GitHub Username: joysoi (Nikola Nikolovski)

AppProtect

Description

App Protect's main feature is to “lock” user selected apps with a fake ("Application not responding") pop up.

Intended User

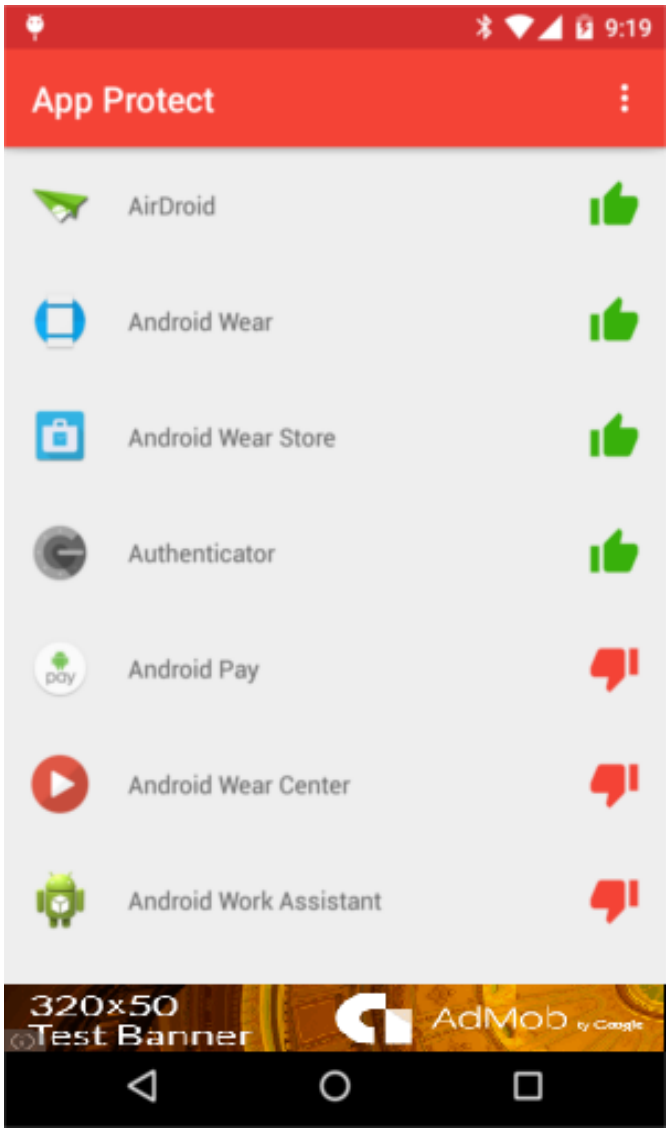
Intended for users of all ages that want to protect sensitive data.

Features

- Restricts access to user selected apps
- Use in-app advertising banner to show ads
- Intuitive and friendly UI

User Interface

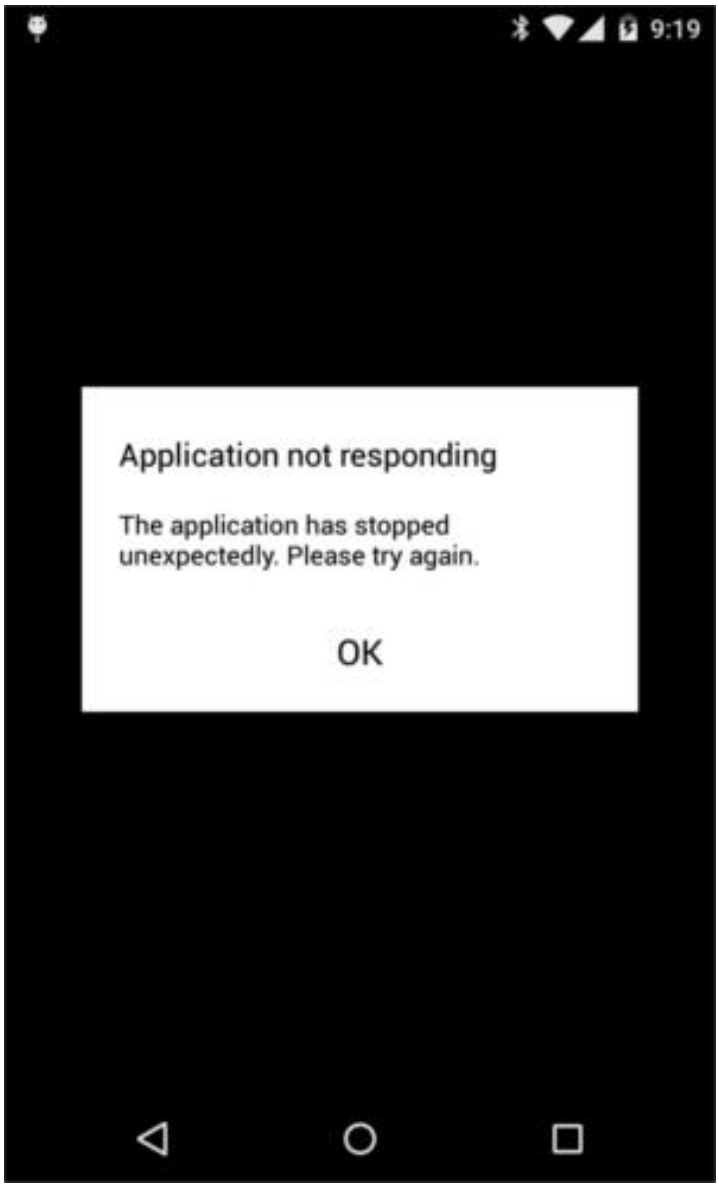
Screen 1



All user installed apps and some system apps are presented in a list that is constantly updated according their installed status.

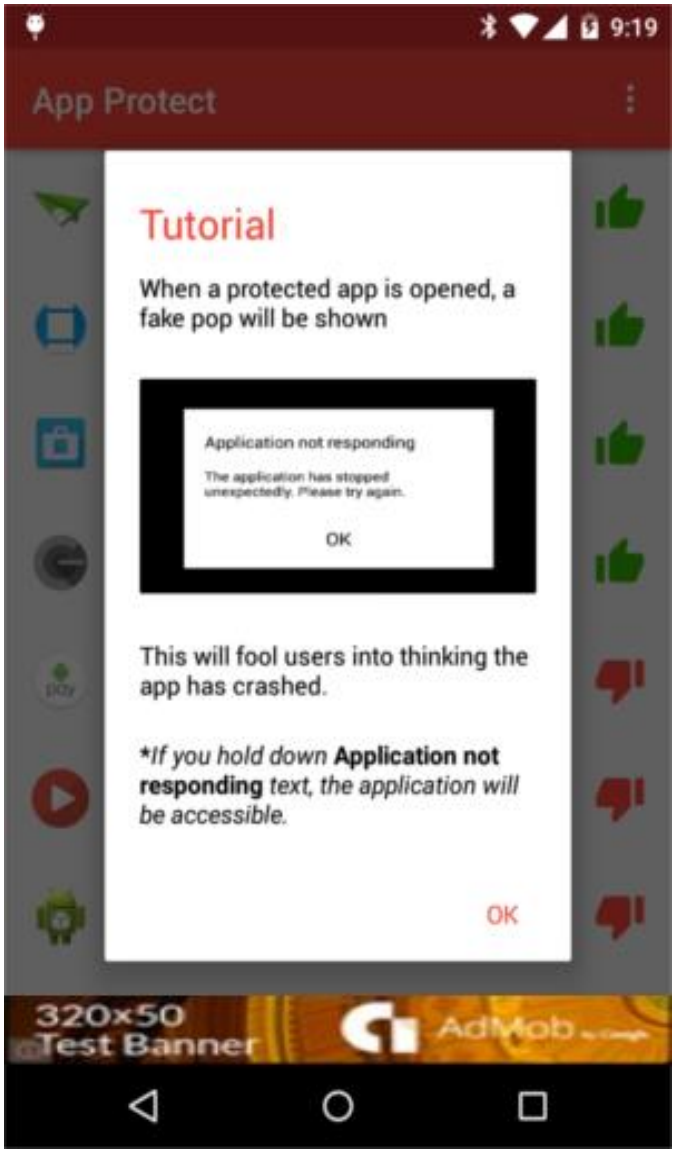
- “Green thumb up” is signifying that the app is locked.
- “Red thumb down” is signifying that the app is unlocked (default).

Screen 2



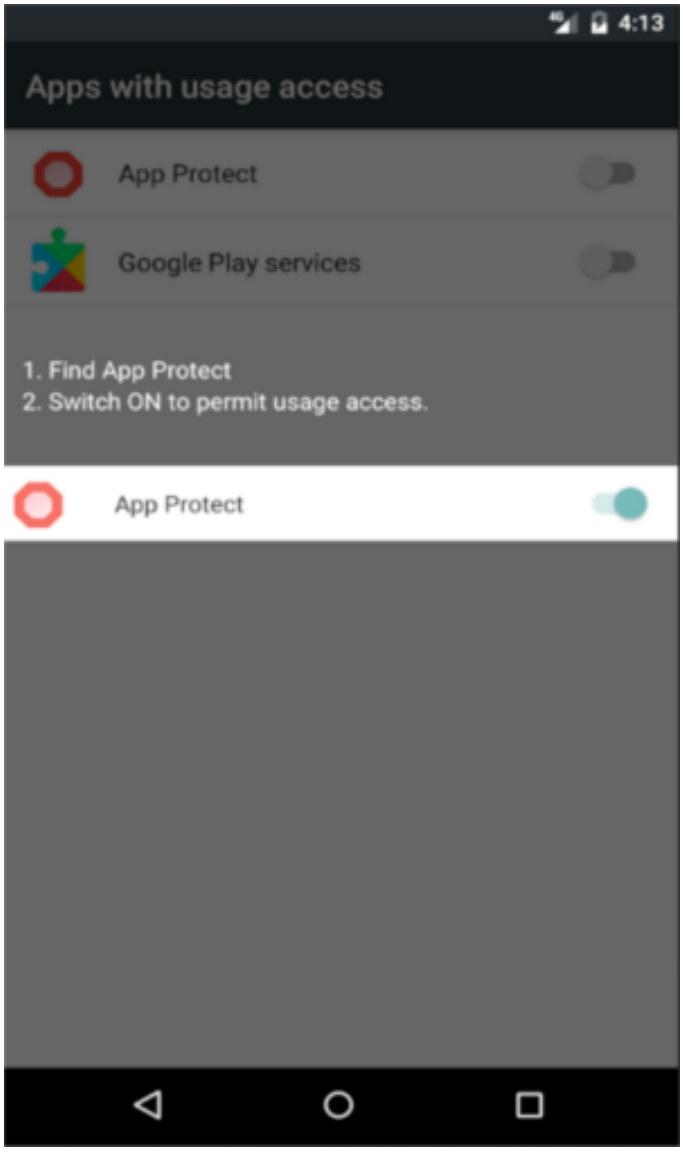
Pop up "Application not responding" screen is appearing whenever a “protected” / “locked” app is launched.

Screen 3



In the overflow menu button, there is a brief user tutorial to explain the functionality of the app.

Screen 4



Whenever the user wants to lock an app, on devices above version 21 usage access permission is required.

App Widget Icon

On app widget click all locked apps will be unlocked.

Key Considerations

How will your app handle data persistence?

I used Cupboard library to manage persistence in a SQLite instance on the app. It is a small library that makes managing client-side models extremely easy in simple cases.
Cupboard works like an Object Relational Mapper(ORM) by mapping java classes to database tables and mapping java class member variables to the table columns. Through this process, each table maps to a Java model and the columns in the table represent the respective data fields. Similarly, each row in the database represents an object.

Describe any corner cases in the UX.

- If the user holds “Application not responding” screen for a few seconds, the “locked” app will be able to access the app.
- In Android 5.1 or above, the user must manually permit access for the Service component to loop and compare “locked” apps package name with the foreground app.
- Hiding the app from recently used apps to make illusion that the pop up is a system message.

Describe any libraries you'll be using and share your reasoning for including them.

Cupboard - create, modify, delete, and query the SQLite database using instantiated objects instead of writing SQL every time.

ButterKnife - annotate fields with @BindView and a view ID for Butter Knife to find and automatically cast the corresponding view in your layout.

Timber - write logs in different places and control how it's done in a centralized manner.

Glide - image loading and caching library for Android focused on smooth scrolling.

RecyclerView - flexible view for providing a limited window into a large data set.

Firebase Ads - integrate the Google Mobile Ads SDK and use it to display an AdMob banner ad.

Firebase Analytics - provides app usage info and user engagement.

Describe how you will implement Google Play Services.

I will implement:

- Mobile Ads SDK and use it do display AdMob Standard banner for phones and tablets.
- Firebase Analytics SDK collects usage and behavior data for your app.

Required Tasks

Task 1: Project Setup

App Protect functionality:

- When launched, presents the user with list of all current user apps and system apps.
- Allows user to press the button (red thumb down) next to every app in the list and "lock" any app that the user desires (green thump up).
- If "locked" app is launched, will appear pop up "Application not responding" screen and access will be restricted.
- If user press and hold the pop up screen for few seconds, the access will be granted.
- AdMob Standard Banner presented.
- App Widget on the Home screen available.

Task 2: Implement UI for Each Activity and Fragment

Subtasks:

- Build UI for MainActivity containing Fragment.
- Build UI for Fake_PopUp
- Build UI for FakePopUp_Tutorial
- Build UI for Dialog_Access_Tutorial.
- Build UI for App Widget

Task 3: Fundamental Setup

- Declaring permissions in Manifest.
- Gradle setup.
- Setting up resource values.
- AsyncTask will be implemented on short background tasks

Task 4: Creating Activity and Fragment class:

- Defining Activity Layout
- Initializing Fragment.
- Defining Fragment Layout
- Establishing communication between Activity and Fragment

Task 5: Creating Tutorial Dialog:

- Creating the Dialog and its Layout.
- Handling menu item click

Task 6: Creating Custom Adapter

- Creating class that extents RecycleViewAdapater.
- Creating ViewHolder Layout.
- Implementing Glide Library
- Implementing Vector Drawable
- Handle Click Events.
- Implementing search bar.

Task 7: Defining communication between Data Layer and View Layer:

- Adapter setup
- Implementing Callbacks.

Task 8: Creating DB:

- Implementing Cupboard library.
- Creating DBHelper.
- Creating DataProvider and Loaders
- Creating Model/Entity.

Task 9: Handling DB operations:

- Defining methods with create, modify, delete and query responsibility respectively.
- Implementing Stetho.

Task 10: Receivers:

- Declaring Broadcast Receiver
- Registering for actions.
- Handling DB operations accordingly.

Task 11: Lock Logic:

- Declaring Service
- Maintain Service state beyond application lifecycle.
- Displaying Views with Window Layout Manager according logic.

Task 12: Implementing Ads and Analytics Services:

- Implementing Google Play Services.
- Creating Layout
- Initializing

Task 13: Creating App Widget

- Adding the widget in Manifest
- Creating widget Layout, Class, Metadata and initializing it.

Task 14: Testing

- Testing all components.