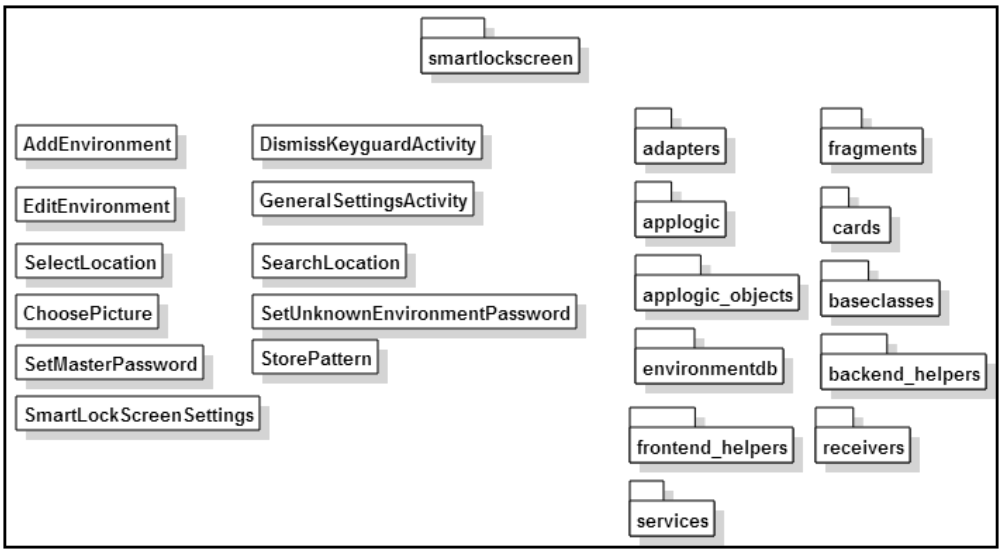


Figure 3.26: com.pvsagar.smartlockscreen package

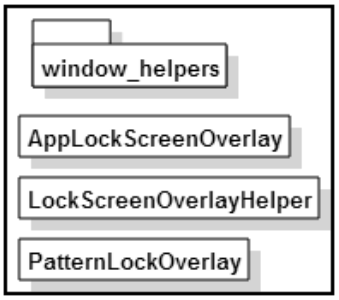


Figure 3.27: window_helpers package

3.5 Class diagrams

Class diagrams of important back-end classes are shown in figures 3.28 to 3.34. Implementation details along with related classes can be found in section 4.1. For links to complete source code and documentation, see Appendix A.

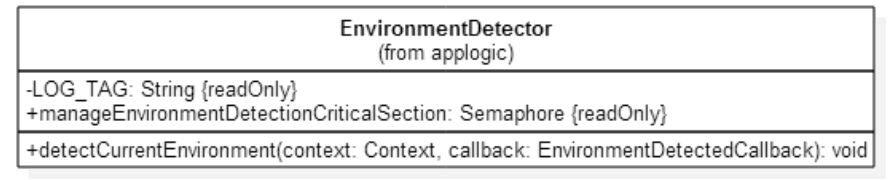


Figure 3.28: EnvironmentDetector class

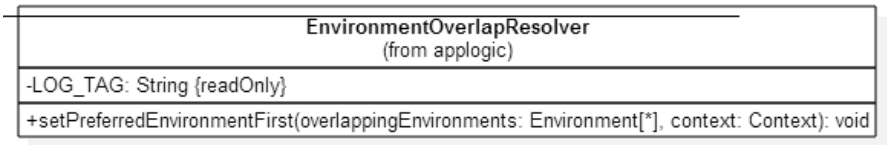


Figure 3.29: EnvironmentOverlapResolver class

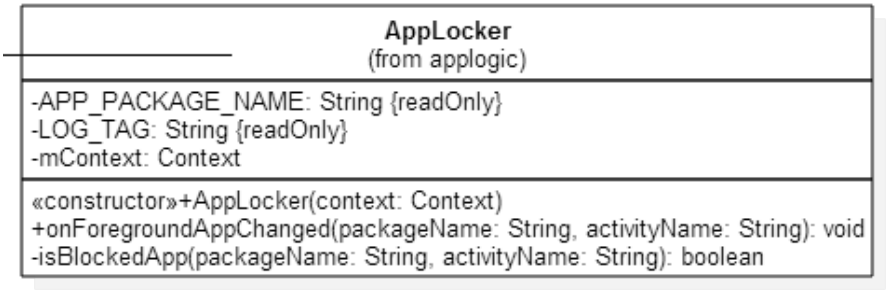


Figure 3.30: AppLocker class

Figure 3.31: Environment class

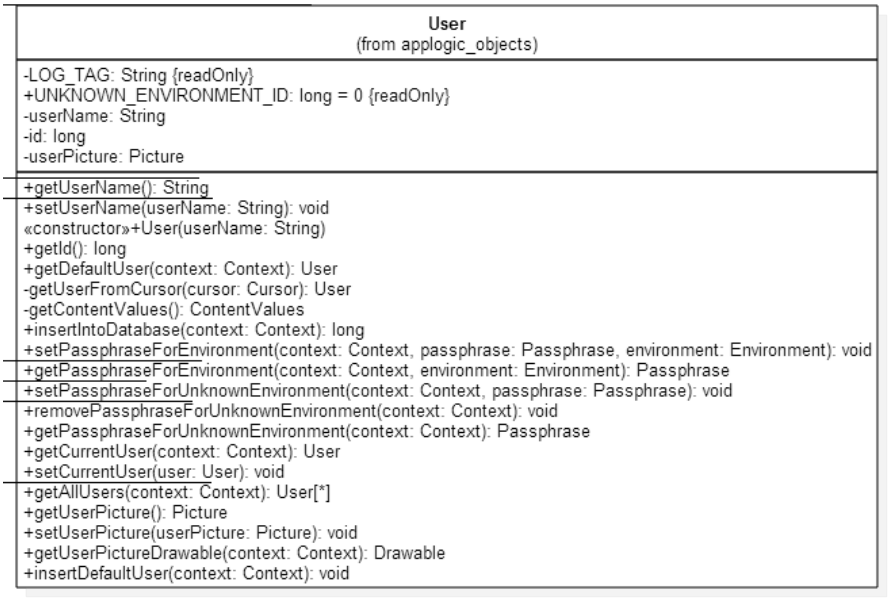


Figure 3.32: User class

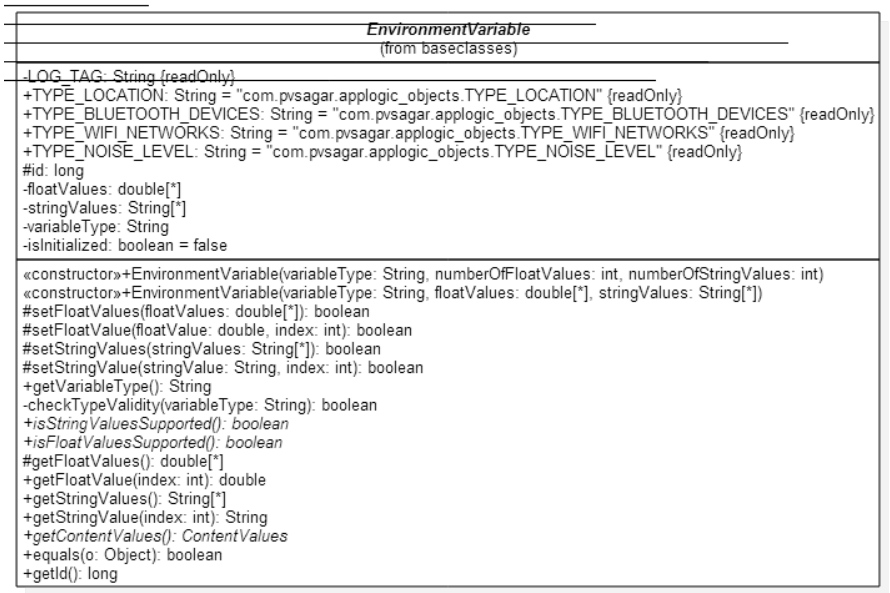


Figure 3.33: EnvironmentVariable abstract class

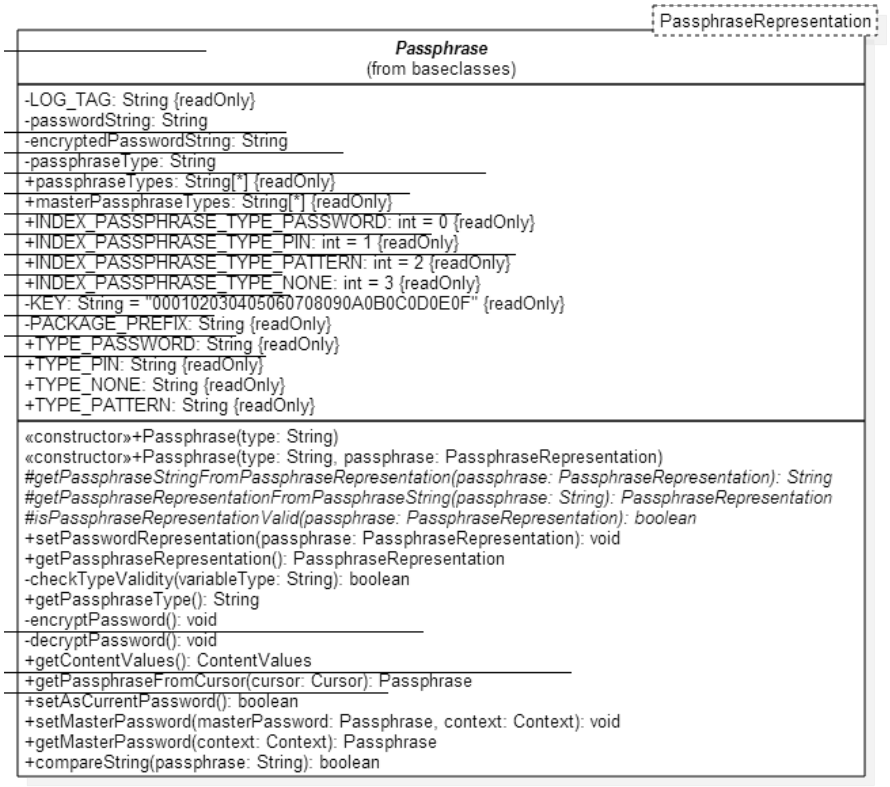


Figure 3.34: Pass-phrase abstract class

4. Implementation Details

4.1 Features implemented so far

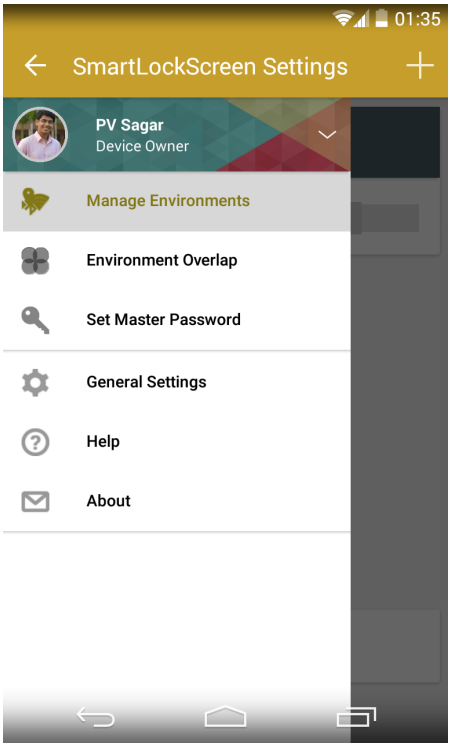
4.1.1 Managing Environments and Users

The user interface for managing environments and users has been implemented. The main activities are the following:

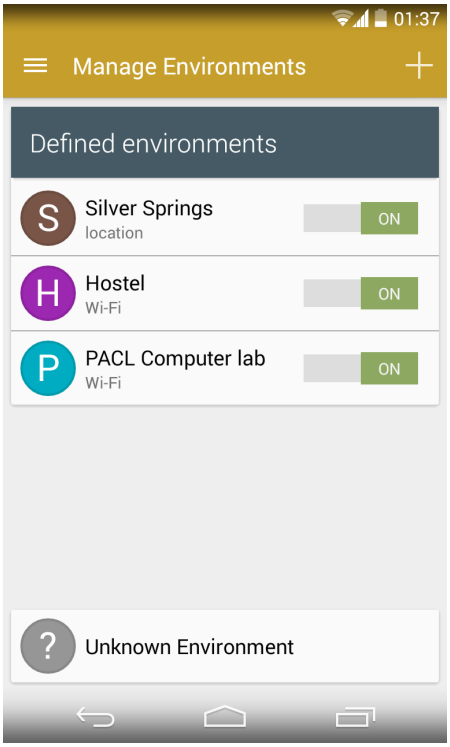
SmartLockScreen Settings Activity This is the central point of the user interface for managing all the settings related to the app. This satisfies part of use case 7. The interface is according to Google design guidelines. We have a beautiful Navigation Drawer, which slides in from the left, giving access to various screens of the app (Figure 4.1a). Related class: com.pvsagar.smartlockscreen.-SmartLockScreenSettings. At the top of the drawer, we have user controls, where user for whom settings are being currently changed, is shown. A profile for Device Owner is made automatically at first start of the app. Clicking on the user area will reveal all the restricted profiles added. Settings for any of those profiles can be changed by clicking on the respective profile. Option for adding new profiles is also available in this list.

The main entries of the navigation drawer will change according to the type of profile currently chosen. If the profile under management is of 'Device Owner' type, 3 options related to managing environments are shown. Clicking on any of them replaces the main area of Settings activity with the corresponding 'Fragment'. Each fragment is a view which can be placed in activities. The fragments are detailed below.

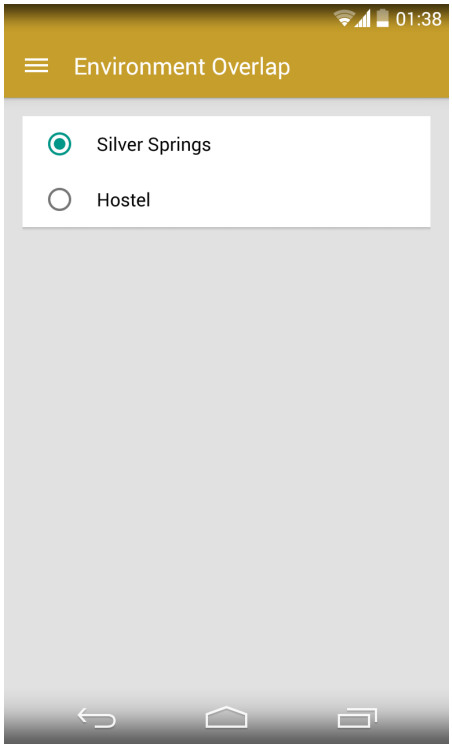
1. **Manage Environments fragment(Figure 4.1b):** An overview of all the added environments are shown, with an option to switch environments on or off.
   - A menu item '+' is available to add new environments.
   - Clicking on an existing environment takes it to edit mode.
   - Environments have pictures associated with them, to make them more user friendly and easily identifiable, as these environment names and



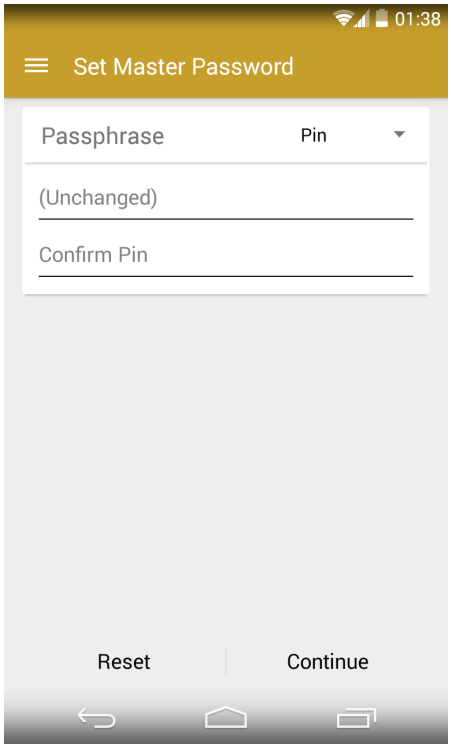
(a) Navigation Drawer of SmartLockScreen Settings Activity



(b) Manage Environments Fragment for Device Owner

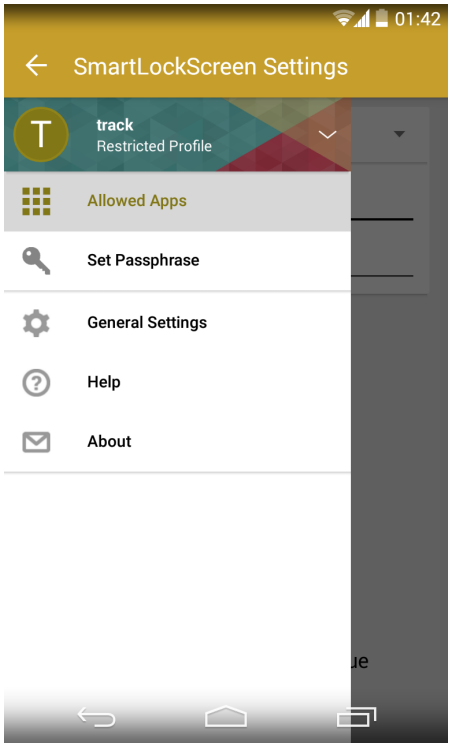


(c) Environment Overlaps Fragment

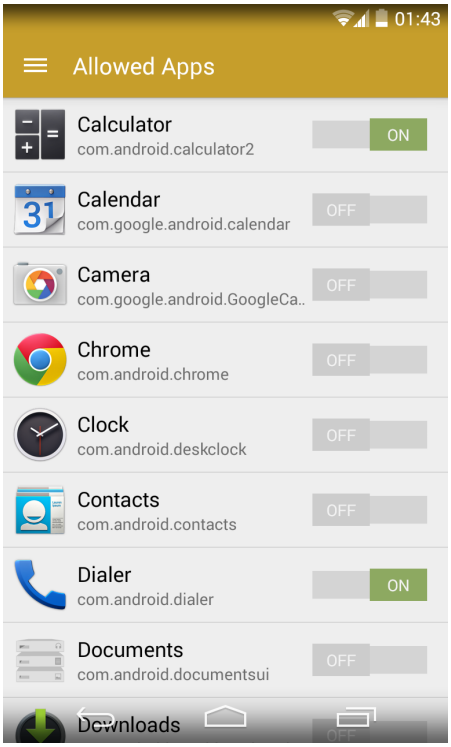


(d) Set master password Fragment

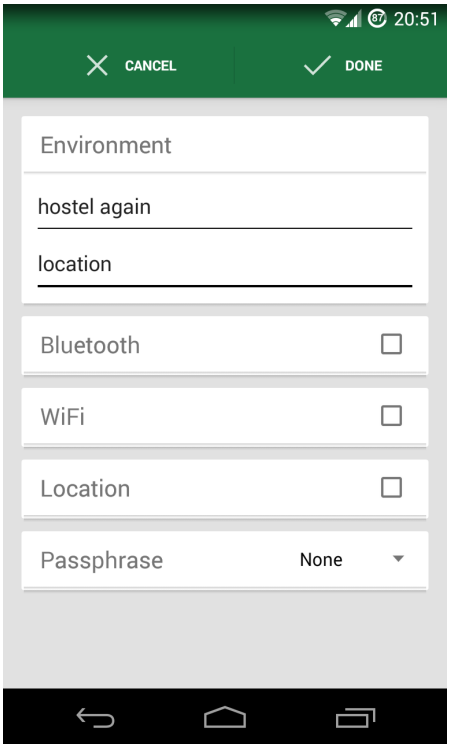
Figure 4.1: Screenshots of the app - I



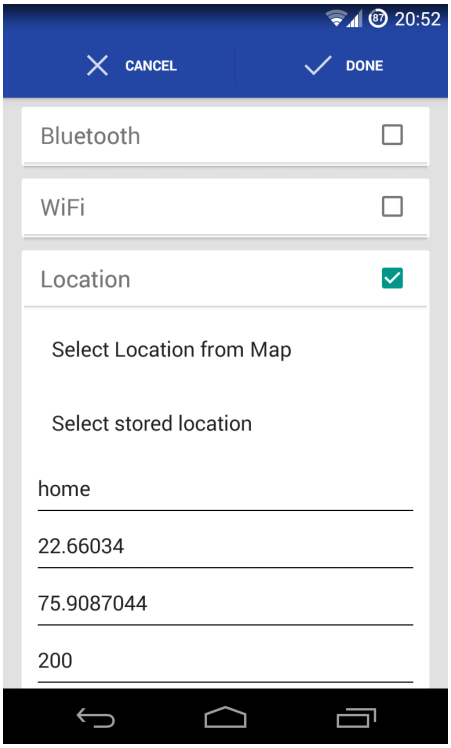
(a) Navigation drawer for restricted profiles



(b) Allowed Apps Fragment for restricted profiles



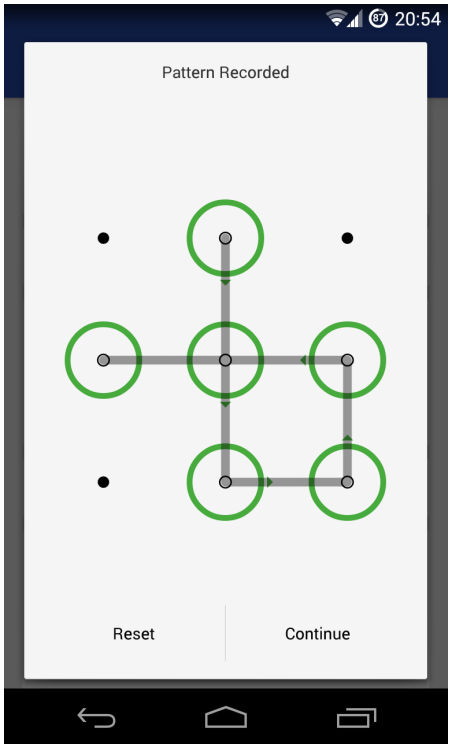
(c) Add Environment Activity



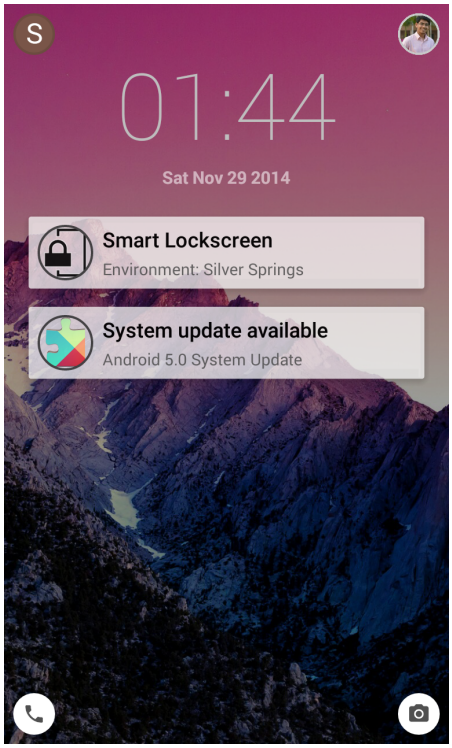
(d) Edit Environment Activity

Figure 4.2: Screenshots of the app - II

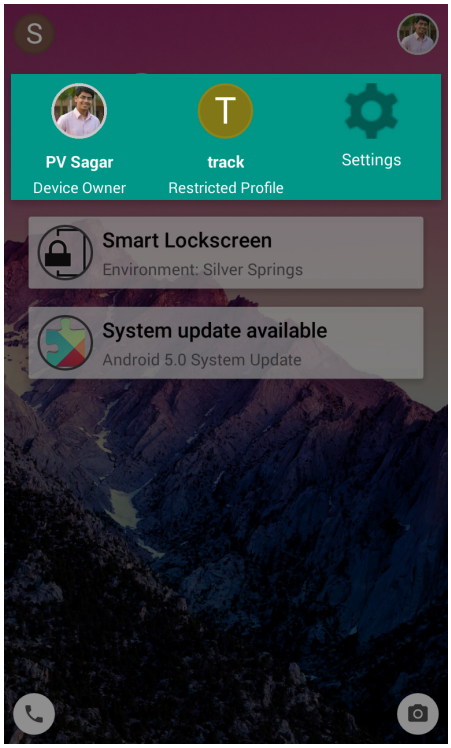
(a) Select Location Activity



(b) Set pattern lock Activity



(c) Lock-screen overlay



(d) Switching users

Figure 4.3: Screenshots of the app - II

pictures is going to be shown in the lock-screen overlay. Ability to customize pictures is an ongoing work, and is near complete.

- Long pressing on an environment triggers multi-select mode, in which a menu item to delete selected environments appear.
- Lastly, an additional 'Unknown Environment' item is also present in the list, which allows the user to change the default behaviour of using the master password when in an unknown environment.

Associated:
com.pvsagar.smartlockscreen.fragments.ManageEnvironmentFragment
class. This satisfies use case 3.

2. **Environment overlaps fragment(Figure 4.1c):** The identified overlaps of added environments are shown here. An environment will be selected by an overlap resolution algorithm by default, and here, the user can override this selection manually. Overlap resolution is done using the following algorithm:

   In case of an overlap of multiple environments, the environment with the strongest pass-phrase among them is selected. In case 2 environments have pass-phrases with the same strength, the environment which was added first is selected. Pass-phrase strength can be ascertained using the following algorithm:

   Types of pass-phrases can be arranged according to decreasing order of their strength, as Password >Pin >Pattern >None. That is, a password is always stronger than a pin, pattern or no pass-phrase, and a pin is always stronger than a pattern or no pass-phrase etc. Strengths of 2 passwords can be ascertained using their lengths and number of numerals and special characters in use. For comparing pins or patterns, their simple length is used.

   Associated:
   com.pvsagar.smartlockscreen.fragments.OverlappingEnvironmentsFragment
   class. This satisfies use case 4.

3. **Set Master Pass-phrase fragment(Figure 4.1d):** Here the user can specify the master pass-phrase. Only pin and password are allowed here since the master pass-phrase should be secure. This screen is shown when the app is started for the first time, and it is mandatory to set master pass-phrase before using any other features of the app.

   Associated:
   com.pvsagar.smartlockscreen.fragments.SetMasterPasswordFragment
   class. This satisfies part of use case 1.

If the profile under management is a restricted profile, then 2 options are available in the navigation drawer(Figure 4.2a). The corresponding fragments are detailed below. These satisfy use case 8.

1. **Manage Environments fragment(Figure 4.2b):** Here, the accessible apps for the profile under consideration can be set. A list of all apps installed along with their icons are shown, with switches for turning them on/off individually.

   Associated:
   com.pvsagar.smartlockscreen.fragments.AllowedAppsFragment class.

2. **Set pass-phrase fragment:** This is similar to item 3, but instead of setting the master pass-phrase, it sets pass-phrase for the profile under consideration. Whenever a user wants to access the device using this profile, the pass-phrase set here has to be used. SmartLockScreen Settings is always locked for restricted profiles.

   Associated:
   com.pvsagar.smartlockscreen.fragments.SetPassword Fragment class.

Besides the main options, static secondary options are available below the main options, which provide access to miscellaneous settings, help, and about pages.

Add/Edit Environment Activities(Figure 4.2c) User interface for specifying environment details.

- The name, hint and the Wi-Fi network, Bluetooth devices, and Location corresponding to the environment can be specified.
- Only the enabled options are shown, and rest are hidden to provide a clean interface (see Figure 4.2d where Wi-Fi is enabled for an environment and options related to that are visible).
- Location coordinates can be entered manually or selected from a map(see Figure 4.3a). Each location is also assigned a name, so that it could be reused in other environments. A radius should also be specified with the coordinates, so that an area can be identified and used for geofencing.
- Pass-phrase to be used for that particular environment can also be selected in this screen. 4 pass-phrase options are available, namely password, pin, pattern(see Figure 4.3b), and none. Pass-phrases are encrypted before inserting them into the database (Using com.pvsagar.smartlockscreen.backend_helpers.EncrypterDecrypter class).
- Once the user clicks 'Done' button, changes are effective immediately.

The classes AddEnvironment and EditEnvironment in com.pvsagar.-smartlockscreen package are used here. This satisfies use case 2.

Set pass-phrase for Unknown environment User can specify a different pass-phrase for unknown environment if master password should be used only for administrative purposes. The activity defined by com.pvsagar.smartlockscreen.-SetUnknownEnvironmentPassword is activated whenever user click unknown environment item in the list view in ManageEnvironmentFragment. Here they can either set a new pass-phrase for unknown environments or reset it so that master pass-phrase will be used.

4.1.2 Smart Lock

This satisfies use case 6. Implementation of Smart Lock, that is, changing the pass-phrase according to the current environment is also complete. This is done by maintaining the following data at all times:

1. Currently connected Wi-Fi network
2. List of connected Bluetooth Devices
3. List of current geo-fences

When the app is started, it populates these data elements, and subsequently, listens for any change in any of these variables. In package com.pvsagar.smartlockscreen.-receivers, BluetoothReceiver class receives broadcasts during changes in bluetooth device connections, and WifiReceiver class receives Wi-Fi state change broadcasts. Geofence transitions are received by GeoFenceIntentService in package com.pvsagar.-smartlockscreen.services.

BaseService is the background service running all the time which acts a central point of control(com.pvsagar.smartlockscreen.services.BaseService class). Whenever a change is encountered, the BaseService is notified, and it runs the current environment detection (handled by com.pvsagar.smartlockscreen.applogic.-EnvironmentDetector class). In case multiple environments match the current value of the environment variables, EnvironmentOverlapResolver class is used to modify the list such that the preferred environment is at the top of the list(If the user has set a preference, that preference is used, else the algorithm defined in Managing Environments and Users is used).

This environment detection runs in a separate thread, and to ensure that multiple threads of environment detection does not run at once, a semaphore is used.

This list is returned to BaseService, which stores it for the purposes of displaying it on the lock-screen overlay. It also sets the current pass-phrase according to the top most

environment in this list. In case the list is empty(i.e. the environment is unknown), then pass-phrase pertaining to unknown environment is set.

4.1.3 User profile switching

From the lock-screen overlay, user profile switching can be done(Figure 4.3d). When the current profile is changed, the reference pass-phrase of the device is changed in background. After unlocking the phone using this pass-phrase, the user will be able to access only the white-listed apps. Clicking on any other app icon, or trying to get into any other app in any way, will block the app and show a window in front, asking the user to enter the master pass-phrase if he wishes to access the app. Switching back to device owner profile can be done from lock-screen overlay, and once he unlocks the phone with current environment pass-phrase, unrestricted access to the device will be allowed again.

Monitoring the currently active app is done by com.pvsagar.smartlockscreen.services.AppLockService class. Allowing/denying access to the app based on current conditions is done by com.pvsagar.smartlockscreen.applogic.AppLocker class.

4.1.4 Lock screen overlay

A lock-screen overlay is shown over the lock-screen(Figure 4.3c). This is accomplished by monitoring screen off and screen on actions (By com.pvsagar.smartlockscreen.receivers.ScreenReceiver class). Whenever the screen is switched off, BaseService is notified, and it starts the overlay. It contains time, current environment, current user, and option to switch user, and current notifications. A mechanism for removing the overlay is also available. Swiping up/down will dismiss it, left will open camera, and right will open dialer. It shows pleasing animations while doing all this. Custom wallpaper is also supported. com.pvsagar.smartlockscreen.services.window_helpers.LockScreenOverlayHelper class deals with this overlay.

When pattern lock is in use, dismissing the lock-screen overlay will show pattern lock overlay (com.pvsagar.smartlockscreen.services.window_helpers.PatternLockOverlay class). User should enter the correct pattern to continue. Implementation details are described in subsection 4.3.1. This satisfies use case 5.

Start on Boot

The app now automatically starts on boot without explicit user interaction. This is achieved by com.pvsagar.smartlockscreen.receivers.StartOnBootReceiver class, which

receives a broadcast message after the phone has booted, and starts the services accordingly.

The app also reinitializes itself after it has been updated (i.e. the package was reinstalled). This is facilitated by com.pvsagar.smartlockscreen.receivers.-OnUpgradeBroadcastReceiver class. This is important because the service should be restarted after the app is updated, for example, via Google Play Store.

4.2 Brief explanation of important classes and packages

The classes touched upon in the previous section (4.1) are not explained here. For more information, please refer Documentation(javadoc), in Appendix A.

com.pvsagar.smartlockscreen.applogic_objects package. Many classes related to representing environment, different pass-phrases, and environment variables are present in this package. These classes also include static helper functions for database and other related operations.

DismissKeyguardActivity This is used when pattern lock is in use. To know more, please refer subsection 4.3.1.

GeneralSettingsActivity in com.pvsagar.smartlockscreen package. This gives an interface where miscellaneous settings of the app can be adjusted, like settings related to lock-screen overlay.

AdminActions in com.pvsagar.smartlockscreen.receivers package. Deals with administrative actions in the app, like changing system passwords.

com.pvsagar.smartlockscreen.frontend_helpers package. Contains helper classes for UI, like classes to manage notifications, get automatic images for user/environment pictures (the automatic picture is the first letter of the name in a circular filled background) etc.

com.pvsagar.smartlockscreen.cards package. Classes related to 'cards' used in the UI. Card view is a building block of our UI, and helps our UI to achieve a more 'Material Design' look and feel. Two card libraries are in use, one is the support library provided by Google, and second is a third party card library with additional functions. See cardslib by Gabriele Mariotti in appendix for details on the third party library.

com.pvsagar.smartlockscreen.backend_helpers package. Contains helper classes for back-end, like encrypter and decrypter, helper class for shared preferences of the app, helper class for managing all wakelocks in the app etc.

All passwords are encrypted before being stored in the database. This is done using EncryptorDecryptor class in this package[17].

4.3 Challenges overcome so far

We faced some significant challenged during the development of this app, and we've been able to overcome most of them. The important ones are mentioned below.

4.3.1 Pattern Lock challenge

We cannot set the system pass-phrase as a pattern, as we can do for pin or password, using Admin privileges. Therefore, only pin and password options were available initially. Now we have devised a way for users to have the convenience of pattern, without compromising on the security aspect. To achieve this, a custom pattern view was developed (Figure 4.3b), which mimics the Android pattern screen. This view is used while setting pattern lock as well while unlocking via pattern.

When the current environment has its pass-phrase set as a pattern, it sets the password as a string derived from the pattern, in the background. When the user wants to unlock the phone, pattern entry is shown, and if user enters the correct pattern, the password is changed to blank in the background. This presents another problem. When password is changed to blank in the background while the screen is on, it needs to turn off the screen and turn it on again, for the change to completely take effect. To overcome this, we start a blank activity which has certain flags set, which tells the activity manager to draw the activity on top of lock-screen, and dismiss it in background if the lock is insecure. Since the password is set as blank, it detects that lock-screen is insecure and dismiss it. As soon as this happens, the blank activity is closed automatically, along with the overlay showing pattern lock view. After the phone is unlocked, password is set in the background again.

Thus all that the user sees when lock-screen overlay is dismissed, is a pattern entry screen, and when they enter the correct pattern, the phone is unlocked. Setting the background password is necessary for security reasons, else anyone will be able to overcome pattern screen by simply rebooting the phone. If a password is not set, upon reboot, insecure lock-screen will be shown (before our app loads). But in our implementation, this is prevented by setting a system password in the background.

4.3.2 Environment overlap resolution

We have designed a suitable algorithm to resolve environment overlap, i.e. select a preferred environment when multiple environments are detected at the same time. The

algorithm is explained in subsection 4.1.2. We choose the strongest of the available pass-phrases to ensure security at all times. Once the user has authenticated himself, he'll have the ability to set a preference for that overlap, making it convenient during further overlaps of the same set of environments.

4.3.3 Minimize battery usage

We've designed the app in such a way that all actions happen according to events that occur in the system, that is, there are no infinite loops or constant monitoring that happens in the background. The app receives broadcasts when any relevant event happens, and take suitable actions for that. Thus, resources are used only when necessary, thus minimizing the battery footprint of the app.

5. Testing and Analysis

5.1 Unit testing

Back-end(especially database and related classes) were tested extensively using the Android test framework. Variety of unit test cases were written and and used for testing each stage of development.

Apart from that, we have been using the app in our phones continuously, in different conditions and configurations of the app, monitoring any abnormal/unexpected behaviour, and taking appropriate remedial bug fixes.

5.2 Alpha release and testing

We put up our app as an alpha release, available for download, at XDA Forums, on Oct 27, 2014. App description, instructions to use, screen-shots etc. were also put up along with download links. We received positive feedback, and constructive suggestions and feature requests for the app from the members of the community within a few days.

Three more updates were published since, in which many of the community suggested features were implemented. Each version got over 1000 downloads within the first week of putting them up. The app was even featured on a website wonderhowto.com, complete with a video tutorial.

The links to the above mentioned XDA thread can be found in Appendix A.

5.3 Analysis of possibility to increase convenience

The below given data table (Table 5.1) contains the time taken to unlock different phones with different unlock methods (different pass-phrases). The data was collected from different phones by recording the screen and evaluating the time taken via the recorded video.

We will now consider different user scenarios to understand how our app reduces the time taken to access the smartphone without compromising security. In every scenario we consider the user unlocks the phone on an average of 110 times/day[1]. In all these

Table 5.1: Unlock times for different pass-phrases

Pass-phrase/Phone	Nexus 5	Nexus 4	Xperia ZL	Moto G	Note 2	Average
None	1.375	1.444	1.512	1.499	1.745	1.515
Pattern Easy	2.069	2.520	2.203	2.420	2.492	2.341
Pattern Difficult	3.389	3.506	3.372	3.332	3.411	3.402
Pin	2.924	2.571	2.991	3.422	3.210	3.024
Password Easy	3.244	3.141	3.343	3.520	3.314	3.312
Password Difficult	5.668	4.775	5.744	5.822	5.037	5.409

scenarios the default security setting without our app will be considered as the highest security setting within the scenario (since security is given priority).

Scenario 1: The user, an employee in a company, sets an easy pattern in the office and no password at home and difficult password everywhere else. This user on an average day spends 30% time of overall smartphone usage in the office and 50% at home and 20% everywhere else. For this scenario, the total time taken with our app will be 280 seconds. Time taken without our app will be 594 seconds.

Scenario 2: The user, a student, sets an easy pattern for hostel since its a shared environment, no password for college as the user will always have the phone with him/her and pin for everywhere else. A typical college student spends 25 time of overall smartphone usage in the college and 55% at home and 20% everywhere else. For this scenario, the total time taken with our app will be 250 seconds. Time taken without our app will be 332.64 seconds.

Scenario 3: The user, a security concerned user, wants a difficult password to unlock whenever the phone is not in his hand. He sets no password when the phone can find his bluetooth based smart-watch, else difficult password. This user on an average day wears his watch 80% of the time. For this scenario, the total time taken with our app will be 252 seconds. Time taken without our app will be 594 seconds.

These scenarios were chosen with carefully considering the different lifestyles of people who are potential users of our app. In the case of phone accessibility, even 500 milliseconds makes a huge different. In the above mentioned scenarios we can see significant decrease in the accessibility time. Hence with the above results we can see that our app provides a possibilty to access the phone faster without compromising the security, leading to faster access to the phone when required, and lesser user frustration.

Additionally, providing notifications in lock-screen reduces the number of times the user needs to unlock the phone, and even more time is saved.

6. Conclusion

6.1 Challenges

Although we solved some of the challenges that we faced earlier, some are yet to be solved, and new ones have come up in the mean time.

1. **Android Lollipop Compatibility:**
   Android Lollipop (version 5.0) has been officially announced. Devices with Android Lollipop has started shipping from late November 2014, and a few devices are also getting updates at the same time. Android 5.0 is a big leap forward, and there are many changes throughout the system, as well as the APIs available. While we have maintained a compatible look and feel of the app in accordance with the new design guidelines, several APIs, central to our app, have either changed their behaviour, or have been depricated. Some of the difficulties with Lollipop are mentioned below.

   - Setting blank password via device admin does not work properly. After setting password/pin programmatically, user is still shown the password input box, and clicking done button with blank entry will unlock the phone. This unnecessary step in between is inconvenient and against our main aim, which is maintaining convenience along with security. Since our custom pattern lock is dependent upon setting a blank password(as described in subsection 4.3.1), pattern lock also becomes inconvenient while using the app in Android 5.0.
   - Some APIs related to restricting access to certain apps, have been depricated in Android 5.0, since they pose a security problem to the enhanced multi-tasking features in Lollipop. On the other hand, Lollipop introduces native support for restricted user profiles, and hence we're planning to make the restricted profiles option available only to devices running Android KitKat(version 4.4) and below.

2. **Backward Compatibility:** Maintaining backward compatibility is always a challenge. As of now, we are supporting JellyBean 4.1 and above, but notification

feature works only in 4.3 and up.

3. **Understanding the user:** The environment as recognized by the user, and as recognized by the app should be the same. Although this aspect has been improved throughout the development process, there is always a scope for making it better. Another challenging aspect is to develop a non-intrusive way through which the current user of the device always knows the current user profile in use.

6.2 Future work

6.2.1 Extending the scope of Environments

A framework for adding sensors as needed by the user, will be great addition to the app. Many sensors have been introduced to smart phones recently, and many more could come in the future. Android's built in sensor framework[18] can be used as a base to build this. A tool can be implemented, for generating source according to user defined sensors, and environment definition and detection will be based on those sensors once the code is compiled and installed on any device.

6.2.2 Automatic screen on and off

Switching on or off the screen automatically according to sensor data from Proximity sensor and Accelerometer, is a feature which we wish to implement after we perfect the remaining parts of the app.

6.2.3 Miscellaneous features

1. Prevent unlock and give options for master pass-phrase unlock after a fixed number of invalid pass-phrase entry attempts.

2. Option to auto detect current values of all variables in Environment Addition screen.

3. More general settings, which includes a global on-off switch, options to show/hide pattern while entering it, etc needs to be done.

4. Customizing user and environment pictures is an ongoing work(back-end, and some of the front end is complete).

References

[1] Woollaston, Victoria. How often do you check your phone? The average person does it 110 times a DAY (and up to every 6 seconds in the evening). *Daily Main UK*. October 8, 2013.

`http://www.dailymail.co.uk/sciencetech/article-2449632/How-check-phone-The-average-person-does-110-times-DAY-6-seconds-evening.html` (Accessed Nov 29, 2014).

[2] Uellenbeck, Sebastian, Dürmuth, Markus, Wolf, Christopher and Holz, Thorsten. *Quantifying the Security of Graphical Passwords: The Case of Android Unlock Patterns*. Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany, 2013

[3] Hussain, Faisal. 2014. Face Unlock May Be Faster on Lollipop, But It's Still Not Secure. *Nexus 5 WonderHowTo blog*. November 15.

`http://nexus5.wonderhowto.com/inspiration/face-unlock-may-be-faster-lollipop-but-its-still-not-secure-0158568/` (Accessed Nov 28, 2014).

[4] O'Boyle, Britta. How does the Samsung Galaxy S5 fingerprint scanner work? *Pocket-lint*. April 11, 2014.

`http://www.pocket-lint.com/news/127605-how-does-the-samsung-galaxy-s5-fingerprint-scanner-work` (Accessed Nov 28, 2014).

[5] Verduzco, Will. 2014. Dynamic Pin Locks Your Screen and Apps with Intelligence. *XDA Developers portal*. August 5.

`http://www.xda-developers.com/android/dynamic-pin-screen-app-lock/` (Accessed Nov 28, 2014).

[6] Saleem, Hammad. 5 Best Free Tools To Lock Or Password-Protect Apps On Android. *Addictive-tips.com*. November 30, 2013.

http://www.addictivetips.com/android/best-free-android-tools-to-lock-password-protect-apps/ (Accessed Nov 28, 2014).

[7] Mohta, Ashish. Get Unlock with Android Wear Feature to Work on Any Android Phone. *Wearables Arena*. August 7, 2014.

http://wearablesarena.com/unlock-android-wear-app-review/ (Accessed Nov 28, 2014).

[8] Warren, Christina. Tap Your iPhone to Unlock Your Mac With Knock. *Mashable*. November 06, 2013.

http://mashable.com/2013/11/05/knock-unlock-app/ (Accessed Nov 28, 2014).

[9] Nickinson, Phil. Trusted Bluetooth devices: A must-have for every smartphone going forward. *Android Central*. September 11, 2013.

http://www.androidcentral.com/trusted-bluetooth-devices-must-have-every-smartphone-going-forward (Accessed Nov 28, 2014).

[10] Android Developers. Notifications.

http://developer.android.com/guide/topics/ui/notifiers/notifications.html (Accessed Nov 28, 2014).

[11] Understanding notifications on iPhone, iPad, and iPod touch.

http://support.apple.com/en-in/HT201925 (Accessed Nov 28, 2014).

[12] Android Developers. Android Anatomy and Physiology.

http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf (Accessed Oct 27, 2014).

[13] Yarow, Jay. This Chart Shows Google's Incredible Domination Of The World's Computing Platforms. *Business Insider*. March 28, 2014.

http://www.businessinsider.com/androids-share-of-the-computing-market-2014-3 (Accessed Oct 27, 2014).

[14] Android Developers. Building and running.

http://developer.android.com/tools/building/index.html (Accessed Oct 28, 2014).

[15] Android Developers. Building and running from the command line.

http://developer.android.com/tools/building/index.html (Accessed Oct 28, 2014).

[16] Android Developers. Android Studio.

https://developer.android.com/sdk/installing/studio.html (Accessed Oct 28, 2014).

[17] Wikipedia. Advanced Encryption Standard

http://en.wikipedia.org/wiki/Advanced_Encryption_Standard (Accessed Nov 29, 2014).

[18] Android Developers. Sensors Overview.

http://developer.android.com/guide/topics/sensors/sensors_overview.html (Accessed Nov 28, 2014).

Appendices

A. Source, Documentation and Release

A.1 Source Code

The project uses Git version control, and is hosted at `http://github.com/aravindsagar/SmartLockScreen`. The repository is private, and hence cannot be accessed publicly, for now. We are planning to make it public after the app reaches beta stage, along with adding a suitable open source license. If you want access to the source, please contact us.

Pattern Lock library used in the app was developed in house. Sources can be found online with demo application, at `https://github.com/aravindsagar/Lock_Pattern_Grid_View`

A.2 Documentation

Documentation can be accessed at `http://aravindsagar.github.io/SmartLockScreen/Documentation/index.html`. It is publicly available and will be updated when the development continues.

A.3 Alpha version release

To gain a wider testing audience, the app has been put up at XDA Forums, as an alpha release. Download links, instructions, and more can be found at `http://forum.xda-developers.com/android/apps-games/app-smartlockscreen-android-enjoy-t2919989/`

B. Third party libraries in use

Apart from support libraries provided by Google, there are a few third party libraries used in our project, mainly related to the UI.

B.1 cardslib by Gabriele Mariotti

Copyright 2013-2014 Gabriele Mariotti. Licensed under the Apache License, Version 2.0.

`https://github.com/gabrielemariotti/cardslib`

B.2 Android System Bar Tint, by Jeff Gilfelt

Copyright 2013 readyState Software Limited. Licensed under the Apache License, Version 2.0.

`https://github.com/jgilfelt/SystemBarTint`

B.3 Android CropImage by Lorenzo Villani

Licensed under the Apache License, Version 2.0.

`https://github.com/lvillani/android-cropimage`

Glossary

environment overlap happens when the environment detection algorithm determines that 2 or more of the defined environments match the current conditions of the environment variables. 37

geofence is a location coordinate (a pair of latitude and longitude), along with a radius. Thus it defines a circular area on the earth. 38, 39

lock-screen overlay is a window shown on top of the standard Android lock-screen, which displays additional information like current environment, current user profile, notifications etc. 39, 40

Material Design is a comprehensive guide for visual, motion, and interaction design across platforms and devices, specified by Google. 41

pass-phrase is a general name given to different kinds of methods available to unlock the phone. 38, 59

password is a kind of pass-phrase. It refers to a string which can contain alphabets, numbers and special characters. 38

pattern is a kind of pass-phrase. A pattern is entered in a grid of 9 dots (3x3 grid), where at least 4 dots are connected in some particular order. See Figure 4.3b. 38, 40, 42

pin is a kind of pass-phrase. It refers to a string of numbers. 38