Mobile Based Source Code Editor

by

Gurjeet Singh
Bachelor of Technology, Guru Gobind Singh Indraprastha University, 2012

A Project Submitted in Partial Fulfillment
of the Requirements for the Degree of
MASTER OF SCIENCE
in the Department of Computer Science

University of Victoria

Mobile Based Source Code Editor

by

Gurjeet Singh
Bachelor of Technology, Guru Gobind Singh Indraprastha University, 2012

Supervisory Committee

Daniel M German, Department of Computer Science
Supervisor

Samer Moein, Department of Electrical and Computer Engineering
Member

Abstract

Mobile based source code Editor (MBSCE) is an Android Mobile application that helps in editing user's java source code. It also provides many other efficient features like code indentation, keywords coloring, code grouping/ungrouping, cloud access (with Google Drive), code sharing (with email) and auto text completion.

The application has a variety of potential users like students, developers who like to access their source code on their mobile devices. It helps its users to gain access to their source code stored on their Google drives. Once the code is accessed, this application colors the java keywords, indents the source code, creates buttons to group/ungroup blocks of code and finally displays the code on the user's mobile. Then the user can modify the source code on his/her mobile and can save the code back to the google drive. User can also write the code from scratch, upload it to the google drive and share it over an email.

This report provides the details regarding the problem statement and the requirements of the application created. It also provides the in-depth details about the use cases, design, implementation of each feature, benefits, limitation and in the end, it also enlists the possible future enhancement that can be done later.

Table of Contents

List of Figures

List of Tables

Glossary

1) MBSCE : Mobile Based Source Code Editor
2) UML : Unified Modelling Language
3) IOS : I-phone Operating System
4) SDK : Software Development Kit
5) XML : Extensible Markup Language
6) IDE : Integrated Development Environment
7) API : Application Programming Interface

Acknowledgement

I would like to thank **Dr. Daniel German** for his continuous support and mentoring throughout the project. His valuable feedback and suggestions helped me a lot to improve my application.

Also, I would like to thank my friends and family who encouraged me during this project.

1. Introduction

These days we have so many editors available to edit/format source code almost for any programming language. Some of these famous editors are Notepad++, Atom, Text-Edit etc. The problem with these editors is that you need to have a computer system or a laptop to use these applications. In other words, there is an additional overhead of downloading/installing these applications on a computer. So, in the era of fast moving world where the technology has almost been shifted to mobile phones where you can do almost everything, there is a need to create a mobile application that can edit/format source code written in different programming languages.

MBSCE is an android application that allows its users to access their cloud directory (Google Drive) to load java source code on the mobile device to edit it. This application allows the user to use this application on the phone without the need of installing or purchasing traditional editors on his/her system (laptop or computer system). The application is very user friendly with a simple Graphical User Interface (GUI) to use. In addition, this application has many other prominent features which will be mentioned in detail in this report.

1.1 Target Market

For this application, mainly two kinds of users have been identified who can use this application depending upon their usage. Following are the expected users and their type of usage:

- **High School Students**: Students who are learning programming in Java can use this application to edit their programs on their phones.

- **University students or Developers**: These people can use this application for making changes into their code in a flash by just accessing it on to their cell phones and then saving it back to their drives.

1.2 Android platform

Android is an operating system for mobile phones, tablets and many other smart devices. It is developed by Google. It gives us a flexible user-friendly environment to create applications (commonly known as Apps) and games that can be distributed to the users in the open marketplace instantly.

Shown in Figure 1, is the Android platform[1] that specifies various components at different level. On the bottom layer, there is a Linux based kernel and it also holds drivers for USB, Display, Bluetooth etc. Layer next to it (moving bottom to top) has Android Libraries that can be used while creating applications. This layer also provides the Android runtime environment. Next layer provides various managers like resource manager to manage resource usage by an application, activity manager to manage the flow of control, content provider to use data from other applications etc. The final layer (top layer) hosts the native, third party and developer applications.

Figure 1 is referred from *http://mrbool.com/android-development-framework/29690*

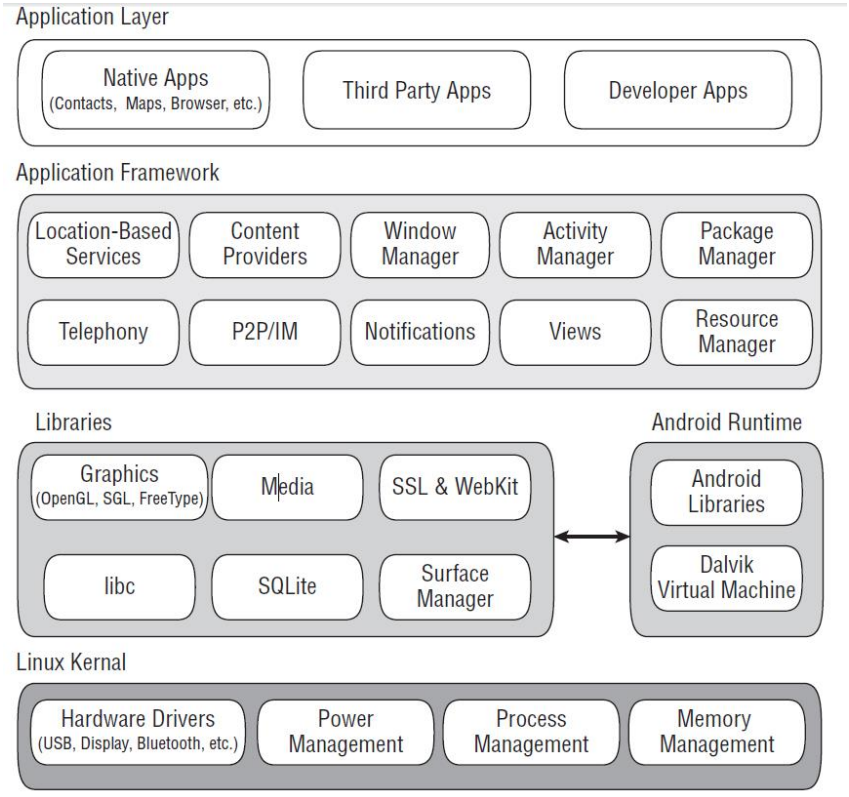

Figure 1: Android platform

1.2.1 Benefits of the Android framework

To build MBSCE, I used Android framework, Android SDK (version 26), Java and XML. Mention below are the two main reasons that I chose to build this application in android.

- **Previous experience in Java**

  It was tougher in the beginning when I started understanding the Android framework, but with my previous experience in Java I was able to grasp new things quickly. I also thought of building this application in IOS but for that I needed to learn a completely different language (Objective C or Swift).

- **Easy and flexible IDE**

  I used Android studio[9] which has a very easy and flexible integrated development environment. GUI is built easily using drag/drop options. It also has an inbuilt emulator where you can test your application with/without debugging.

1.2.2 Application life cycle of an Android Application

The heart of an Android application lies in a very simple concept called Activity which is a single screen with a user interface just like windows. Android application life cycle[12] is very simple which pushes the "Activity" in different states depending upon its interaction with the user.

Some of the prominent methods of the application life cycle shown in Figure 2 are mentioned below:

- onCreate() – called when any activity is created. Also used to initialize the activity.
- onResume() – called if the activity comes into use again after it gets paused by calling onPause() method.

- onPause() – called when the activity stops interacting or some other activity comes into the foreground.
- onStop() – called when the activity is no longer visible. Operations like closing file, releasing resources should be handled in this method.

Figure 2 is referred from *https://developer.android.com/guide/components/activities/index.html*

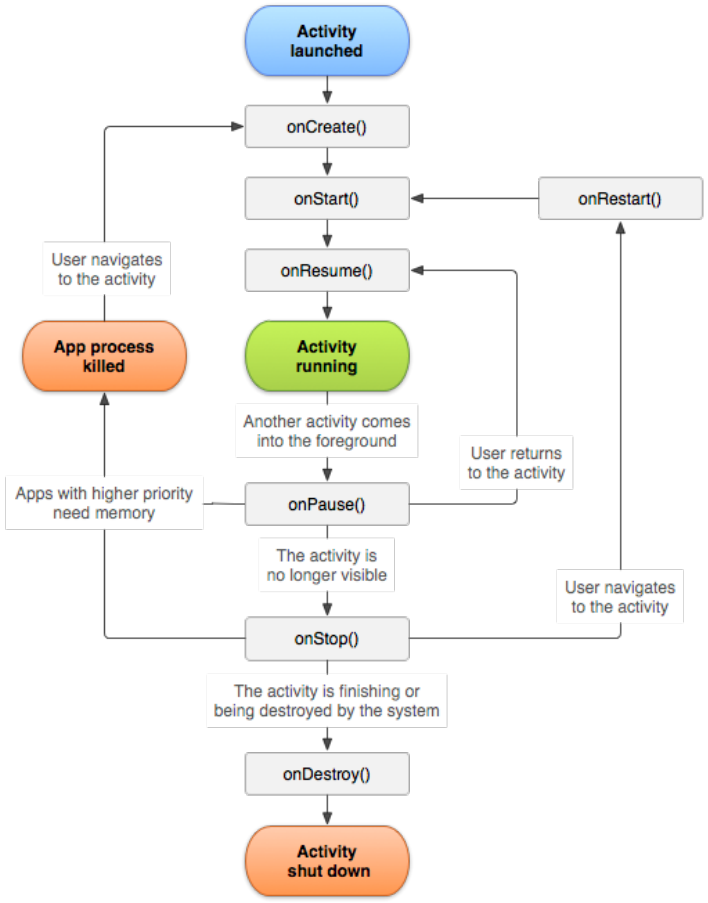

Figure 2: Android application life cycle

Some of the important states of any activity are:

- Running – ensure activity is running, visible and interacting with the user.
- Paused – activity is still visible but is paused by other activity that comes into foreground.
- Stopped – means activity is not active any more i.e. it is stopped.
- Killed – means activity has been terminated by giving a call to finish () method.

1.3 MBSCE walk through

There are several other editors available online or there are editors available that can be downloaded on the system to edit/format Java source code. My application (MBSCE) does not have the same set of features but it does have the basic functionalities of an editor along with google drive access.

Mentioned below are some interactive features of my application:

- **Google sign in Authentication:** This application authenticates the user using the google sign in. If the authentication is successful, the user can use the application else not. This feature is important considering the security of the user's Google drive. User can use the existing Gmail account on his/her android phone or can login with the new account.

- **Coloring source code**: Java source code can be divided into 5 types of elements mainly variables, classes, functions, comments and statements. To increase the readability of the source code, some of these elements can be represented by some set of respective colors. Different colors help in segregating the above-mentioned elements and hence increases the readability.

- **Indenting source code**: It is another important feature to increase the readability of the source code by logically diving it in terms of blocks by providing proper spacing in the beginning of each statement under that block. After the indentation is done we can easily see the scope of classes, functions and if statements. The scope of these things is defined by an opening brace "{" and a closing brace "}".

- **Code Collapse/Expand**: With the limitation of a phone screen size, this feature will help in collapsing or grouping a block of source code in one line. Let's say user has 3 blocks of code with each block having 50 lines. Using this feature user can collapse a block and can see the next block instantly without the need of continuous scrolling. Once a code is imported from Google drive or written from scratch, automatic buttons are displayed using which a user can collapse or expand the blocks of source code.

- **Auto Completion capability**: This is a feature by which the implicit characters are inserted automatically even if the user misses to write it. For the Java source code, if a user puts a double quote (“), a single quote (‘), an opening brace ({), an opening bracket (( ), and an opening square bracket ([ ) then this application automatically inserts corresponding closing quote, brace, bracket etc.

- **Access to cloud Storage (Google Drive)**: Not only a user can write the code from scratch but he/she can import the code from his/her Google drive. The source code file can be opened in the application GUI and then the user can format or edit it and then save it back to the drive. User can also write the source code from scratch and save it as a new file on the Google drive. User can use the existing Gmail account on his/her android phone or can login with the new account to use this feature.

- **Ability to share the code over an Email**: The application allows the user to share the code over an email after the modifications/changes are done. User can use the existing Gmail account on his/her android phone or can login with new account to use this feature.

2 Software Modelling and Design

A brief object-oriented design for MBSCE has been developed. The Unified Modelling Language (UML) is used for the graphical representation of the design.

2.1 Use case Diagram

2.1.1 Use Case 1

"User accesses the source code, make changes and saves it back to the drive or share it over an email"

User first signs in using his/her Google account. Next, Google drive is accessed to download a sample source code to edit. Next, as the code is being edited it simultaneously gets indented, keywords get colored and collapse/expand button positions get updated. Finally, after the editing is done, code is saved back on the drive or shared over an email.

Use case diagram for the above use case is shown below in Figure 3. Blue lines represent the flow of events and black lines represent the features included in a specific feature.

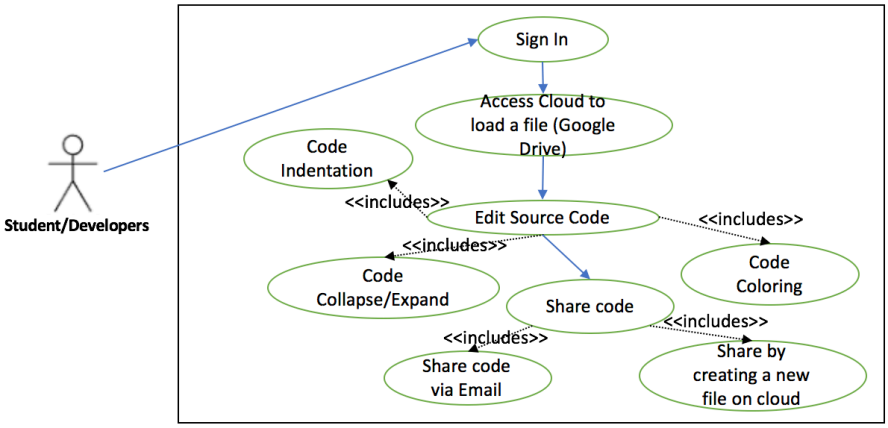

Figure 3: Use case 1

2.1.2 Use Case 2

"User writes the code from scratch and saves it as a new file"

User first signs in using his/her Google account. Next, as the code is being written it simultaneously gets indented, keywords get colored and collapse/expand button positions get updated. Finally, it is saved on the drive or shared over an email.

Use case diagram for the above use case is shown below in Figure 4. Blue lines represent the flow of events and black lines represent the features included in a specific feature.

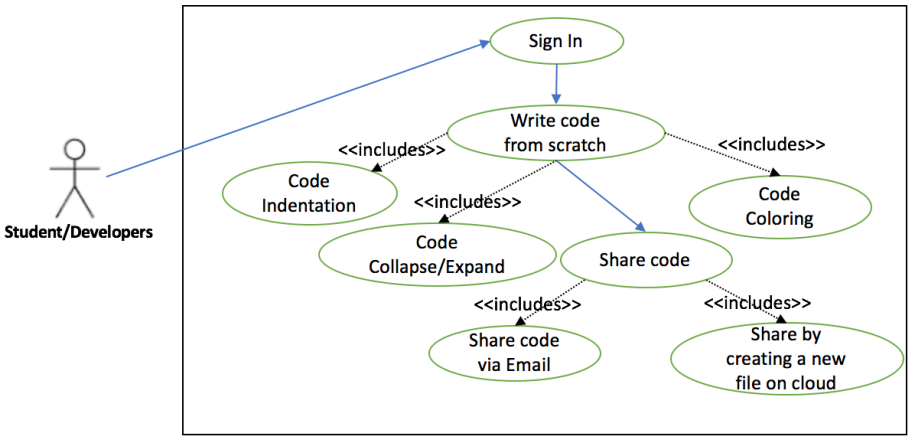

Figure 4: Use case 2

2.2 Sequence Diagram

This section describes how sequence of events are called over time in the form of sequence diagrams for each use case mentioned in section 2.1.

2.2.1 Sequence diagram for Use Case 1

The flow of the events for the use case *"User accesses the source code, make changes and saves it back to the drive or share it over an email"* is shown below in Figure 5.

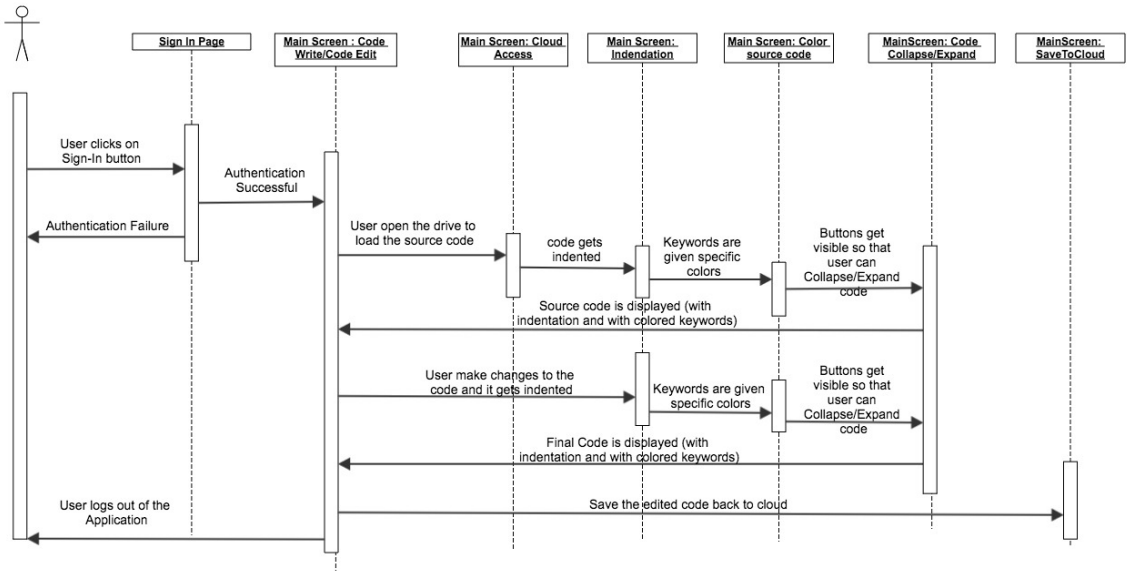

Figure 5: Sequence diagram for first use case

- Flow starts with the user signing in on the Sign-In page. If authentication is successful, the user is directed to the next screen which is the main screen, else the user is presented with the login screen again due to invalid credentials.

- From the main screen, user loads a file from the Google Drive.

- Once the source code is loaded, it gets indented, keywords are colored, Collapse/Expand buttons get visible on the screen.

- User then make changes to the source code. The code gets indented, keywords are colored, Collapse/Expand buttons get refreshed on the screen.

- Next the user can save the code back to the drive or can share it over an email.

- User can then Log out of the application or can load a new file again.

2.2.2 Sequence diagram for Use Case 2

The flow of the events for the use case *"User writes the code from scratch and saves it as a new file"* is shown below in Figure 6.

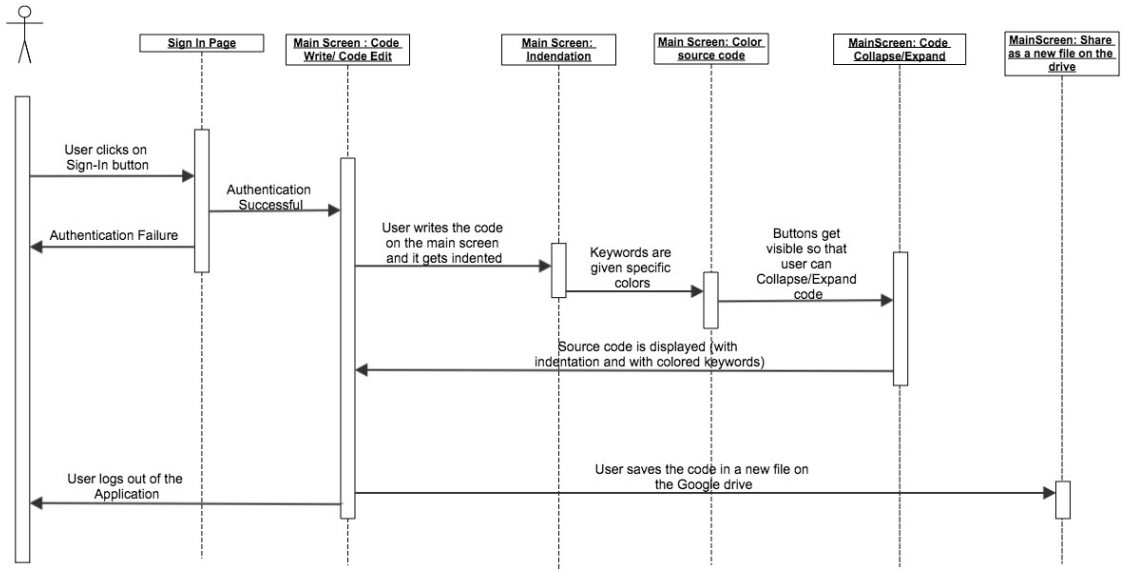

Figure 6: Sequence diagram for second use case

- Flow starts with the user signing in on the Sign-In page. If authentication is successful, the user is directed to the next screen which is the main screen, else the user is presented with the login screen again due to invalid credentials.

- On the main screen, user writes a java code from scratch.

- The code then gets indented, keywords are colored, Collapse/Expand buttons get visible on the screen.

- Next the user can save the code back to the drive as a new file.

- User can then Log out of the application or write a new file.

2.3 Graphical User Interface

Shown below in Figure 8 is the graphical user interfaces created for this application.

A. **Sign-In Page**: As shown in the left half of the Figure 7, user has to sign in using the valid Gmail ID.

B. **Main Page**: Shown below in the right half of the Figure 8 is the main page of the application. In the middle, there is a text box where user can write/edit his/her source code. On the most left pane, there are line numbers. Next to it are the buttons to Collapse/Expand the source code. The top section just displays the application name. Descriptions for other features are highlighted below in Figure 7.

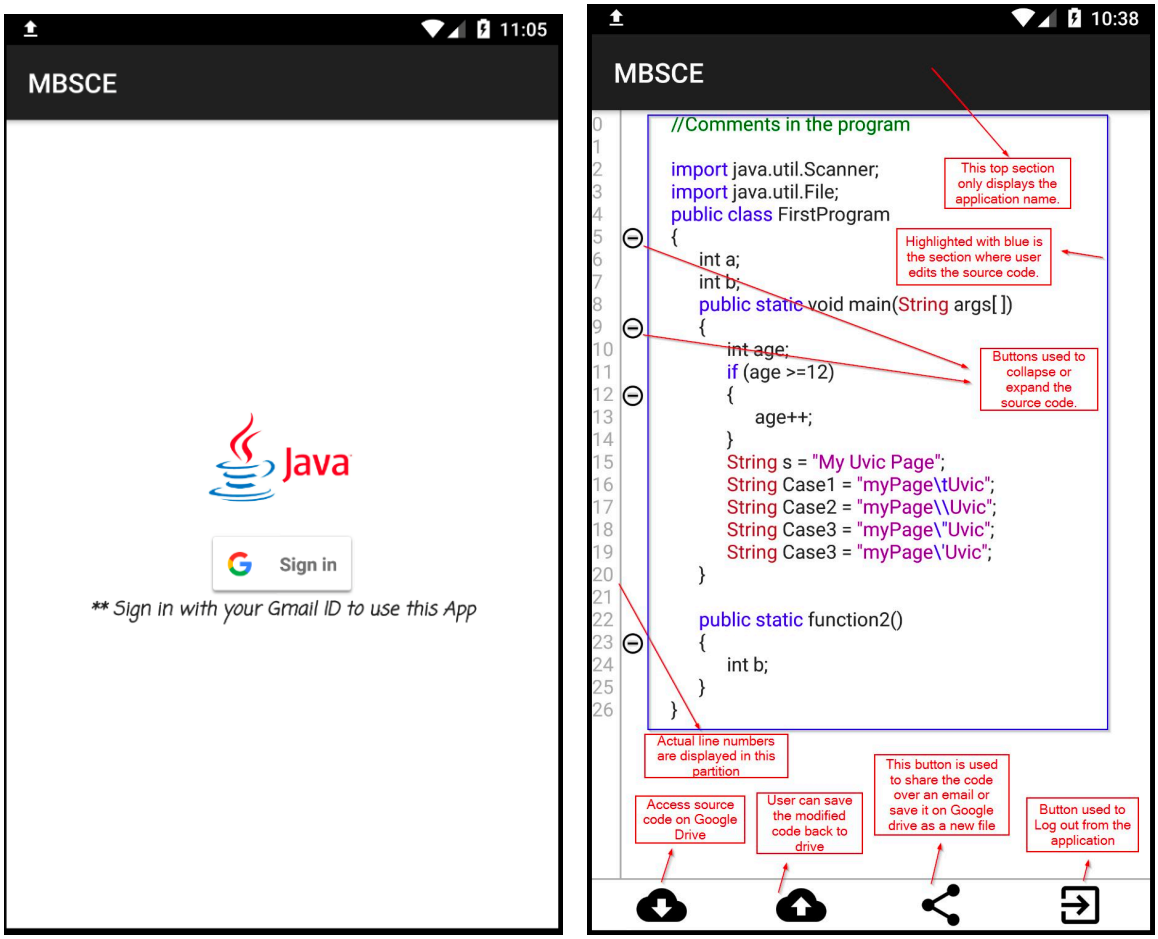

Figure 7: Sign in and main page

3 Technologies and Tools

This section briefs the technologies and tools used in the implementation of this application.

Technology	Description
Java Development Toolkit	Java development toolkit version 7 has been used in this project.
XML	Graphical user interface designing has been done using the XML language. Besides this other configuration files are also written in XML.
Android Studio	Provides the fastest tools for building apps on every type of Android device. Allows code editing, debugging, performance tooling and an instant build-deploy functionality.
Android SDK (Software Development Toolkit)	Provides packages to run android applications. Also support integration with many Google APIs.
Android Emulator	Allows us to install and run the apps faster than a physical device. Also gives us the option to test the application on tablets, Android wear and Android TV devices besides just an Android phone.
Genymotion	Alternate emulator used because of some limitations of the Android Emulator provided by Android studio.
Virtual Box	Installed as a prerequisite to install Genymotion.

Table 1: Technologies and tools used in the project

4 Implementation of Mobile based source code editor

4.1 Programming languages

The implementation of MBSCE involved the use of the following programming languages.

- **Java and Android programming language**: Object oriented features of java along with many android packages have been used in the implementation by creating separate classes and making use of code reusability.

- **Graphical User Interface**: For creating a simple and user-friendly interface, Android studio has been used because of its flexible, interactive and debugging features. The front end has been built using XML where you can set properties of the objects that are there on the screen. You can even drag/drop various component and set their properties in the properties pane of the android studio.

4.2 Features of the application

This section gives a detailed overview of all the features implemented in the MBSCE with some screenshots.

4.2.1 Google Sign-in Authentication

This feature ensures that only the authenticated users, who have a valid Google account, can use this application.

This feature has been implemented using the Google Sign-in API. Before any Android application can use the Google Sign in authentication, it has to do some initial configuration and key generation. First, there is a need to add a configuration file (google-services.json) to the project which can be downloaded at developers.google.com. The configuration file provides service-specific information for the application. Next, the plugin is included in the build.gradle file that handles all the builds of an application. Next, an API key is generated that authenticated my application to use this API. Next, an OAuth client ID is generated that actually requests the user's consent so that my application can access the user's data.

After the above steps are done, an object of class GoogleApiClient is initialized that is used to connect to Google Sign-in API. Now if a user clicks the sign-in button, an internal activity appears asking for user's credentials. On submission of this activity, the result code is checked internally by the API and returns the results which is then compared in the onActivityResult ( ) method. If the result matches the predefined static integer constant (RC_SIGN_IN) then the credentials are valid and user is navigated to the main screen. Figure 8 shows the sign in page of this application.

Classes and important methods implemented/overwritten are mentioned below.

- **Class**: SignInActivity, GoogleApiClient
- **API**: Google Sign-in API (GoogleSignInAPI)
- **Methods:**
    - **Intent GoogleSignInApi.getSignInIntent (mGoogleApiClient)** → On click of Sign-In button by the user, a method called signIn( ) is called. This method in turn calls the getSignInIntent ( ) method which is just used to get user's Gmail credentials and it is called with an object of class GoogleApiClient. The function returns an intent object that holds the credentials data.

    - **void startActivityForResult (intent)** → performs validation on user's credentials and returns the result in the form of intent object by calling **onActivityResult()** method.

    - **void onActivityResult (int requestCode, int resultCode, Intent data)** → gets the result of the validations performed in **startActivityForResult()** in terms of 3 parameters mentioned above. This method does not return anything but in its overwritten implementation we can compare the results with expected value. If they match, the user is navigated to the main screen.

4.2.2 Coloring Key Words

This is one of the important feature that helps in separating the java source code into comments, variables, classes, functions etc. by coloring different elements with specific colors.

I have used six different colors to represent different elements.

- **Blue**: Used for 53 different reserved java keywords[11] like (if, static, int, long etc.). This covers a large set of java keywords. Escape sequences like \t, \', \n etc. are also handled with this color.
- **Green**: This color is used for coloring single line comments only followed by //.
- **Maroon**: This color is used to color String object's data within double quotes and characters within single quotes. It is also used to color ~4000 classes of Java API[8] version 7.

- **Black**: This is used for all the other java statements.
- **Grey:** Used to color the line numbers on the extreme left side of the main page.
- **Dark Orange:** Used for the blocks of code that are collapsed or grouped in one line.

Initially the user's code that is visible on the main page is split into lines and is passed to the *setColor(EditText mainEditText*) method as one complete String. Then the parser runs on this string of code and compares each word (separated by a space) with elements that can be colored Blue (if, static, void etc.), Maroon (string between double quotes), Dark Orange (block of collapsed code) and then assigns the respective color if the match is correct. Final comparison is done to find if the word actually starts with 2 forward slashes or not. If yes, that means it's a comment and then the parser colors everything Green starting with // until the end of the line (\n) is found. Everything that is left remains Black in color which is the default color of the String.

To color portions of a string with different colors, I have used a SpannableString[4] class of the android text package that takes the reference of the string passed to it for initialization and can then color portions of the same string with different colors. Shown below in Figure 8 is the output of how a source code looks like after the portions of it are colored with different colors.

Class and methods implemented for this functionality are mentioned below.

- **Class**: ColorManager
- **Method(s)**:
    - **SpannableString setColor(EditText mainEditText)** → This method takes the reference of the Text Box that holds the user code and use SpannableString class in the android text package to color portions of the user code with different colors.

      In the end, this method returns the modified SpannableString class object that holds the user code which is then finally updated on the user's main screen.

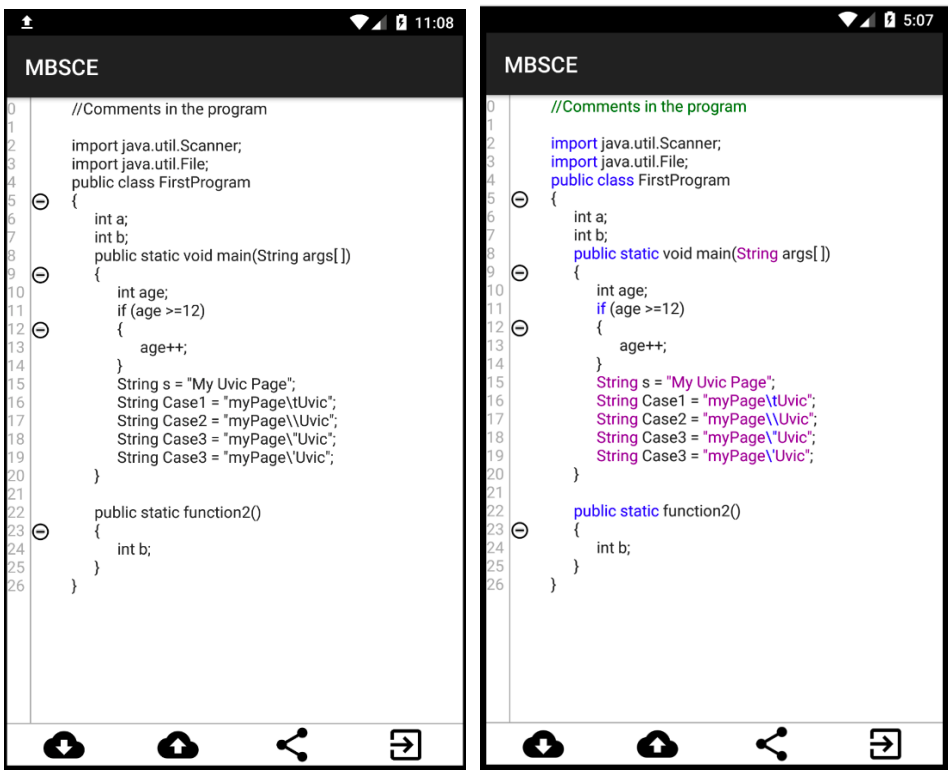

Figure 8: Before and after coloring portions of the source code

4.2.3 Indenting source code

Indentation is one of the prime factors that defines the readability of the java source code and it also conveys information about the program structure to its readers. Almost every programming language has some set of rules to indent its source code but the main motive is always same i.e. to increase the readability of the code.

To implement this functionality, first the java source code is split into lines since the source code in the text box is always in the form of one complete string. Next, a stack is initialized which only stores the location (or line number) that contains an opening curly brace ("{"). Next each line is parsed and the following cases/functions are performed.

- If there is an opening brace "{" in the line, the line number is pushed to the stack using the push ( ) method. Let's say value A is pushed to the stack.

- If there is a closing brace "}" in the line, then the top element is popped from the stack using the pop ( ) method which gives the corresponding location (or line number) of the opening brace. Let's say the line number in which the closing brace is found is B and the element popped is A. Finally, all the lines between A to B are appended with a fixed space of six blank characters in the beginning and the algorithm continues until it parses the last line.

- If none of them ({ or }) is found in a line then that iteration or that line is ignored. Algorithm then scans the next line and continue until it parses the last line.

Output of a source code is shown in Figure 9 after the indentation is applied to it. Class and important methods implemented to achieve this functionality are mentioned below.

- **Class**: IndentationManager

- **Method(s)**:

    - **List<String> getIndentedText (List<String> lines)** → this method takes the source code split by line ("\n") and returns the indented lines of code. It parses the code line by line and pushes the line number to a stack whenever it parses an opening brace ("{"). Once it finds a closing brace ("}") on any line, an element is popped from the stack and then it adds a fixed space in the beginning of all the lines between opening and closing brace location. It continues until it parses the last line.

    - **int getCursorLocation (int currentLine)** → This function is called once the user enters a new line character by hitting enter. Then this function uses the line number passed to it as a parameter, calculates the new cursor position and returns the same.

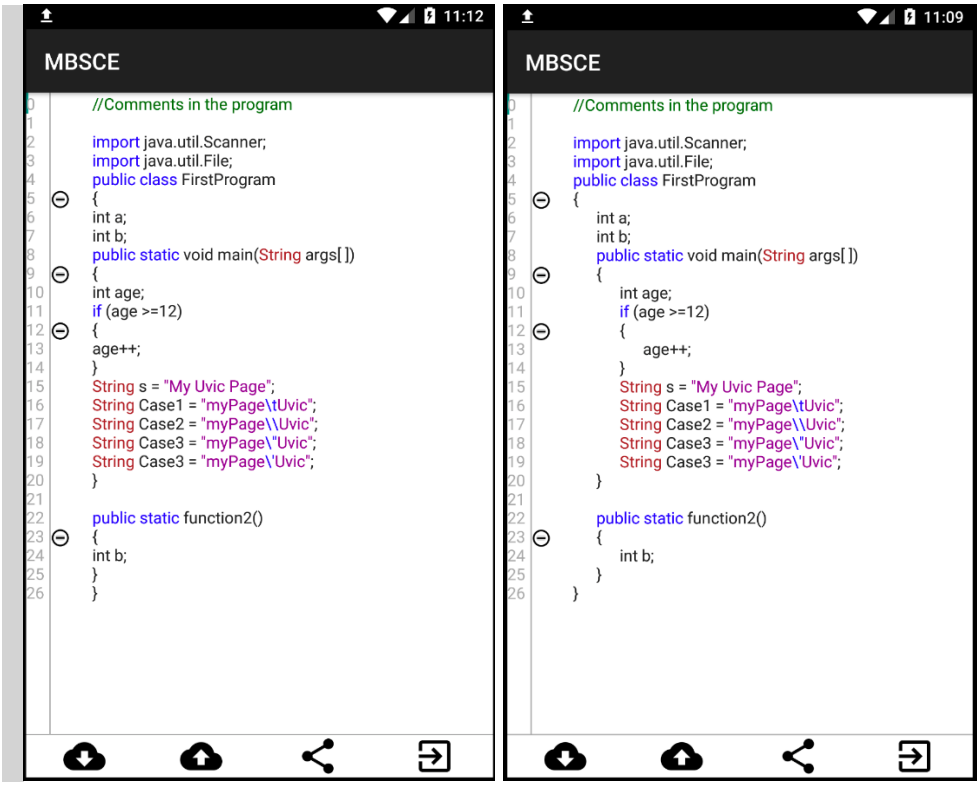

Figure 9: Before and after source code indentation

4.2.4 Code Collapsing/Grouping

Due to the limitation of the screen size of any phone or screen, this feature needs to be there in any editor; otherwise the user needs to scroll a lot to see long source codes. Source code with 90-100 lines is still fine but with increase in number of lines, there should be a way to scroll down in a flash by grouping or collapsing the blocks in the source code represented by opening and closing braces. Also, it's one of the very hard problems because many things are to be taken care once the source code is collapsed.

Initially when the source code is loaded from Google drive, it is split into lines and stored as a list of Strings. Next, when the parser scans each line it adds a minus (-) button (just before the start of line) if that line contains an opening brace ("{"). Button is assigned an ID which is the

current line number itself. Initially all the buttons created will have a minus sign (-) as its source image assuming the source code is fully expanded in the beginning.

Also, OnClick listener gets set on the button which basically toggles the button and change it to a positive button (+) by changing the image source only. Next in sequence, when the indentation feature is called it fills a data structure (defined by me) that holds opening/closing location of the blocks (represented by { and }). It stores this information in the form of parent-child relationship i.e. it knows which blocks of code are nested under which block. This information is necessary because when user collapses a block of code, all the nested block of codes should be hidden. Also, there are 2 lists of Strings maintained throughout the execution. First is the true copy that has the actual source code split into lines and second list stores the code that is currently displayed on the screen (Collapsed or Expanded).

Now using the methods below, once the user presses a minus button (-) to collapse a code at a particular line, the above-mentioned structure is referred to find the corresponding line where the block should close and replace the whole code by {….}. It also changes the current button image to (+), recursively hides the nested blocks and hides the nested buttons as well. Similar operation is performed in the reverse order when user expands the code.

Class and important methods implemented to achieve this functionality are mentioned below.

- **Class:** ButtonManager
- **Method(s):**
  - ✓ **void createButtons(List<String> lines)** → creates buttons and returns nothing.
  - ✓ **void addButton(int lineNumber, int actualLineNumber)** → adds a button at a specific line number.
  - ✓ **void removeButton(int lineNumber)** → removes a button on a line number.
  - ✓ **void moveButtonsDown(int index, int length)** → move all the buttons down starting at index (or line number) by a length.
  - ✓ **void moveButtonsUp(int index, int length)** → move all the buttons up starting at index by a length.

- ✓ **String getCollapsedString(List<String> indentedLines)** → It returns the collapsed source code as string object once the user collapsed/grouped any block of code.
- ✓ **hideButtons(Map<Integer, CollapseDetails> children)** → Once a block of code is collapsed/grouped, this function hides the nested buttons between the block.

This feature implemented on a sample source code is shown below in Figure 10.

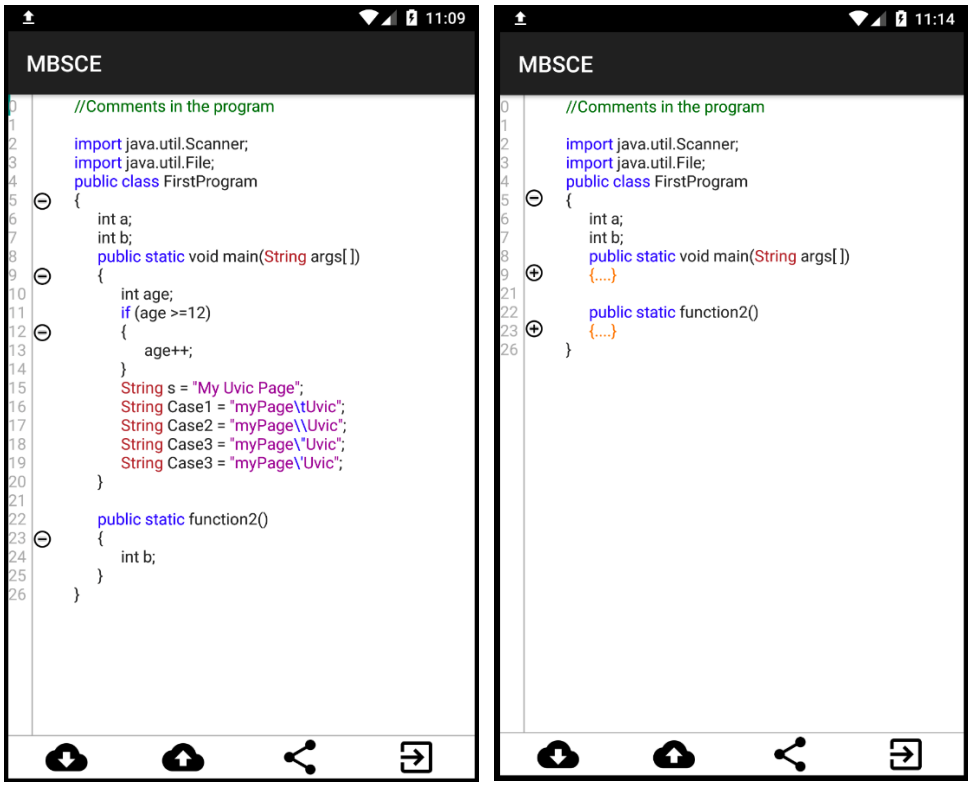

Figure 10: Before and after source code collapsing

4.2.5 Auto Completion capability

This feature of the application automatically inserts the closing implicit characters when the user enters the following characters:

‘ : **Single Quote** → Closing single quote is inserted just next to the opening single quote on the same line and the final cursor position is set between them.

“ : Double Quote → Closing double quote is inserted just next to the opening double quote on the same line and the final cursor position is set between them.

{ : Opening curly brace → Closing curly brace “}” is inserted just next to the opening curly brace on the same line and the final cursor position is set between them.

[: Opening square brackets → Closing square bracket is inserted just next to the opening square bracket on the same line and the final cursor position is set between them.

(: Opening bracket → Closing bracket is inserted just next to the opening bracket on the same line and the final cursor position is set between them.

To handle this functionality I have applied a listener on the Textbox that holds the user’s source code and also overwritten the onTextChanged ( ) method of the Android Text class. Listener for Text is commonly called Text Watcher in Android.

Next, If user writes something new in the Textbox that holds user’s source code, then the onTextChanged ( ) method records the new character and save it in a global variable. Then this new character is compared with the characters mentioned above (single quote, double quote, curly brace etc.) and if there is a match, the corresponding closing character is inserted.

Class and important methods used to implement this feature are mentioned below:

- **Class:** Processor
- **Method(s):**
  - ✓ **void setTextWatcher ( ) →** This method returns nothing and is only used to set listener on the text box that contains the source code. Once the new text is added to the source code, an overwritten method onTextChanged ( ) is called.

- ✓ **void onTextChanged (CharSequence s, int s, int b, int c)** → This method is called automatically by the text watcher (with four parameters mentioned above) when there is a change in the text. It returns nothing but it stores the newly added text in a global variable using the parameters passed to it when this function is called.

- ✓ **void handleCharacters (int lineNumber)** → This function gets the line number as the input and it uses the global variable updated by the onTextChanged ( ) method mentioned above. It compares that variable with the characters (single quote, double quotes, opening bracket etc.) and if there is a match, it inserts the corresponding closing character.

4.2.6 Access to cloud:

Using this feature user can access his/her java source code from the cloud which is the Google drive only. For this feature, Google drive API[7] is imported and some of its functions are overwritten as per the requirement of the project.

To implement this feature, a google API client (googleApiClient) of class GoogleApiClient is initialized and is linked to the drive API. Next, once the user clicks on the download or upload button as shown in Figure 8 before, newOpenFileActivityBuilder ( ) method is called which in turn opens the Google Drive and returns an object of type IntentSender that stores the reference of the opened Google Drive.

Now user can perform two actions:

- If the user has clicked on the download button then a new method called startIntentSenderForResult ( ) is called with an object of type IntentSender class. Then the reference of the file selected by the user is get by calling getFile ( ) method of the Drive API. File is then opened in the READ mode. Next, using the BufferedReader class object, the file is read and displayed on to the main screen.

- If the user has clicked on the download button then a new method called startIntentSenderForResult ( ) is called with an object of type IntentSender class. Then the reference of the file selected by the user is get by calling getFile ( ) method of the Drive API. File is opened in the WRITE mode and then using the FileOutputStream class object, file is first written with the user's code on the main screen and then it is closed.

Class and important methods implemented to achieve this functionality is mentioned below.

- **Class**: MainActivity
- **API**: Google Drive API
- **Methods:**

    - **void buildGoogleApiClient ( )** → Used to initialize google API client and is linked to the drive API. It is then used later when the Google Drive API's methods are called.
    - **IntentSender DriveApi.newOpenFileActivityBuilder()** → Method used to open Google drive interface on the application and returns an object of class IntentSender that stores the reference to the opened Drive.
    - **void startIntentSenderForResult (IntentSender, REQUEST_CODE_*)** → This method returns nothing and starts an internal activity that is used to perform further actions (read from a file or write to a file) in the Google drive depending upon the second parameter passed to it. If it is *REQUEST_CODE_SELECT*, that means users selects a file to read. If it is *REQUEST_CODE_SAVE*, that means user selects a file to write to it.
    - **DriveFile DriveApi.getFile (googleApiClient)** → This method is used to get the reference of the selected file by the user in the Google drive and it returns an object of DriveFile class.

User clicks the download button shown before in Figure 7 and then the steps shown below in Figure 11 are followed to load a file from the Google drive.

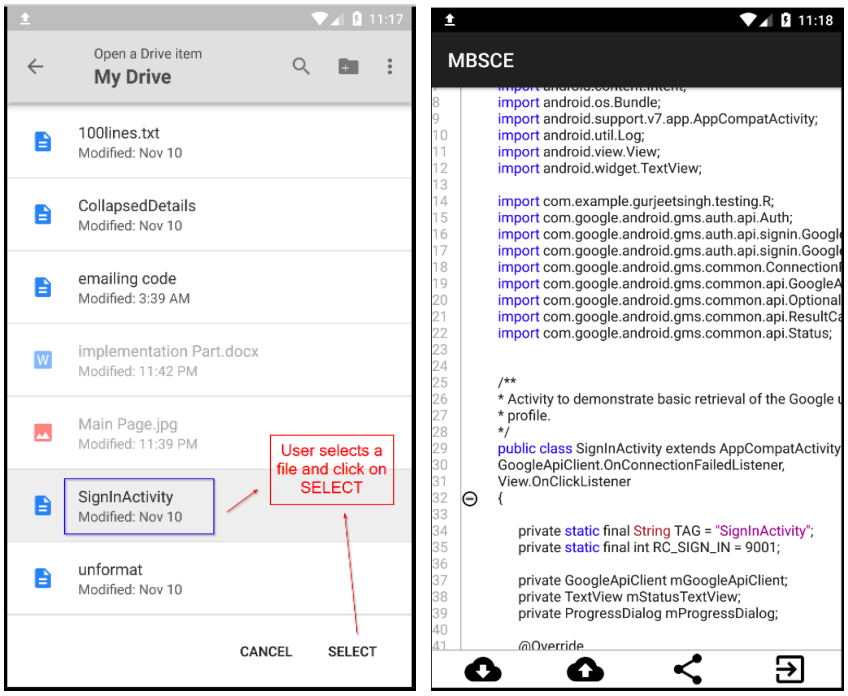

Figure 11: Accessing google drive

4.2.7 Ability to share the code over an Email

Once the user has done changes after accessing the source code from the cloud (Google Drive) or after the user has written the code from scratch, he/she can then share the code over an email.

This functionality is implemented by creating an Intent object which is initialized once the user clicks the share button on the main page as shown in Figure 8 before. This intent object is set with the default email header as “emailing code” and email body as the user’s source code. Next the startActivity ( ) method of the class Activity is called with the intent object as the parameter which opens an internal Gmail Activity using which the user can send an email. Modifications to email’s header and body can be done before sending an email as shown below in Figure 12.

- **Class:** MainActivity

- **Method(s):**
  - **void shareCode ( )** → This function returns nothing and is called once the user clicks on the share button on the main screen where he/she has edited the source code. This method initializes an Intent object that contains email header ("emailing code") and email body as user's code. It then calls the startActivity ( ) method.

  - **void startActivity(Intent.*createChooser*(intent, "Email "))** → This function return nothing and it operates on the Intent object that contains the email header and the email body. It triggers an internal Gmail activity which fills the header and the body using values in the intent object passed to this function.

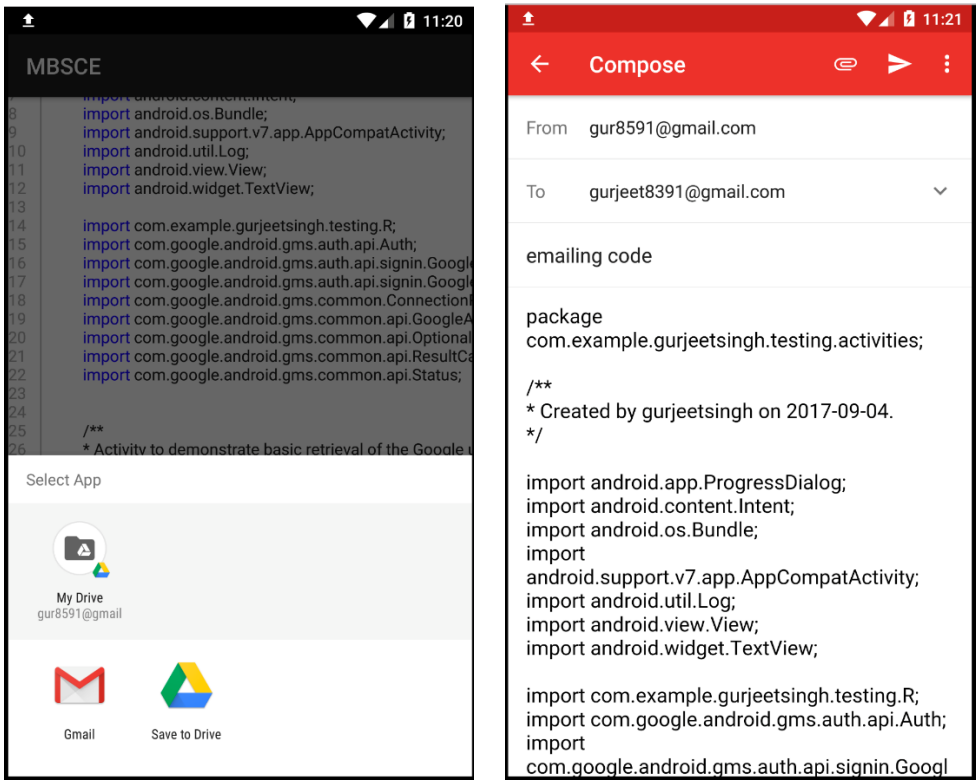

Figure 12: Sharing code over an email

4.2.8 Logging Out of the application

Users can successfully logout of the application after they are done with everything so that no one else can access their Google drive.

The way to implement this function is very straight forward. It makes use of the same object (**mGoogleApiClient)** created while implementing the Sign-In functionality. Once the user clicks the Sign out button, signOut ( ) method of the Google Sign in API (GoogleSignInApi) is called which logs the user out of the application.

Once this is done, the user is navigated to the Sign-In page again where he/she can Sign-in again and use the application. Shown in the Figure 8 is the sign out button that user clicks to sign out from the application and then the control is transferred to the sign-in page.

Class, API and functions used to implement this feature are mentioned below:

- **Class**: SignInActivity, GoogleApiClient
- **API**: Google Sign-in API (GoogleSingInAPI)

- **Method(s):**
    - **void exitApp (View v)** → This function is called when user click the Sign-out button. This methd in turn calls the signOut ( ) method of the Google Sign-in API (GoogleSignInApi) and takes the control back to the Sign-in page.

    - **void *GoogleSignInApi*.signOut(mGoogleApiClient)** → This method of the Google Sign-in API actually logs out the user. It is called with an object (**mGoogleApiClient)** of the class GoogleApiClient which was initialized during signing in to the application.

5 Limitations

Every application is developed considering the available resources, time and flexible user requirements, so it is not false to believe the fact that fulfilling all the requirements might cause your project to have some limitations.

This application does have some below listed limitations:

- Auto completion feature does not recommend complete syntaxes in terms of function or class names. It only inserts basic characters (} ,] , ” etc.)

- User can only view/edit one file at a time. It does not allow user to open multiple files in multiple tabs like other editors.

- It doesn’t have a FIND feature to locate a specific word in the source code. this can be very handy in case there is a lengthy source code with 1000 lines.

- Application works fine for 900-1000 lines of java source code but its performance decreases with more number of lines.

- Color feature colors all the classes in JDK version 7 but cannot color classes in the externally imported JAR files or classes in some other toolkit like Android SDK. Also, it doesn’t color constant values and method names.

6 Future Work

This section specifies the possible enhancements that can be done in future to increase the capabilities of this application.

- **Extend Auto Completion capability to make an Intelligent Editor:**

  By this feature the editor can even automatically recommend complete syntaxes to avoid complete writing of the whole statement by the user.

- **Compile and run the source Code:**

  This can be a great addition to this application where in a user can compile his code to see the errors in the code. Once all the errors have been removed, the user can run the code to see the output in a separate window on the phone itself. This can be implemented by hosting the application on the server and then the user's mobile can send data to and fro using REST protocols to the server hosting javac compiler and see the output on the screen.

7 Conclusion

This project has gone through various phases during the implementation of different features and finally a complete application has been built that allows its users to edit their source code in a flash by just using their mobile phones. Brief about each feature of this application, followed by application modelling and then the implementation details of each feature has been described in this report. Lastly, the limitations of this project have been discussed briefly followed by the possible enhancements that can be done as a part of future work.

8 Document Revision History

This section briefs all the documents created so far for this project along with other necessary details.

Document name	*Description*	*Date of Submission*	*Version*
Project Proposal	This document is the project proposal that specifies the problem statement with the solution	28-Aug-17	V1.1
Project Proposal	Document updated with limitations and some details in section 3.4	04-Sept-17	V1.2
Project Report	Detailed document that enlists the problem statement, design, implementation details, limitations of the project, possible future enhancements etc.	30-Oct-17	V1.1
Project Report	After review, changes have been done in the previous version of the report.	12-Nov-17	V1.2
Project Report	Changes suggested during the oral exam have been done.	11-Dec-17	V1.3

Table 2: Details about the project progress

9 References

[1] Android development framework
http://mrbool.com/android-development-framework/29690

[2] Android Studio - TutorialsPoint
http://www.tutorialspoint.com/android/android_studio.htm

[3] Code folding
https://en.wikipedia.org/wiki/Code_folding

[4] Spannable Interface to specify multiple colors in a text box
https://developer.android.com/reference/android/text/Spannable.html

[5] Indentation - Code Conventions for the Java Programming Language
http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-136091.html

[6] Google Developer - Authorization API implementation
https://developers.google.com/android/reference/com/google/android/gms/auth/api/Auth

[7] Google Developer Drive Access API and available methods
https://developers.google.com/android/reference/com/google/android/gms/drive/DriveApi.DriveContentsResult

[8] Java 7 API - Oracle Help Center
https://docs.oracle.com/javase/7/docs/api/

[9] Android studio in depth understanding
https://developer.android.com/studio/intro/index.html

[10] Start Integrating Google Sign-In into Your Android App
https://developers.google.com/identity/sign-in/android/start-integrating

[11] Java Language keywords
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

[12] The Activity Lifecycle
https://developer.android.com/guide/components/activities/activity-lifecycle.html