

INDEX

S. No.	Topic	Pg. No.	Remarks
1	Introduction	1	
2	Existing System	2	
3	Drawbacks in the Existing System	2	
4	Proposed System	3	
5	Objectives	3	
6	Modules & Description	4	
7	Hardware & Software Requirements	25	
8	Weekly Contribution	27	
9	SDLC Model Used	30	
10	Declaration	32	
11	Certificate	33	

Introduction

In this contemporary world, many new software and applications have arrived and they have given all the comforts that the programmer or user has demanded. But still, there are some demands which still have not been solved. So, we are here to present the most useful feature that is **Cafii**. In everyday life, we are using one or more electronic gadgets and after the covid-19 pandemic, it has become mandatory to perform most of the tasks through the digital method that is the internet. If you are a user who is constantly using the gadget screen for a long time but sometimes then **Cafii** is the solution.

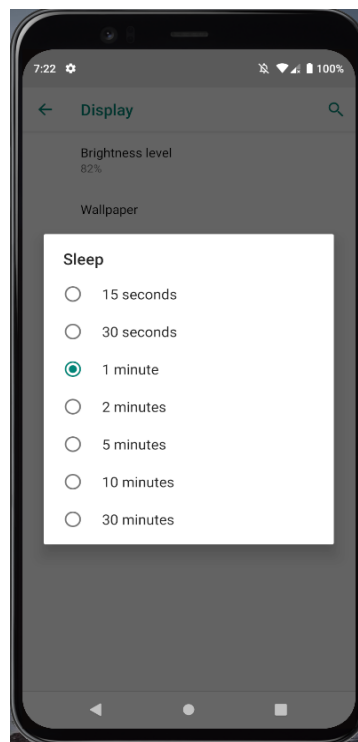
The user can't change the screen on time repeatedly in the device-specific settings. **Cafii** is a better solution that increases the screen on time with a single click or tap within the app. It is an android app for the implementation of a temporary screen on time. It is written in Java language. It prevents the screen from turning off. We have provided 5 types of presets that users mostly use. Therefore, if the user is taking the control of the device screen using **Cafii** then he/she does not need to interfere with the default device screen settings.

Existing System

Numerous platforms are existing in the present like Windows, Android, Linux, etc. There are also numerous hardware platforms like desktops, laptops, smartphones, smartwatches, etc. So, our project is mostly dependent upon the operating system platform and least dependent on hardware specification because **Cafii** does not require high-end systems or hardware.

Nowadays due to pandemic, we are attending online classes and others are attending meetings of their office. So, as we are students, we need to make notes using the document provided by our faculty, so in that case, we cannot repeatedly touch our screen to make it light up continuously. Now the solution is we can do that is to easily go to our respected mobile setting and change to our preferred timeout, but that's not the solution, it's not a short process for changing and reverting it back. Our app does that work well in this situation, its main feature is that it reverts the setting back after the particular timer ends.

Drawbacks in the Existing System



In our smartphones, we have N numbers of setting by which we can make our mobile easy to use as per our preference but the main flaw arises when we need to revert the settings that we have changed it for temporary purpose because we can simply forget to revert it and that will lead to unnecessary battery drain and more consumption of CPU. For example, if we set our screen time out to 30 minutes then we can easily forget to change it back to our default timeout, due to which it might be possible that whenever we keep mobile aside the screen will be lit up to 30 minutes without getting noticed.

Proposed system

In our minor project, we had made a **Cafii** prototype which was a simple java program that use to run on IDE, but in our major project, we have taken it to another level which means we have implemented it as an application that will run in android. It will fulfill the requirement to set screen timeout for the particular time period. We have tried our best to make it as robust as possible. And will not produce any of the glitches or bugs.

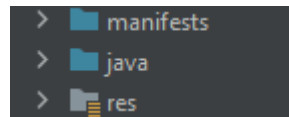
Objectives

- 1) To introduce software by which we can handle and operate the screen time out temporarily by a single tap on the profile buttons.
- 2) This will make it easy for the user to control the screen without any worry about changing device-specific settings repeatedly and will save the device battery from draining.
- 3) Our application is only made to run on android and supports most of the android operating systems.

- 4) For now, it's stable for android and in the future, we will develop it for Linux and Windows devices.
- 5) This will be a very helpful tool for those who love reading from the gadget's screen. We may also be bringing more features and optimizations to it.

Modules & Description

Android Project Structure

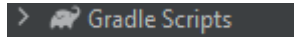


Manifests - Every project in Android includes a manifest file, which is `AndroidManifest.xml`, stored in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements.

This file includes nodes for each of the Activities, Services, Content Providers, and Broadcast Receiver that make the application and using Intent Filters and Permissions determines how they co-ordinate with each other and other applications. The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, unit tests, and define hardware, screen, or platform requirements.

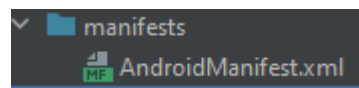
Java - Android applications are developed using the Java language. As of now, that's really your only option for native applications. Android relies heavily on these Java fundamentals. The Android SDK includes many standard Java libraries (data structure libraries, math libraries, graphics libraries, networking libraries and everything else you could want) as well as special Android libraries that will help you develop awesome Android applications. This folder consists of packages as well as the Android activities, services, EventBus based on Java and other Java classes as well.

Res - The res folder holds resources for the project, which includes things like strings, drawables, and layouts. The benefit of having these resources separate from your Java code is so that they can be updated independently.

A dark rectangular button with a right-pointing chevron and the text "Gradle Scripts".

Gradle Scripts - Gradle is a build system that takes the best features from other build systems and combines them into one. It is improved based on their shortcomings. It is a JVM-based build system. That means you can write your script in Java, which Android Studio makes use of. Gradle is a plugin-based system. Gradle Script files use a domain-specific language or DSL to define custom build logic and to interact with the Android-specific elements of the Android plugin for Gradle.

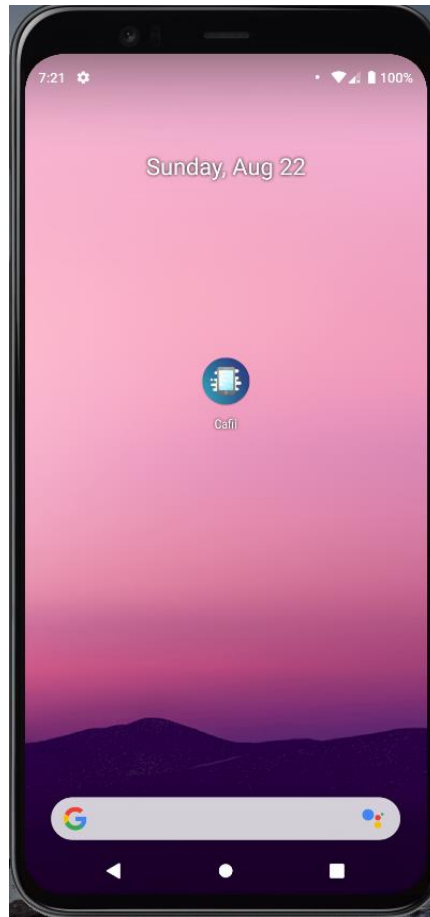
Manifests



AndroidManifest.xml - It only includes the AndroidManifest.xml. It stores essential information about the application to the Android system, information which the system must have before it can run any of the application's code.

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"
    tools:ignore="ProtectedPermissions" />
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"
    tools:ignore="QueryAllPackagesPermission" />
```

```
<activity android:name=".MainActivity"
    android:exported="true"
    android:excludeFromRecents="true"
    android:noHistory="true" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<service android:name=".CafiiService"/>
```

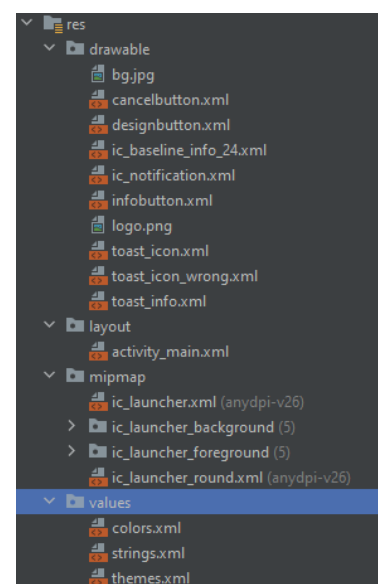


Res

Drawables - When you need to display static images in your app, you can use the Drawable class and its subclasses to draw shapes and images. A Drawable is a general abstraction for something that can be drawn.

Layout - A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. For our MainActivity layout and UI design, we are using **activity_main.xml**.

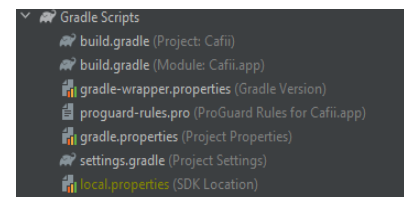
Mipmap – It's just like Drawable but with the advantage that using it as the source for your bitmap or



drawable is a simple way to provide a quality image and various image scales, which can be particularly useful if you expect your image to be scaled during an animation. **Values** - is used to store the values for the resources that are used in many Android projects to include features of colour, styles, dimensions, etc.

Gradle Scripts

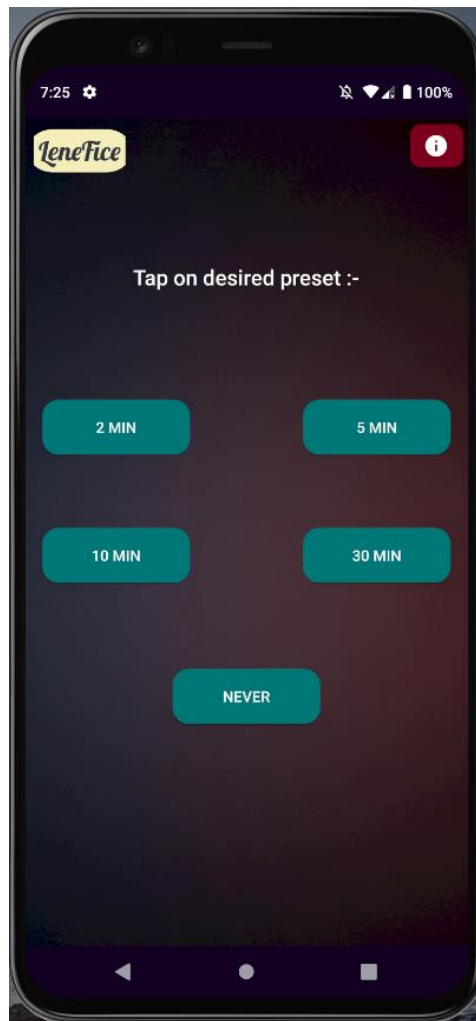
Build.gradle - Build.gradle are scripts where one can automate the tasks. These also include dependencies of internal and external APIs.



Other Files present are auto generated by Android Studio for Building Purpose.

Java

MainActivity.java - The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. MainActivity is the default activity that is provided by the Android Studio whenever we create a new project on it. This class extends AppCompatActivity class which is the base class for activities that wish to use some of the newer platform features on older Android devices. This class contains methods which define lifecycle of Activity for example onCreate(), onStop(), etc. We can override these methods according to our needs. The MainActivity mainly plays the role of interacting with the user and loading the UI that is being designed in the activity_main.xml.



Methods Used

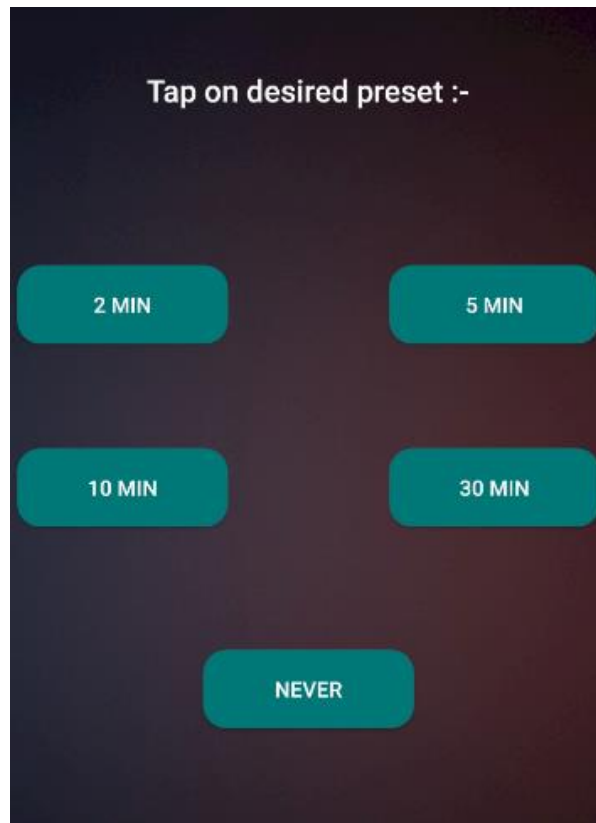
onCreate(Bundle savedInstanceState) – This method is executed as soon as the user taps on the application icon and the activity are created and shown to the user that means the application is launched. In our case it is MainActivity. Bundle savedInstanceState is used to save & recover state information for the activity. In instances like orientation changes or killing of your app or any other scenario that leads to calling of onCreate() again, this can be used to reload the previous state information.

We had override this method with our definition. We are calling the super onCreate() method. We are loading the user interface from activity_main.xml. setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT) method is used to lock app rotation to portrait. Other methods called are toMapComponents(), detectAndroidVersion(), askPermission(), detectDevice().

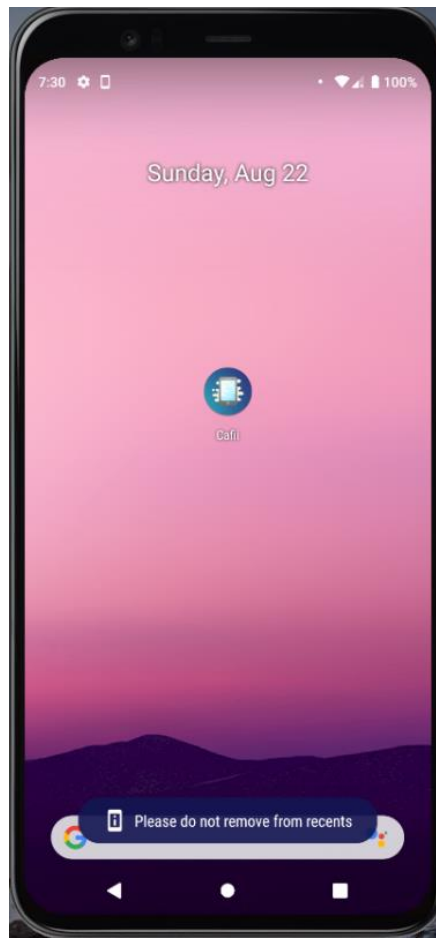
onStart() – This method is executed after the creation of activity or when we switch back to our activity. We have to override this method with our definition. We have called the super onStart() method. After that, we enable the buttons by calling the buttonsAreEnabled() method. After this, we have initialized on-click listener for all the buttons. Whenever this method is executed a condition check will be performed that will be checking either the service is already running or not, for checking the service status we are calling the method isMyServiceRunning() method. If the service is already running button status will be set by buttonsAreDisabled() and eventbus will be registered.

onClick(View view) – This method will be executed whenever a button is clicked. As soon as this starts to execute a switch case will determine which button has been tapped with the help of the button id cases. After the particular case matches the respective code inside the case gets executed.

If the preset buttons are tapped then the sendDataStartService() method is called by passing the time value that is 2 minutes, 5 minutes, etc. If the cancel button is clicked two methods will be called cancelTriggered() and buttonsAreEnabled(). If the info button is pressed then the showNoticeDialog() method is called.

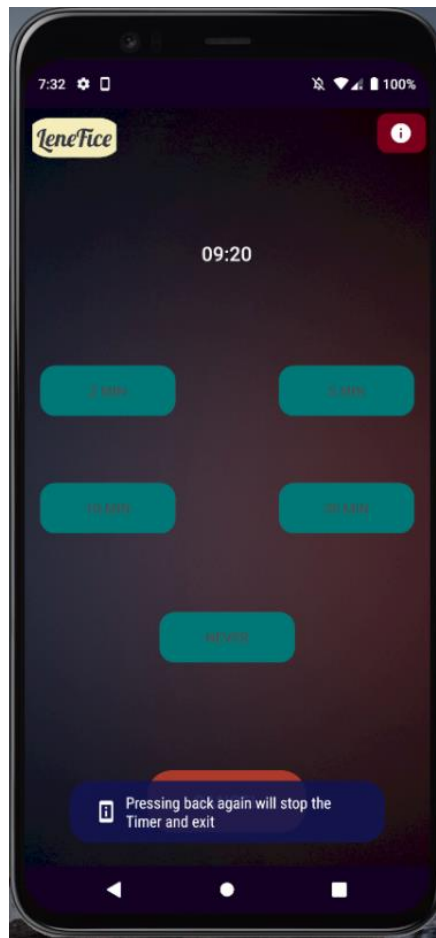


onStop() – This method is executed when the app activity is minimized or removed. Like onCreate() and onStart() we call super onStop() for this. Similarly, like them, we have overridden and defined this method. We are performing a condition check that will check whether the service is running or not with the help of the isMyServiceRunning() method. If the service is running it will check the android version and will display the Toast accordingly. It will also unregister the eventbus.



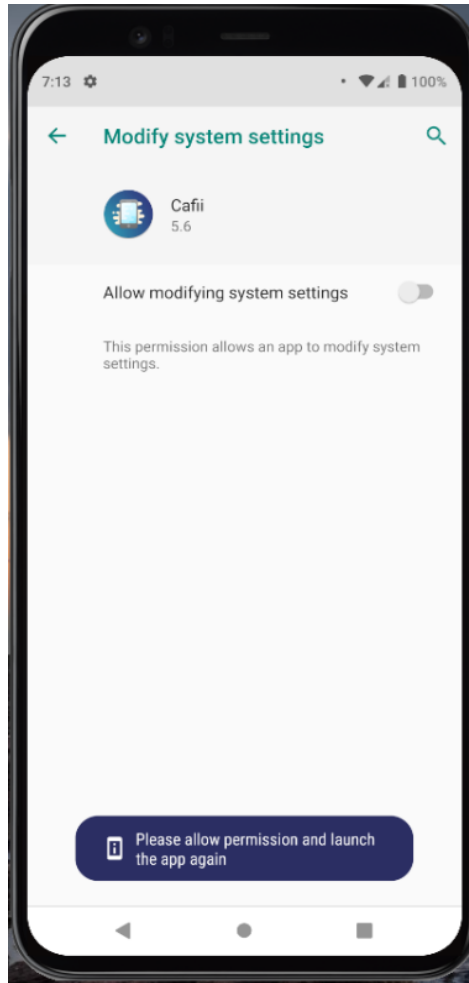
onBackPressed() – This method will be executed whenever the back button or gesture is triggered. This method is also overridden and defined by us. It consists of conditional checks for checking the status of service. If the service is running it will check if the back action is triggered twice in 2 seconds or not. If back action is triggered twice in 2 seconds then service will be stopped, `cancelTriggered()` method will be called and appropriate toast will be displayed according to the android version.

If back action is triggered only once in 2 seconds, then it will display the toast accordingly. In simple language, it means if back action is triggered twice in time of 2 seconds while service is running then the only service will get stopped and activity will be closed. If service is not running and back action is triggered then the activity will be simply closed without any toast.



toMapComponents() – This method will simply map the components objects with the component id present in the activity_main.xml. This method is only required to be called whenever onCreate() for the activity is executed.

askPermission() – This method will check the permission to modify system settings. If permission is not granted then it will launch settings app with the activity where user can enable or disable permission to modify system setting for that particular app. It will also force stop the app and toast will be displayed to restart the app after granting the permission. If permission will be already granted then the app will launch normally.

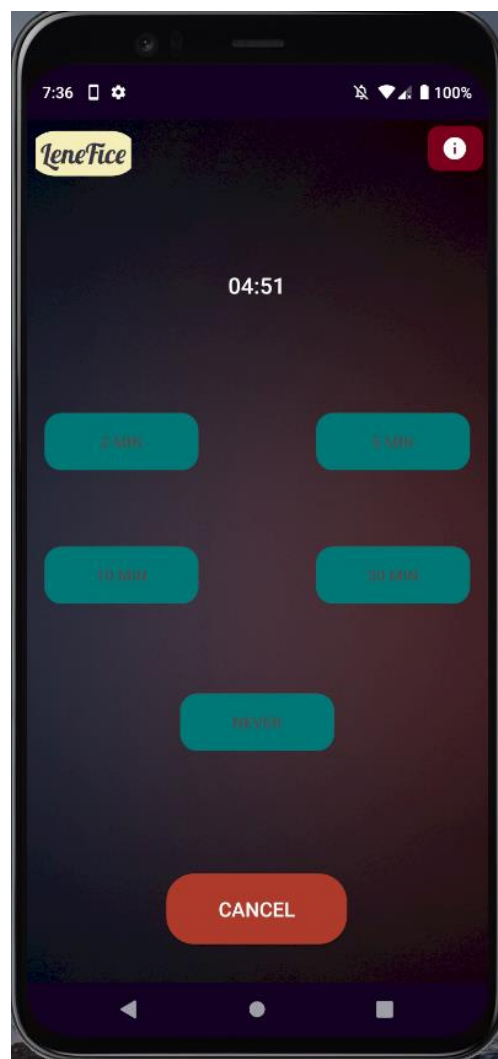


appInstalledOrNot(Context context, String uri) – This method will be taking the passed string when called in other methods and will be checking whether the particular package name exists or not in the android app packages list. If particular package name is found then this method returns true else it will return false.

isAppInSystemPartition(PackageManager pm, String packageName) - This method will be taking the passed string when called in other methods and will be checking whether the particular package is in the system partition or not. In simple language we mean to say whether the app is a system app or a third-party app. If the particular package name is in the system partition it will return true else it will return false.

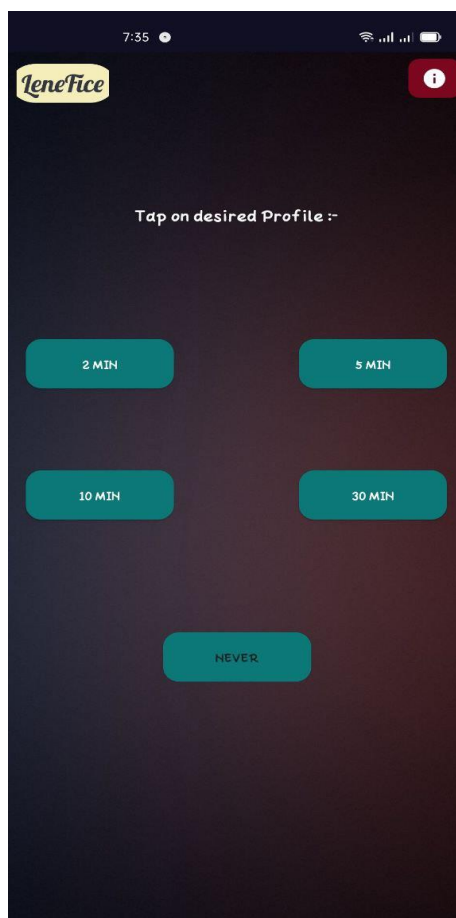
detectDevice() – This method will determine which android OS ROM it is e.g. Miui, OneUi, Stock, etc. Multiple conditional checks are used and the package name is passed to the calling methods `appInstalledOrNot()` and `isAppInSystemPartition()`. Both the methods are of Boolean types and will return only true or false. After these checks, the particular Rom variable of Boolean type is set to true.

detectAndroidVersion() – This method will check whether the android version is above 10 or not and will set the `above10` accordingly to true or false.



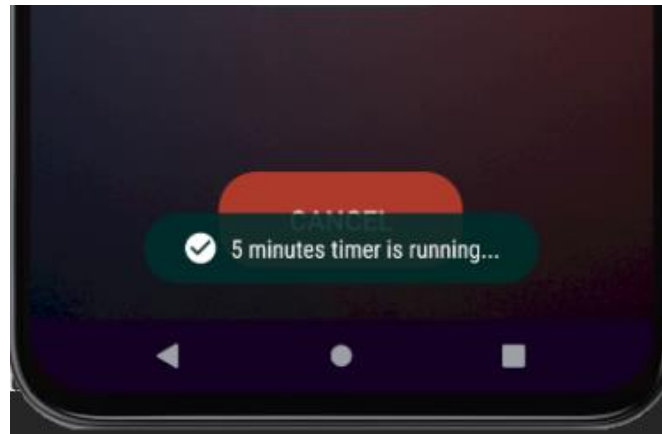
buttonsAreDisabled() – Whenever this method will be called preset buttons status will be set to as false or disabled and the text view colour will be changed to grey. For the cancel button, it will be visible.

showOptionsAsPerDevice() – When this method is called it will set the particular preset buttons to disabled state and will change the text colour to grey after checking the Rom variable.



buttonsAreEnabled() – When this method is called it will set the text for the text view, it will make the cancel button invisible, and set all the preset buttons to enabled state and changing the text colour to white. In the last showOptionsAsPerDevice() is called to set particular button status according to Rom.

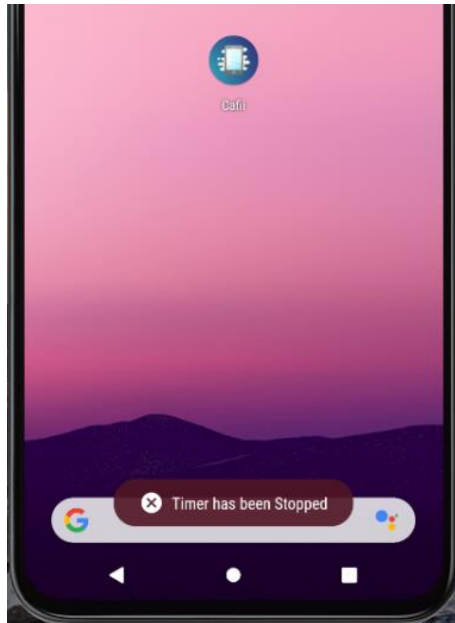
sendDataStartService() – When this method executed it will store time value in the shared preference according to which preset it has been selected and will also store the value of above10 variable. It will register the event bus and will start the service with the help of predefined method `startService(service_name)`. In the last button status it is set by calling the `buttonsAreDisabled()` method.



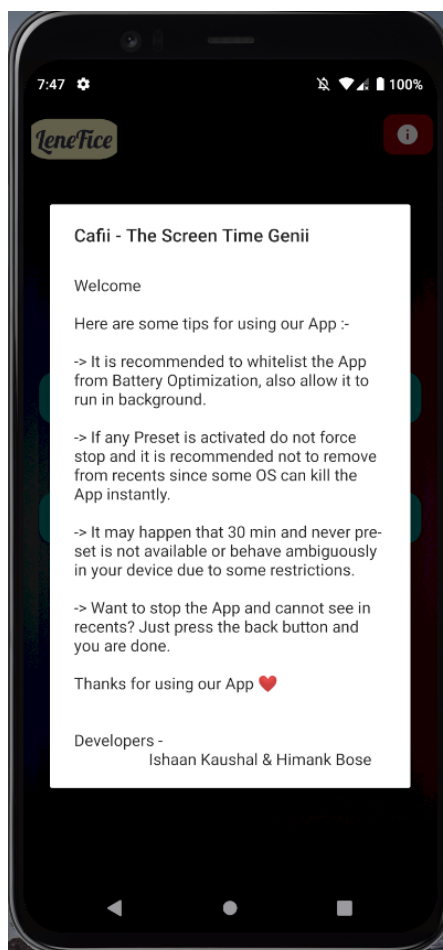
isMyServiceRunning() – When this method is called it will fetch the list of running services currently in android and will check whether our service is present in the list or not. If it is present in the list it means our service is already running and will return true, otherwise it will return false.

onEventStart() - This is a subscriber method that will receive events coming from the service to set the text view for the countdown timer and cooldown timer. Every second it will receive an updated time string from the service and will set to the text view.

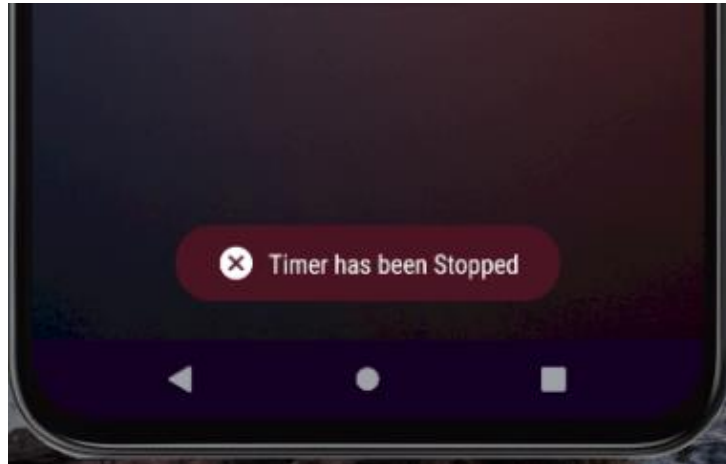
onAutoKilled() – This is also a subscriber method that will receive an event from the service just before it gets destroyed. If the main activity is running either in foreground or background then `buttonsAreEnabled()` method is called to set the button status.



showNoticeDialog() – This method will create an alert dialog box for our need and will show a notice whenever the info button is tapped.

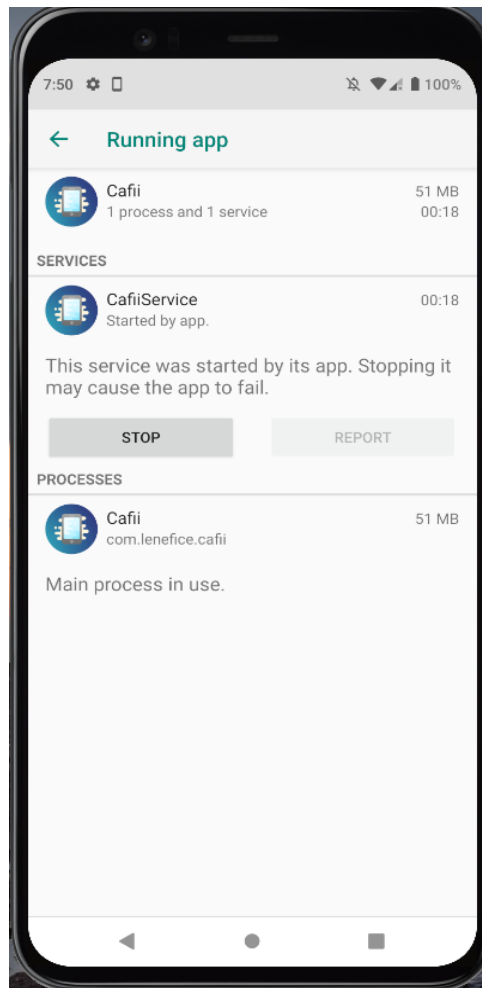


cancelTriggered() – Whenever this method is called an eventbus gets unregistered and service is stopped with the help of a predefined method known as the stopService(service_name) method.



CafiiService.java - Since we cannot perform the foreground and background operations with the help of an Activity, therefore we are using a Service that we have named as CafiiService. This class extends the Service class and like activity, it also has a lifecycle. And the methods which are inherited are like onCreate(), onStartCommand(), onDestroy(), etc. As soon as any preset is tapped by the user the service is started for that respective timer and also it creates a persistent notification so that the service keeps running in the foreground without being killed by the system.

This class also handles the modification of screen timeout in settings. It will back up the default timeout already set in your android phone and then it will change it to the respective preset and then timer will be started. After the timer is ended it will restore the default timeout and the service will be stopped itself.



Methods Used

onCreate() – Like activity, this method also exists in service class and similarly, we have overridden and defined the method to our needs. But it only gets executed once as soon as the service is started by the predefined method `startService()` method used in `MainActivity.java`.

In the starting, we are calling the super `onCreate()` method. We call the `getDefaultTimeOut()` method and access the shared preference with the same key that was used to store by `MainActivity.java`. It will get the value for the timer and `above10` variables.

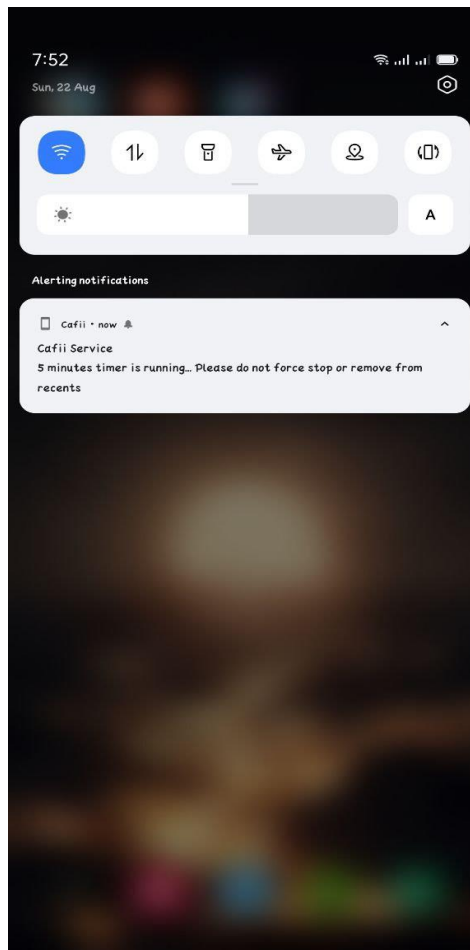
onStartCommand(Intent intent, int flags, int startId) – After the service has been created this method is executed to perform the operations as per our requirements. Like the onCreate() method this method will also override and defines our code. As the method execution starts customNotificationText() method and createNotification() method are called. After that condition check is performed to interrogate whether the newScreenTimeOut variable value is smaller than Integers max value. If it is true the time in seconds will be converted to milliseconds. At the last setTimeout() and setTimer() methods are called.

onDestroy() – Whenever the predefined stopService() method is called in any activity, in our case it is MainActivity.java or stopSelf() method is called within a service itself then this method gets executed. We call the super onDestroy() method in the last so that service gets destroyed. The method has been overridden and defined by us.

In this method we are checking status for both cool down timer and count down timer, if any of the timers are running they are cancelled. After that setTimeout() method is called by passing the value of defaultTimeOut. And then foreground notification is stopped with the help of the predefined method stopForeground(). And after it, it will be displaying the toast message as per the android version.

getDefaultTimeOut() – This method reads the default screen time out settings of the android device and will store it in the integer variable defaultTimeOut.

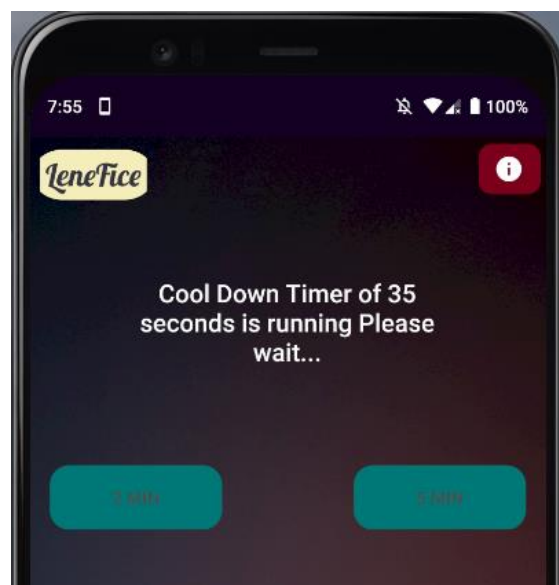
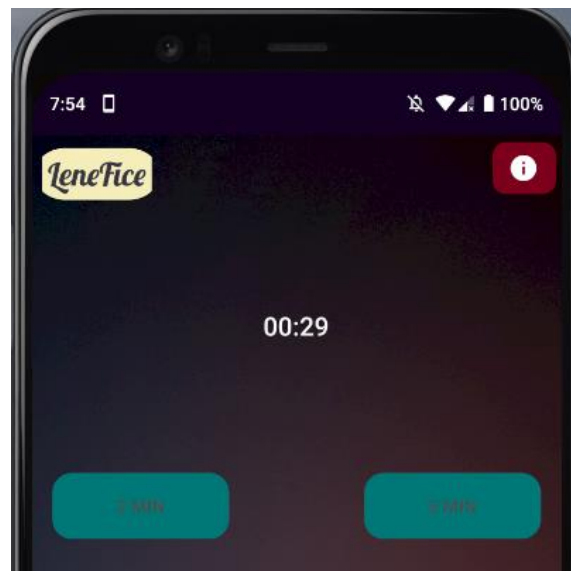
customNotificationText() – This method consists of a switch case statement that will store the String according to timer cases in toastNotify variable. After matching it with toastNotify variable will display the toast message accordingly.



createNotification() – This method will create a persistent notification that will be running in the foreground and if this notification is tapped it will launch MainActivity.java. This notification will display a message according to toastNotify variable. In the last, this created notification is started with the help of the predefined method startForeground().

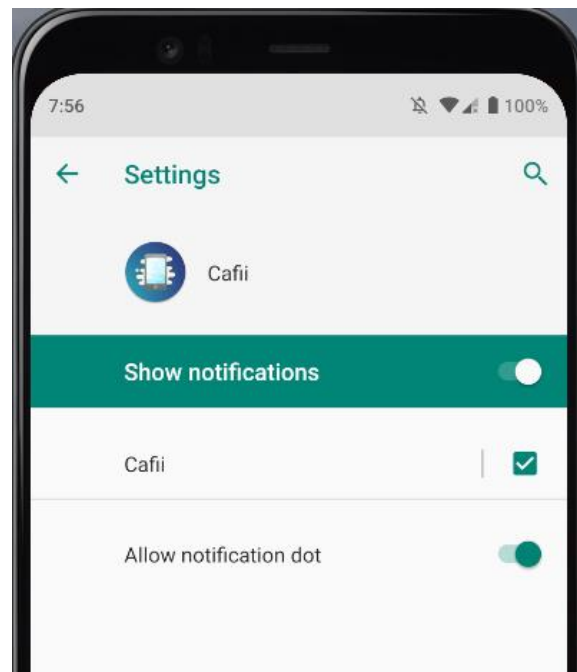
setTimeout(int milliseconds) – This method will take the coming value of milliseconds whenever it is called from any other method and will simply put this milliseconds as a new timeout in the system settings of android.

setTimer(int milliseconds) – This method will start an appropriate timer of about milliseconds that has been passed as an argument by other methods during the call. The status for count down timer is set to true in the isCountDTRunning variable. And then the count down timer is started. Every second event is passed for an updated time through the eventbus named EventTimer.java. After the count down timer has been ended, isCountDTRunning is set to false, status for the cooldown timer is set to true in isCoolDTRunning variable, and cool down timer of 35 seconds is started. After the cooldown timer ends isCoolDTRunning is set to false and an event is passed through eventbus AutoKilled.java, notifying MainActivity.java that the service ended itself. And in the last predefined stopSelf() method is called to kill service itself.



AppNotification.java - Service is only starting the persistent notification but this class helps in declaring and initializing the notification structure. In simple language, we mean that this class will create a notification of the type developer wants to show like persistent or non-persistent. From Android 8 and above we have a new feature known as Notification Channel, so it has been also included in it.

It consists of only one defined createMyNotification() method that will check if the android device is android 'O' or above then it will create a notification channel and the notification will be of high importance.

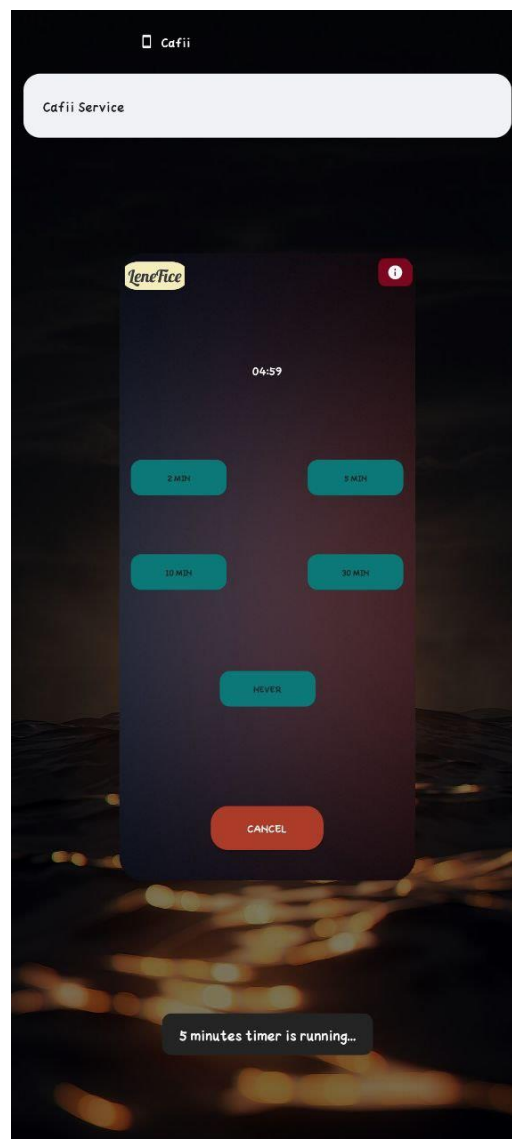


Eventbus - An Eventbus is a mechanism that allows different components to communicate with each other without knowing about each other. A component can send an Event to the Eventbus without knowing who will pick it up or how many others will pick it up. Components can also listen to Events on an Eventbus, without knowing who sent the Events.

That way, components can communicate without depending on each other. Also, it is very easy to substitute a component. As long as the new component understands the Events that are being sent and received, the other components will never know. Eventbus classes used are AutoKilled.java and EventTimer.java.

The AutoKilled.java will supply the event only once from the service to the main activity subscriber method when the service has ended automatically or cancel is triggered in the MainActivity.java that means ending service manually.

The EventTimer.java will supply the event every second when the count down timer or cool down timer is running inside the service to update the timer in the text view of the MainActivity.java.



Hardware and Software Requirements

Hardware: -

The minimum hardware configuration for application to run is as follows:-

Processor	1.5GHz Octa-Core processor
RAM	512 mb (80-100 mb of ram usage)
Storage	512 mb (5-10 mb usage)
Graphics	Integrated graphics is enough with above processor

Software: -

Operating System	Only Android
Android Version	4.4 – Latest

Front End

Android Activity

Back End

Android Service &
Persistent Notification

Written In

Java

Input and Output Requirements

The minimum Input and Output Device configuration for application to run is as follows:-

Input :-

Any type of touch screen or external mouse attached through OTG.

Output :-

Any Android Screen.

Weekly Contribution

Week 1 – In last semester we had created the prototype that was running on IDLE. So, it was decided earlier that we would be creating its real implementation on any of the devices for our major project. So, in week 1 we were analyzing through different devices, in which this feature is mostly required. And then the outcome of this was Android Smartphones. So, we decided to create its first implementation on Android.

Week 2 – We were going through the best implementation approach for our Cafii App. We have gone through different Android forums where we had found different ways of implementing that how to set the screen on for a definite amount of time. We mostly found two ways one was for rooted android devices and the other was for non-rooted Android devices.

Week 3 – After this, we had again analyzed what is the status of most of the android devices. We come to know that most of the devices were unrooted. And only a few percentages of people were using rooted android devices. We also carefully analyzed that there were some high percentages of custom ROM users but they still prefer not to use root.

Week 4 – After week 3 we decided to choose the approach for unrooted android devices. Also, because unrooted apps can be used on rooted android devices as well. But in this approach, there are some limitations. Limitations were android devices ROM other than Google Stock and AOSP may not support custom or few timers presets.

Week 5 – This approach was limited but was beneficial for all the users all over the world. Therefore, we had gone through all the coding and its understanding part in week 5. And we had an app ready that was just only capable of changing screen timeout in settings. After this we had started to think about the logic part.

Week 6 – We had tried to implement the logic we had developed. It took us about two days. It was just only a rough sketch of the application for the testing purpose of the logic. There was a basic UI design by that time. There were huge logic changes, about 2 times in week 6. And still, there was a big loophole related to screen time out.

Week 7 – We had added many fixes to the app that fixed most of the loopholes that resulted in crashing and freezing. After that, we had done optimizations and our application was ready, but only for AOSP devices with no support for foreground or background running. All the operations were loaded on MainActivity.java.

Week 8 – We researched how to detect the Android ROM. And we tried to implement it. It took about 2 to 3 three days to implement it and then we performed some testing on different devices. Some bugs that were found, we fixed them. The later week we had the Multi Android ROMs support in our App.

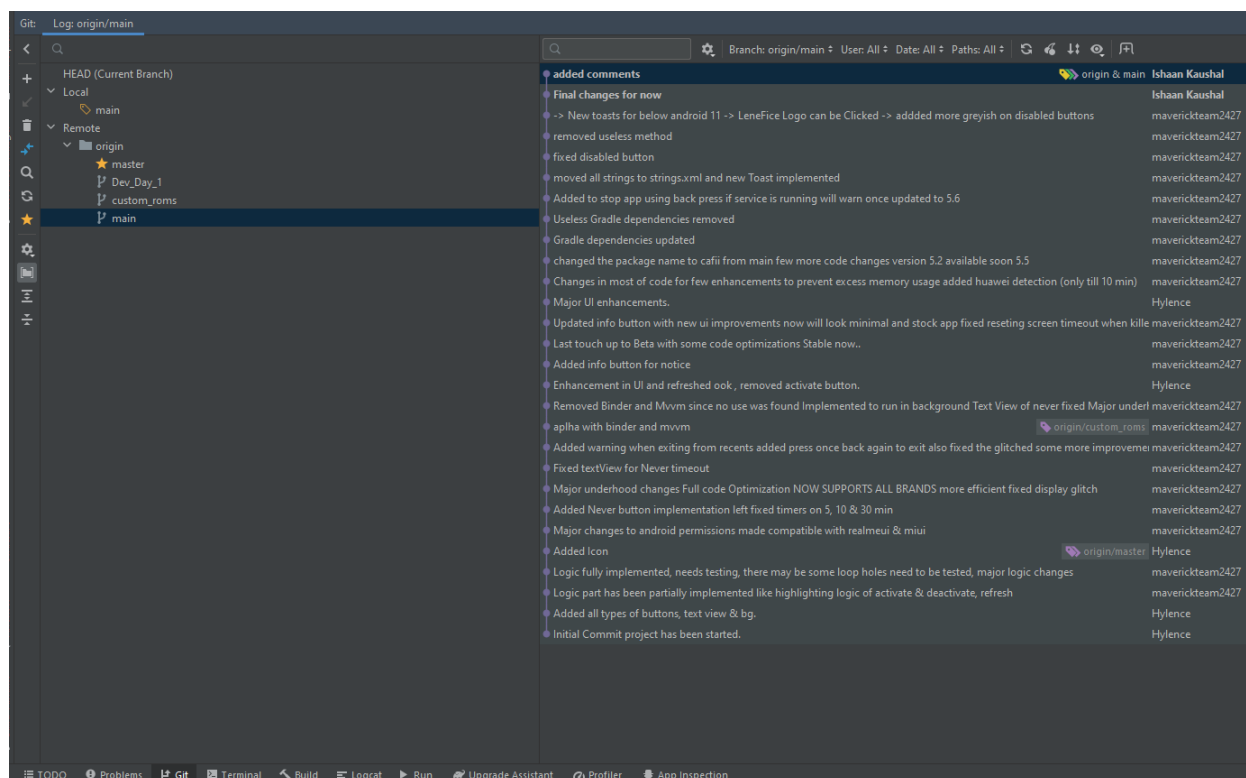
Week 9 – We had the major loophole to be fixed that was our app should run in background or foreground. We have gone through multiple forums, videos, and blogs for the right approach. We found many numbers of solutions. But the perfect working solution was adding foreground service to our application and separating MainActivity.java only for the user interaction part. We also had a great improvement over our logic part. And this resulted in robust logic.

Week 10 – We have created a lot of alpha apks to test out the proper working of the service in the foreground. Later this week we were able to get beta builds. And it was running fine for every Android ROM. We had also gone through the guides for persistent notification implementation so that the service can run in the foreground. Also looked upon eventbus concept to get well interaction between service and the activity.

Week 11 – We had completely analyzed the code from the beginning and had implemented proper methods to prevent unnecessary replication. We shifted all Strings to the strings.xml and similarly for colors we shifted to colors.xml. Now we had the final beta version of the application. The later week we had done thorough testing on the Final Beta Apk. It was found all well and stable.

Week 12 – We release the first stable version of the app that was Cafii 5.5. We had gone through more optimizations as much as possible. And completely redesigned the UI with info and brand buttons. We added Alert dialogue so that before App is used by someone, He\She can go through its limitations as per Android ROM they use. Finally, Major stable release build 5.6.1 was released.

Note – Some of the last features were completely removed like Activate/Deactivate button, the cancel button visible forever, and the removed custom timeout button. Added few things like toast messages, icons, etc. Git and GitHub were used for the creation of the Project. In most of the Android ROMs our App will be hidden from recents app list.



main ▾		4 branches	0 tags	Go to file	Add file ▾
IK sakshamgupta2427 added comments		536ca4d 8 days ago ↻			
📁	.idea	moved all strings to strings.xml and new Toast implemented			
📁	app	added comments			
📁	gradle/wrapper	Removed Binder and Mvvm since no use was found			
📄	.gitignore	Initial Commit project has been started.			
📄	build.gradle	moved all strings to strings.xml and new Toast implemented			
📄	gradle.properties	Initial Commit project has been started.			
📄	gradlew	Initial Commit project has been started.			
📄	gradlew.bat	Initial Commit project has been started.			
📄	settings.gradle	Initial Commit project has been started.			

SDLC Model Used

In our major project, we have used the SDLC model known as Spiral Model. The spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.

The reason for using this model is we had continuously figured out many bugs, glitches, and many more loopholes that were causing app crashes, freezes, and ambiguity. So as the number of times we have gone through we added many fixes, improvements, optimizations, and dynamic approaches. After many numbers of loops, we have achieved a stable version of the app making it as robust as possible. Several times there was modification regarding the logic part. And also, there were huge improvements so that the app can easily recognize android ROMs. Since there were a lot of logic part changes there was the requirement to change the UI part as well.

In our project, we have taken Git into heavy use for development. Not only we used git but also, we created a Repo at GitHub and pushed the source code on it. At the early stage of the project, we had created only an activity to handle all the operations. But later on, we faced many challenges like multi android ROMs support, to protect our application from being killed by the system, and limitation of activate and deactivate button. We created different branches for different feature additions and different bug fix.