GRL2APK: User Guide
From Goal Models to Optimized Android Apps

Version 1.0 Dt: 23.10.2019

Contents

1 Architecture of the developed GRL2APK tool

We refer to the instantiation of the GRL2APK architecture shown in Figure-2 of the main paper. We implement the NFR Optimizer as a Web Service. The complete instantiated framework is shown in Figure-1. We briefly mention the technologies used for instantiating each component of the framework.

1. **Component Extractor:** Implemented in Acceleo. Takes a goal model specification and extracts its components - *Actors, Goals, Goal Decompositions,* and *Softgoals.*

2. **NFR Repository:** Implemented in Google Firebase. Contains the *NFR catalogs, NFR Contribution Functions,* and *NFR Conflict Functions.*

3. **NFR Selector:** Implemented as a Web Service in PHP. The developer provides inputs for *NFR operationalizations* and *NFR prioritizations.*

4. **NFR Optimizer:** Implemented as a Web Service in PHP. Computes the *Quality Function values* for all possible NFR solutions. Provides and interface to the developer to choose and finalize the NFR solution to be deployed.

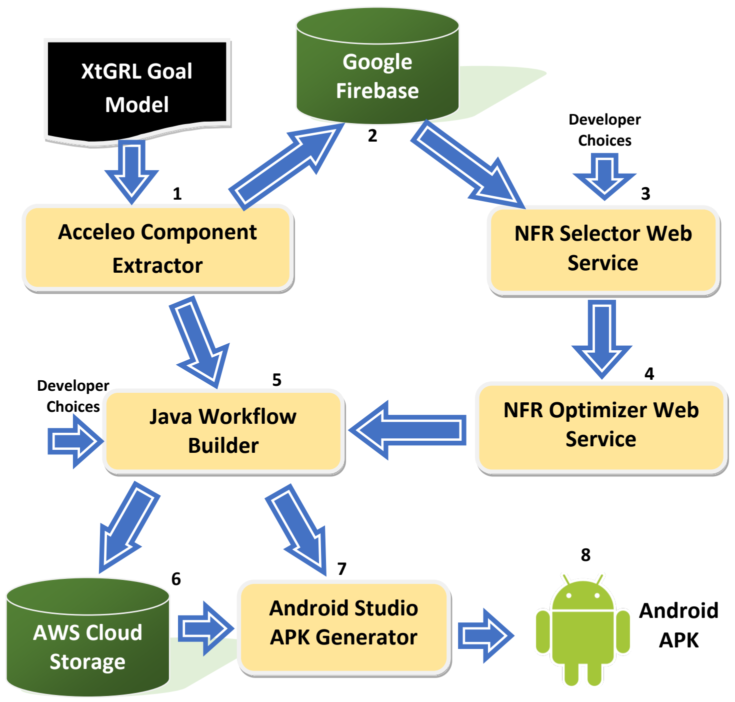


Figure 1: A complete instantiation of the GRL2APK Architecture.

5. **Workflow Builder:** Implemented in Java within the Eclipse framework. Allows the developer to choose *goal fulfilment strategies* and *code signatures* for building the application.

6. **Code Repository:** Implemented using Amazon Web Services (AWS). Contains the *Java code modules* for both goals and softgoals as well as *XML design interfaces* for the app.

7. **APK Generator:** Implemented using AndroidStudio. All the necessary Java and XML codes are integrated with respect to the workflow and compiled to build the app.

8. **Mobile App:** An *Android application* that can be installed and run on Android OS.

2 Step by Step Android App Generation from Goal Model Specification

We have evaluated our instantiated GRL2APK tool using a message transfer feature. In this use case, we have two actors in the goal model namely *Patient* and *Doctor*. The intention is to send medical records from patient's account to a doctor's profile. The record should be compressed and encrypted before sending through the channel for the sake of maintaining *bandwidth efficiency* and *confidentiality*. The goal model graphical representation is shown in Figure-2. In the following steps, we present how each module functions to build the android application from the goal model specification.

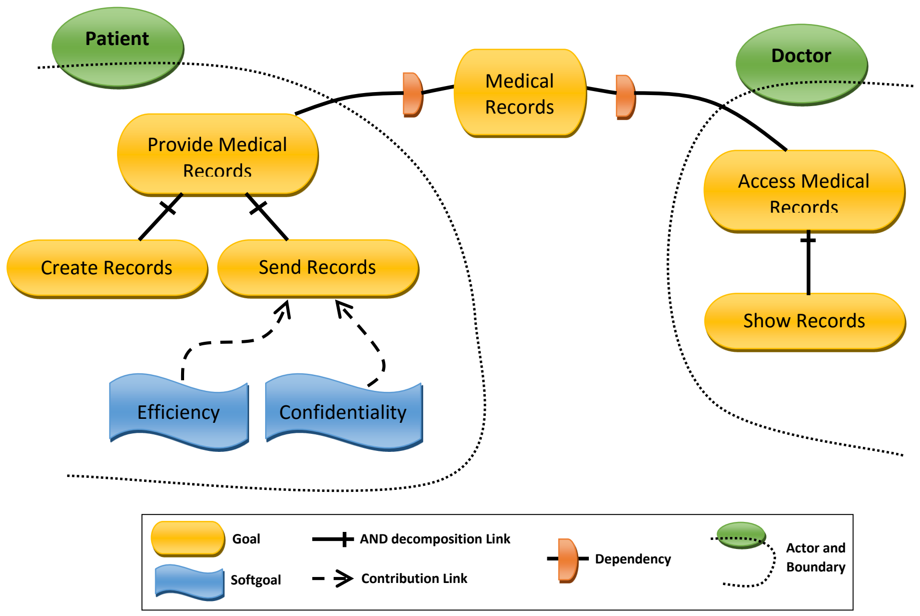


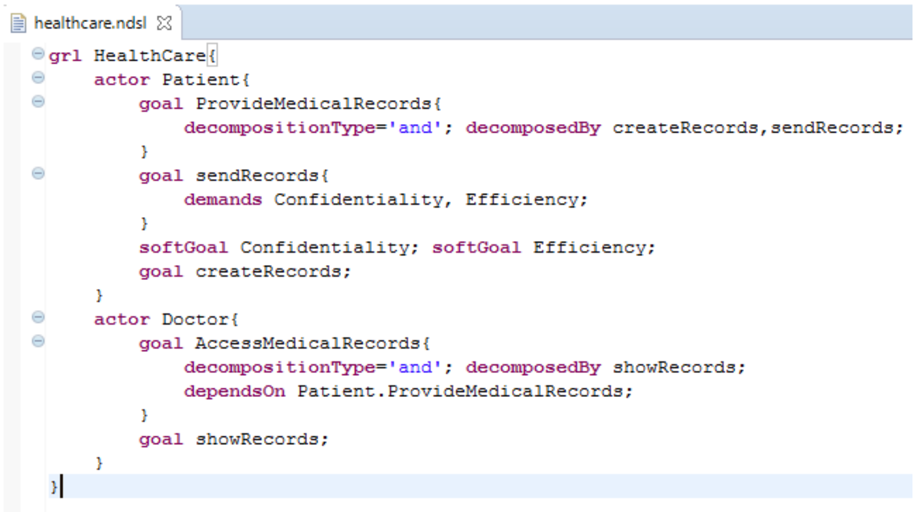
Figure 2: Goal Model Graphical View.

Step 1: Goal Model Specification

We define the XtGRL DSL grammar using XText (Uploaded in GitHub Repository[1]). The graphical goal model (shown in Figure-2) must be described using the XtGRL textual syntax. To ensure that the goal model description is free from syntactic errors, we write the goal model in the Eclipse runtime environment and save it with the *.ndsl* extension. For the above use-case, please refer to the *HealthCare.ndsl* file in the GitHub Repository[2]. A screenshot of the contents of this file is shown in Figure-3.

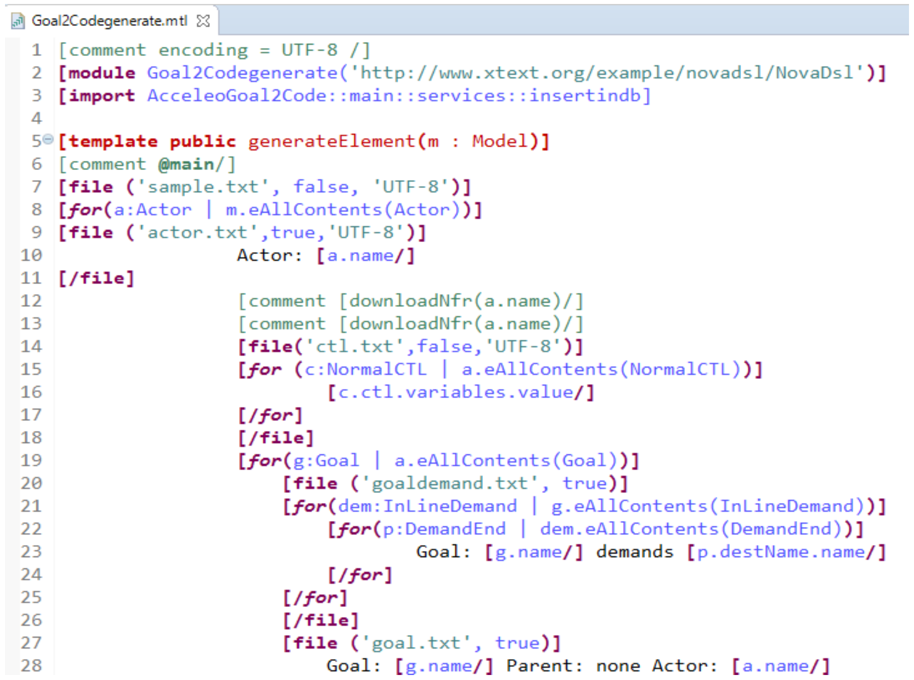
[1]The source of the DSL grammar is freely accessible in https://github.com/GRL2APK/XtGRL

[2]The sources of the tool are freely accessible in https://github.com/GRL2APK/GRL2APKTool



```
grl HealthCare{
    actor Patient{
        goal ProvideMedicalRecords{
            decompositionType='and'; decomposedBy createRecords,sendRecords;
        }
        goal sendRecords{
            demands Confidentiality, Efficiency;
        }
        softGoal Confidentiality; softGoal Efficiency;
        goal createRecords;
    }
    actor Doctor{
        goal AccessMedicalRecords{
            decompositionType='and'; decomposedBy showRecords;
            dependsOn Patient.ProvideMedicalRecords;
        }
        goal showRecords;
    }
}
```

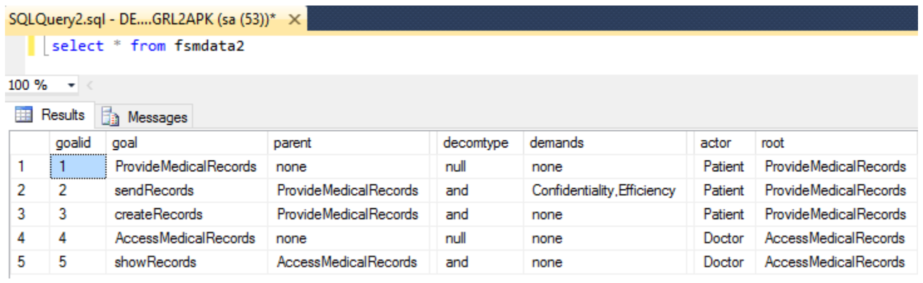
Figure 3: Goal Model Specification.



```
[comment encoding = UTF-8 /]
[module Goal2Codegenerate('http://www.xtext.org/example/novadsl/NovaDsl')]
[import AcceleoGoal2Code::main::services::insertindb]

[template public generateElement(m : Model)]
[comment @main/]
[file ('sample.txt', false, 'UTF-8')]
[for(a:Actor | m.eAllContents(Actor))]
[file ('actor.txt',true,'UTF-8')]
                Actor: [a.name/]
[/file]
                [comment [downloadNfr(a.name)/]
                [comment [downloadNfr(a.name)/]
                [file('ctl.txt',false,'UTF-8')]
                [for (c:NormalCTL | a.eAllContents(NormalCTL))]
                        [c.ctl.variables.value/]
                [/for]
                [/file]
                [for(g:Goal | a.eAllContents(Goal))]
                    [file ('goaldemand.txt', true)]
                    [for(dem:InLineDemand | g.eAllContents(InLineDemand))]
                        [for(p:DemandEnd | dem.eAllContents(DemandEnd))]
                                Goal: [g.name/] demands [p.destName.name/]
                        [/for]
                    [/for]
                    [/file]
                    [file ('goal.txt', true)]
                        Goal: [g.name/] Parent: none Actor: [a.name/]
```

Figure 4: Acceleo MTL for Component Extraction.


SQLQuery2.sql - DE....GRL2APK (sa (53))*

```
select * from fsmdata2
```

	goalid	goal	parent	decomtype	demands	actor	root
1	1	ProvideMedicalRecords	none	null	none	Patient	ProvideMedicalRecords
2	2	sendRecords	ProvideMedicalRecords	and	Confidentiality,Efficiency	Patient	ProvideMedicalRecords
3	3	createRecords	ProvideMedicalRecords	and	none	Patient	ProvideMedicalRecords
4	4	AccessMedicalRecords	none	null	none	Doctor	AccessMedicalRecords
5	5	showRecords	AccessMedicalRecords	and	none	Doctor	AccessMedicalRecords

Figure 5: Components are stored in Database.

Step 2: Goal Model Component Extraction

Figure-4 shows the *Acceleo Component Extractor* module of the GRL2APK architecture depicted in Figure-1.

- **Input:** Accepts the goal model specification file named *HealthCare.ndsl* written in XtGRL. This module also has access to the XtGRL DSL grammar specifications.

- **Process:** Execute the *GoalModelExtractor.mtl* file to extract the different components - goals, softgoals, goal decompositions, goal hierarchies, etc. A partial screenshot of the *GoalModelExtractor.mtl* file is shown in Figure-4.

- **Output:** Extracted components are stored in the SQL Server database. Figure-5 shows a partial screenshot of the database created for the goal model.

  The database has the following fields:

  1. ***goalid:*** Maintains the unique id for each goal.
  2. ***goal:*** Stores the name of the corresponding goal.
  3. ***parent:*** Holds the name of Parent of the a particular goal. The value "none" represents that the goal is the root goal of the actor.
  4. ***decomtype:*** Specifies the decomposition type of the goal. If it is the root goal of a particular actor, it holds "null" value for the record.
  5. ***demands:*** Stores the name of the demanding softgoal for the corresponding goal. The value "none" specifies that the goal does not demand any softgoal.
  6. ***actor:*** Maintains the name of the corresponding actor for a particular goal.

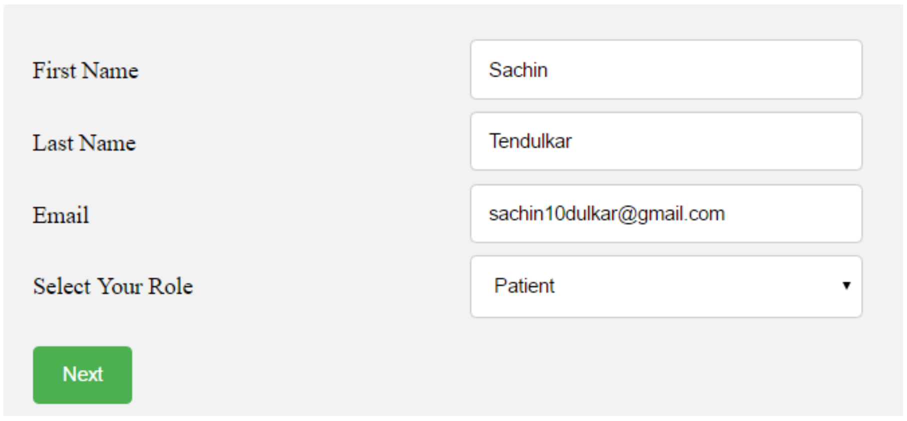
- **How to Access:** The *GoalModelExtractor.mtl* file available in the GitHub Repository[3].

[3]The sources of the tool are freely accessible in https://github.com/GRL2APK/GRL2APKTool

Step 3: NFR Operationalization and Priority Selection

(a) **Registration to Web Application:** This interface depicted in Figure-6 of the Web Application is a part of module 3, i.e., the *NFR Selector Web Service* of the GRL2APK architecture presented in Figure-1.

- **Process:** Register into the system by proving basic details like Name, Email ID etc. Click on "Next" button to store the information in MySQL server database.
- **Output:** The basic registration information is maintained in the MySQL server database.
- **How to Access:** After running the Wamp Server, in the web browser navigate to http://localhost/phpProject/index.php.



First Name	Sachin
Last Name	Tendulkar
Email	sachin10dulkar@gmail.com
Select Your Role	Patient

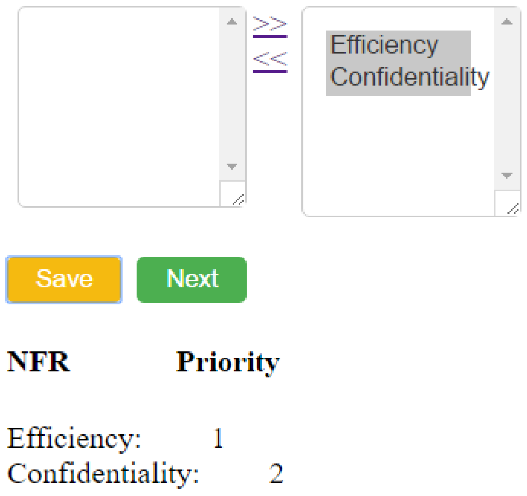
Figure 6: WebApp: Registration Page.

(b) **NFR Prioritization:** The NFR Prioritization interface presented in Figure-7 is a part of the module *NFR Selector Web Service* of the GRL2APK architecture in Figure-1

- **Input:** The SQL Server Database is accessed to extract the high-level NFRs. Previously, these high-level NFRs were specified in the goal model.
- **Process:** In this interface, select NFRs from a list of available NFRs from the left side panel and place them in the right side panel. As shown in Figure-7, The high-level NFR "Efficiency" is placed above "Confidentiality" in the right side panel; the highest priority value (value 1) is assigned to "Efficiency". Click the "Save" button and then click "Next" button to go to the next page.
- **Output:** The button *Save* will save the orderings of the high-level NFRs along with assigned priorities in the MySQL server database.

These priority values are also maintained along with the developer's registration information in the MySQL server database.

- **How to Access:** The previous registration page invokes the NFR Prioritization interface automatically.

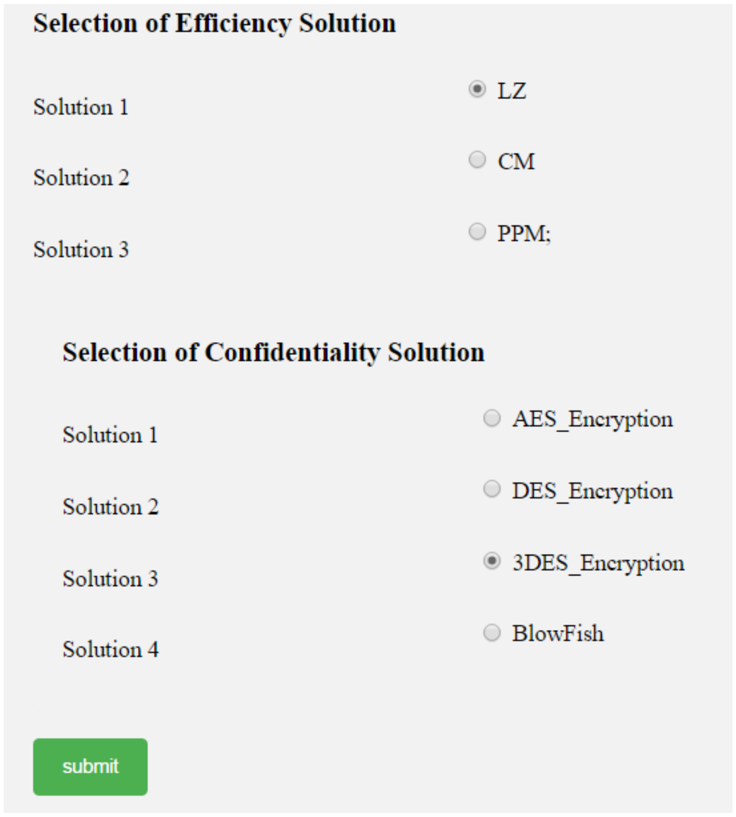


NFR	Priority
Efficiency:	1
Confidentiality:	2

Figure 7: WebApp: NFR Prioritization.

(c) **NFR Operationalization Selection:** This NFR operationalization selection interface presented in Figure-8 also represents another part of the module *NFR Selector Web Service* of the GRL2APK architecture depicted in Figure-1

- **Input:** The web application fetches NFR catalogs from the NFR Repository. NFR operationalizations are specified in the NFR catalogs. The web application fetches them and makes them available (for selection) in the interface. Figure-9 shows the Google Firebase Storage NFR Repository that contains the NFR catalogs. Figure-10 shows the NFR catalogs for *Confidentiality* and *Efficiency*.
- **Process:** Select one of the NFR operationalizations for each high-level NFR. Click on the "Submit" button to navigate to the next page. Figure-8 shows how "LZ" for *Efficiency* and "3DES_ -Encryption" for *Confidentiality* have been selected.
- **Output:** The selected operationalizations are also maintained in the MySQL server database along with developer's registration information.
- **How to Access:** The previous page i.e. *NFR Prioritization* leads the control to this interface in sequence.



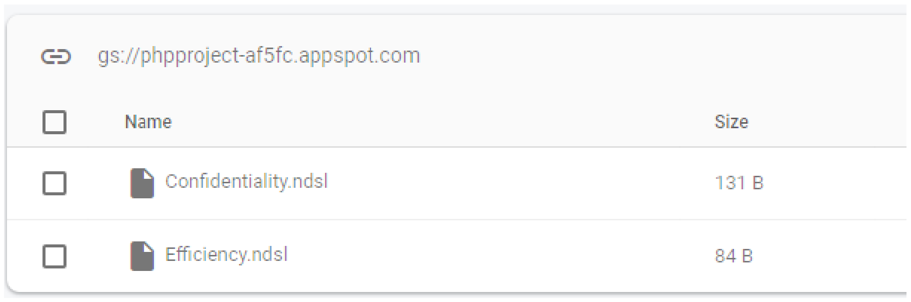
Selection of Efficiency Solution

Solution 1	[✓ LZ]
Solution 2	CM
Solution 3	PPM;

Selection of Confidentiality Solution

Solution 1	AES_Encryption
Solution 2	DES_Encryption
Solution 3	[✓ 3DES_Encryption]
Solution 4	BlowFish

Figure 8: WebApp: NFR operationalization Selection.



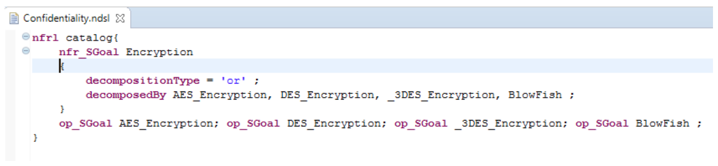
gs://phpproject-af5fc.appspot.com

Name	Size
Confidentiality.ndsl	131 B
Efficiency.ndsl	84 B

Figure 9: Google Firebase NFR Catalog Repository

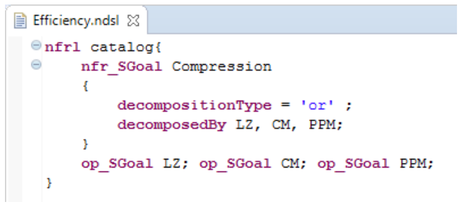
Step 4: NFR Optimization

Figure-11 shows the interface for the NFR Optimizer. This interface corresponds to the *NFR Optimizer Web Service* module of the GRL2APK architecture pre-



Confidentiality.ndsl

```
nfrl catalog{
    nfr_SGoal Encryption
    {
        decompositionType = 'or' ;
        decomposedBy AES_Encryption, DES_Encryption, _3DES_Encryption, BlowFish ;
    }
    op_SGoal AES_Encryption; op_SGoal DES_Encryption; op_SGoal _3DES_Encryption; op_SGoal BlowFish ;
}
```



Efficiency.ndsl

```
nfrl catalog{
    nfr_SGoal Compression
    {
        decompositionType = 'or' ;
        decomposedBy LZ, CM, PPM;
    }
    op_SGoal LZ; op_SGoal CM; op_SGoal PPM;
}
```

Figure 10: NFR Catalogs for Confidentiality and Efficiency

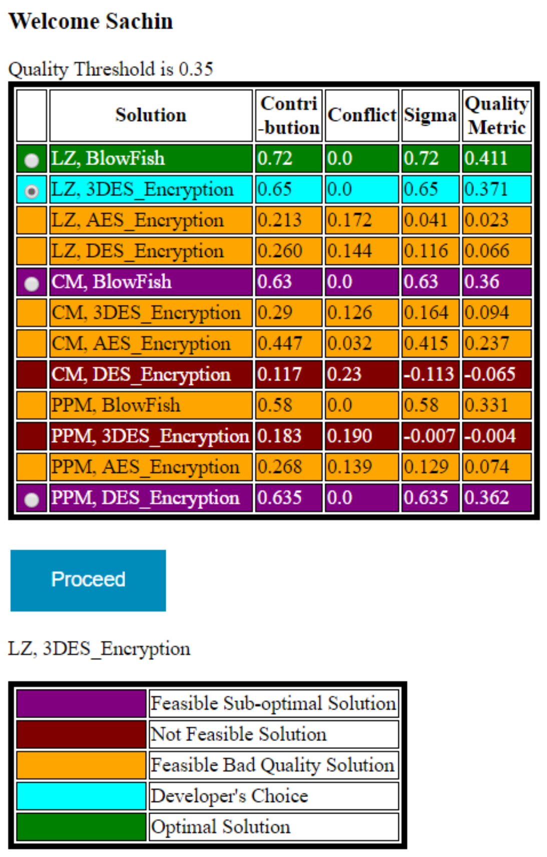
sented in Figure-1.

- **Input:** It fetches all the information of NFR operationalization chosen in the last phase and priorities from the MySQL server database. Other metric like NFR operationalization contribution values are also accessed from the Google Firebase NFR Repository.
- **Process:** Your chosen solution gets highlighted as default solution. Click on other radio button to change the selection of solution. For example, in the previous interface, the combined solution of "LZ" Operationalization and "3DES_Encryption" operationalization have been selected which is highlighted in this interface.
- **Output:** Click "Proceed" button to store selected solution in a text file named *SelectedNFROp.txt* in the current project directory.
- **How to Access:** After submitting the selected NFR operationalizations from the previous interface, the page redirects the control to NFR Optimization Interface.

Step 5: Workflow Building

(a) Figure-12a shows the first interface as the part of the *Java WorkFlow Builder* module of the GRL2APK architecture depicted in Figure-1.

- **Input:** The *SQL Server* Database is accessed to extract the components that were specified in the goal model.
- **Process:** Choose the *Actor*, and one root goal at a time in the present interface. Click on "Proceed" button to bring up the next interface. For example, under the actor *Patient*, the root goal "ProvideMedicalRecords" has been selected.



Welcome Sachin

Quality Threshold is 0.35

	Solution	Contri -bution	Conflict	Sigma	Quality Metric
○	LZ, BlowFish	0.72	0.0	0.72	0.411
◉	LZ, 3DES_Encryption	0.65	0.0	0.65	0.371
	LZ, AES_Encryption	0.213	0.172	0.041	0.023
	LZ, DES_Encryption	0.260	0.144	0.116	0.066
○	CM, BlowFish	0.63	0.0	0.63	0.36
	CM, 3DES_Encryption	0.29	0.126	0.164	0.094
	CM, AES_Encryption	0.447	0.032	0.415	0.237
	CM, DES_Encryption	0.117	0.23	-0.113	-0.065
	PPM, BlowFish	0.58	0.0	0.58	0.331
	PPM, 3DES_Encryption	0.183	0.190	-0.007	-0.004
	PPM, AES_Encryption	0.268	0.139	0.129	0.074
○	PPM, DES_Encryption	0.635	0.0	0.635	0.362

LZ, 3DES_Encryption

🟪	Feasible Sub-optimal Solution
🟥	Not Feasible Solution
🟧	Feasible Bad Quality Solution
🟦	Developer's Choice
🟩	Optimal Solution

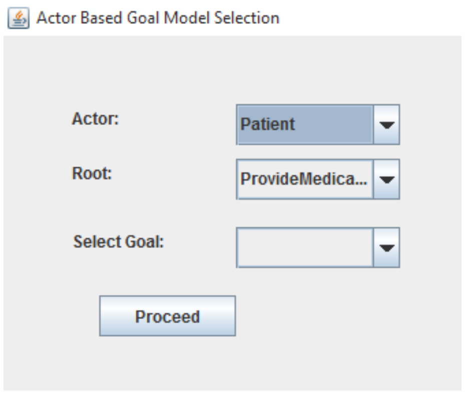
Figure 11: WebApp: NFR Optimization Solutions.

- **Output:** It leads to another interface that provides goal tree representation, through which the workflow of the system can be specified.
- **How to Access:** After completion of the NFR operationalization selection, switch back to *workflow building* section under *Tool Folder* in the GitHub Repository[4]. A file named *GoalModelCaller.java* produces the first interface for the workflow builder.

(b) Figure-12b shows the interface for the goal tree. This interface is also a part of the module *Java WorkFlow Builder* of the GRL2APK architecture presented in Figure-1.

- **Input:** The root goal of a particular actor is provided by the previous interface. The tree structure against the root goal is fetched from the

[4]The sources of the tool are freely accessible in https://github.com/GRL2APK/GRL2APKTool

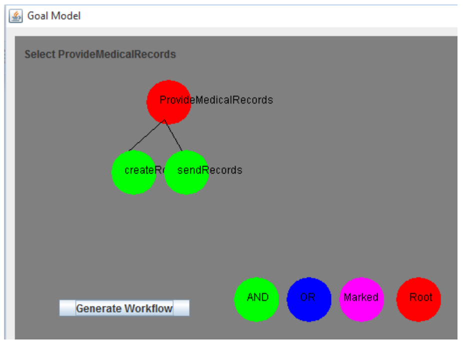

Actor Based Goal Model Selection

Actor: Patient

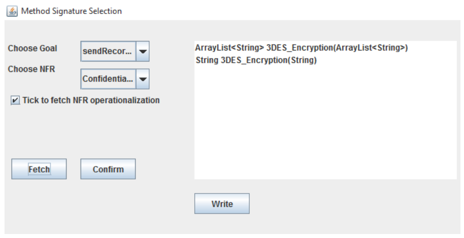
Root: ProvideMedica...

Select Goal:

(a) Workflow Building.



(b) Goal Tree Representation to Specify Workflow.


Method Signature Selection

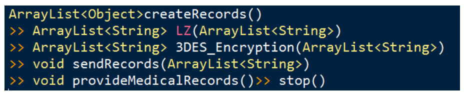
Choose Goal: sendRecor...

Choose NFR: Confidentia...

[✓] Tick to fetch NFR operationalization

```
ArrayList<String> 3DES_Encryption(ArrayList<String>)
String 3DES_Encryption(String)
```

(c) Code Signature Selection.


```
ArrayList<Object>createRecords()
>> ArrayList<String> LZ(ArrayList<String>)
>> ArrayList<String> 3DES_Encryption(ArrayList<String>)
>> void sendRecords(ArrayList<String>)
>> void provideMedicalRecords()>> stop()
```

(d) Generated Workflow.

Figure 12: Workflow Builder Interface

SQL Server database where all the information of the goal model is stored.

- **Process:** Follow the instructions on the screen and select goals and sub-goals according to the decomposition type. For example, for "OR" decomposed sub-goals, once you choose a goal, the other goals and goal trees will be deactivated and removed. For "AND" decomposed sub-goals, you must specify the order in which the sub-goals will be executed. After selecting all the goals click on "Generate Workflow".
- **Output:** The workflow against the goal selection is written in a temporary file which is used in the next interface. "Generate Workflow" button invokes another interface where code signature against each goal is to be selected.
- **How to Access:** The previous interface automatically opens up the interface after clicking the "Proceed" button in that interface.

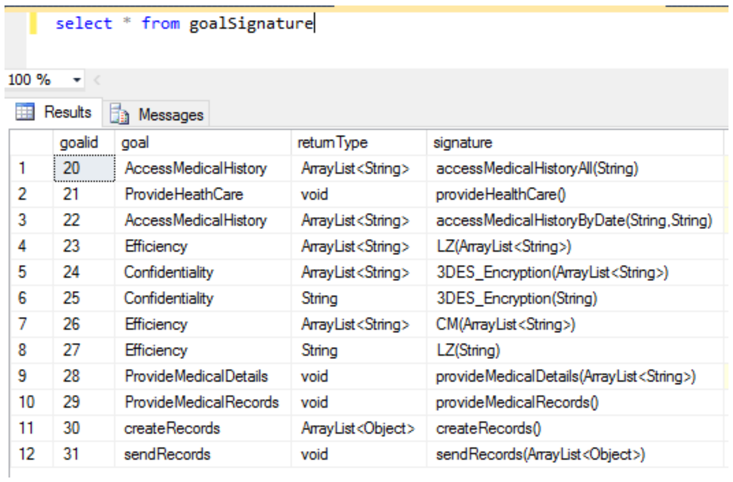
(c) Figure-12c shows the interface representing a part of the *Java Workflow Builder* module of the GRL2APK architecture presented in Figure-1.

- **Input:** Workflow temporary file is provided by the previous interface.

Code signatures are fetched from the SQL Server database as shown in Figure-13 and the mapping between AWS S3 Code repository and the code signature database is also maintained. *SelectedNFROp.txt* file is accessed to extract the selected NFR operationalizations.

- **Process:**
  (a) First, fetch code signatures (by clicking "Fetch" Button) for goals (without selecting the "CheckBox") and then for NFRs (by selecting the "Checkbox").
  (b) In Figure-12c, two method signatures for the NFR operationalization "3DES_Encryption" are shown on the right side panel. Select one of the signatures and click on "Confirm" button.
  (c) After confirming all the signatures for every goal and NFR, click on "Write" button to write the method signatures, according to the workflow, in a file.
- **Output:** A file named *Workflow.wf* depicted in Figure-12d is generated in the current project directory. This file will be accessed by the *APK Generator* module to understand the workflow of the system.
- **How to Access:** The previous goal tree interface brings up the interface automatically after clicking the "Generate Workflow" button.

The entire process needs to be repeated if there exist multiple root goals for each actor.


```
select * from goalSignature
```

	goalid	goal	returnType	signature
1	20	AccessMedicalHistory	ArrayList<String>	accessMedicalHistoryAll(String)
2	21	ProvideHeathCare	void	provideHealthCare()
3	22	AccessMedicalHistory	ArrayList<String>	accessMedicalHistoryByDate(String,String)
4	23	Efficiency	ArrayList<String>	LZ(ArrayList<String>)
5	24	Confidentiality	ArrayList<String>	3DES_Encryption(ArrayList<String>)
6	25	Confidentiality	String	3DES_Encryption(String)
7	26	Efficiency	ArrayList<String>	CM(ArrayList<String>)
8	27	Efficiency	String	LZ(String)
9	28	ProvideMedicalDetails	void	provideMedicalDetails(ArrayList<String>)
10	29	ProvideMedicalRecords	void	provideMedicalRecords()
11	30	createRecords	ArrayList<Object>	createRecords()
12	31	sendRecords	void	sendRecords(ArrayList<Object>)

Figure 13: Code Signatures Maintained in Database.

Step 6: APK Generation

This step represents the *Android Studio APK Generator* module in the GRL2APK architecture depicted in Figure-1. Figure-14 shows the AWS code repository. This is corresponding to the *AWS Cloud Storage* repository depicted in Figure-1.

- **Input:** The workflow file (*Workflow.wf*) generated by *Workflow Builder* module, *AWS Code Repository* and the code signature database are accessed.

- **Process:**

  1. Execute the *CodeIntegrator.java* file which will process the *Workflow.wf* file and download the corresponding codes for each method signature specified in the file from the *AWS Code Repository*. *AWS Code Repository* maintains different code segments in different folder buckets. Those folder buckets also contains the design XML if required to design an interface or only code segments for implementing a java function.
  2. In the next step, execute another java file called *AndroidPackageCreator.java* to create the android project by integrating the downloaded

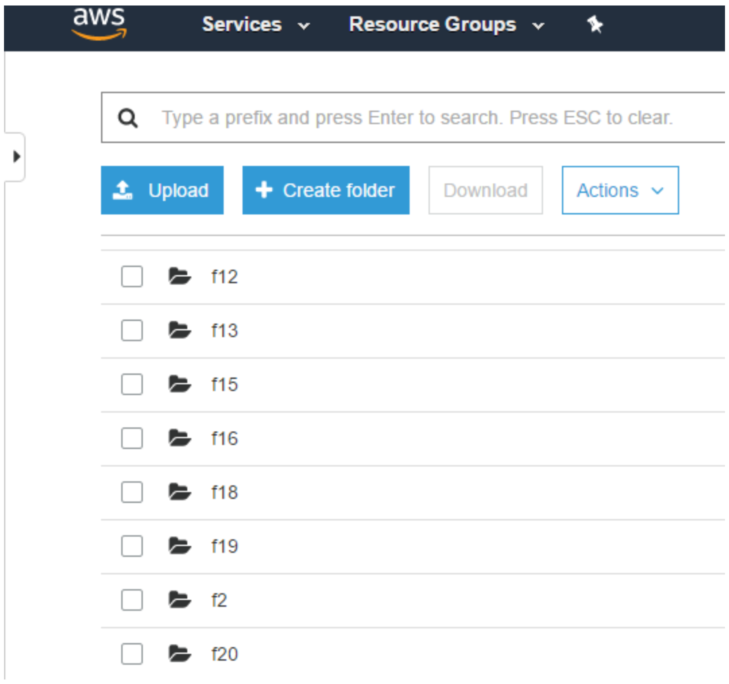
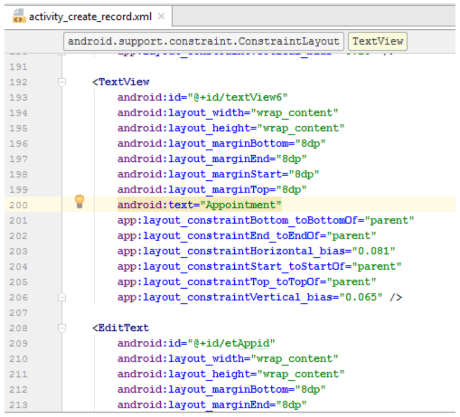


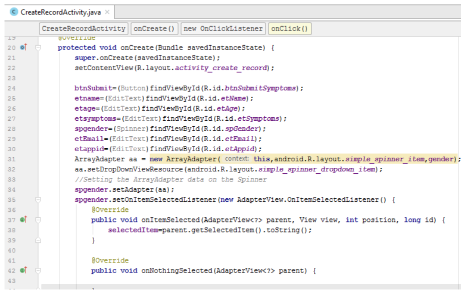
Figure 14: AWS Code Repository.


activity_create_record.xml
android.support.constraint.ConstraintLayout
TextView

```xml
    <TextView
        android:id="@+id/textView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Appointment"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.081"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.065" />

    <EditText
        android:id="@+id/etAppid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
```

(a) Partial XML Code for Interface Design.

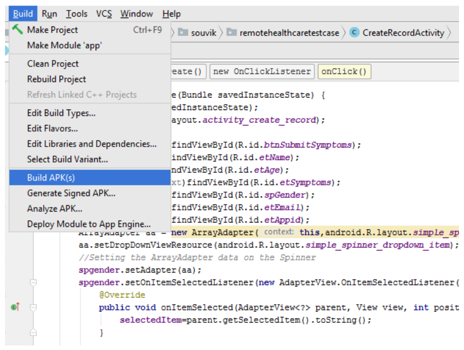


```java
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_record);

        btnSubmit=(Button)findViewById(R.id.btnSubmitSymptoms);
        etname=(EditText)findViewById(R.id.etName);
        etage=(EditText)findViewById(R.id.etAge);
        etsymptoms=(EditText)findViewById(R.id.etSymptoms);
        spgender=(Spinner)findViewById(R.id.spGender);
        etEmail=(EditText)findViewById(R.id.etEmail);
        etappid=(EditText)findViewById(R.id.etAppid);
        ArrayAdapter aa = new ArrayAdapter( context: this,android.R.layout.simple_spinner_item,gender);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        //Setting the ArrayAdapter data on the Spinner
        spgender.setAdapter(aa);
        spgender.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
                selectedItem=parent.getSelectedItem().toString();
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {
```

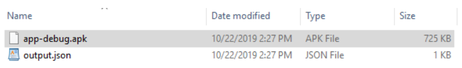
(b) Partial Java Code for Android Application.



```java
e(Bundle savedInstanceState) {
edInstanceState);
ayout.activity_create_record);

findViewById(R.id.btnSubmitSymptoms);
indViewById(R.id.etName);
ndViewById(R.id.etAge);
xt)findViewById(R.id.etSymptoms);
findViewById(R.id.spGender);
findViewById(R.id.etEmail);
findViewById(R.id.etAppid);
        new ArrayAdapter( context: this,android.R.layout.simple_sp
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        //Setting the ArrayAdapter data on the Spinner
        spgender.setAdapter(aa);
        spgender.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener(
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int posit
                selectedItem=parent.getSelectedItem().toString();
            }
```

(c) Generate APK option in Android Studio.



Name	Date modified	Type	Size
app-debug.apk	10/22/2019 2:27 PM	APK File	725 KB
output.json	10/22/2019 2:27 PM	JSON File	1 KB

(d) Generate APK option in Android Studio.

Figure 15: Screen Shots in Android Studio

code snippets. All the supporting files like *AndroidManifest.xml*, *build.gradle* files will be created automatically as it uses the command line android sdk.

3. The next step, run the *AndroidManifestModifier.java* to modify *AndroidManifest.xml*. Modification involves making entries of different activities created by the package manager.

4. Finally, Open the *Android Studio* and export the newly created project in the Android Studio's default project directory. Compile and build the project to create the .apk file for the android application. Figure-15a shows the partial XML code for an application interface opened in Android Studio. Figure-15b shows the partial java code to implement an activity for android application. Figure-15c present the option in Android Studio through which we can generate .apk file for the android platform. Finally, Figure-15d

shows the generated .apk application file in the default directory i.e. *Android_Studio_Project_Directory* $\backslash ProjectName\backslash app\backslash build\backslash outputs\backslash apk\backslash debug$

- **How to Access:** In the GitHub[5]project, a separate folder named *APK Generator* contains all the necessary files mentioned in the above steps.

3 Generated Android Application Usage

1. **Home Screen** The home screen has two buttons for two different actors namely *Patient* and *Doctor*. For simplicity any actor can enter to their corresponding home screens by clicking the buttons. The buttons are the representations of the actors specified in the goal model. Figure-16a is the typical screen for the home screen.

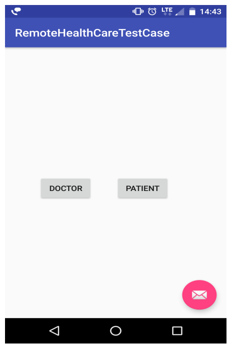
2. **Patient Home Screen** The patient home screen contains a button according to the name of the root goal of the actor *Patient*. The root goal *ProvideMedicalRecords* is represented as the button in this screen. Clicking the button leads the control to the next screen for record creation. Figure-16b shows the screen for the patient's home screen.

3. **Medical Records Creation Screen** In this screen the actor patient can provide the basic details and the problem symptoms as well. This screen is the corresponding to the subgoal *createMedicalRecords*. The *Submit* button in the screen calls a method *sendRecords()* which corresponds to the goal *sendRecords* and is responsible for storing the compressed and encrypted record in the Google Firebase cloud database. The Figure-16c presents the screen for the medical record submission for the patient.

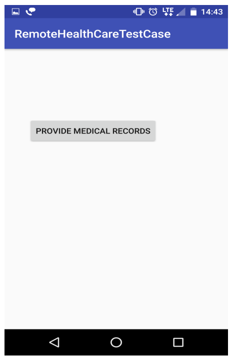
4. **Doctor Home Screen** The doctor home screen is represented in Figure-16d. Doctor can see several medical records of patients and he/she can choose any of them by clicking the radio button. This screen is corresponding to the *AccessMedicalRecords* goal of the goal model. *Show Record* button leads the control to another screen where doctor can view the record in detail. Figure-16d show the screen where doctor can access the appointments.

5. **Viewing Medical Records Screen** After selecting a particular medical record, the doctor can access the detailed record in the next screen corresponding to the *showRecords* goal in the goal model. The Figure-16e shows the detailed medical records of a patient at the doctor's end.

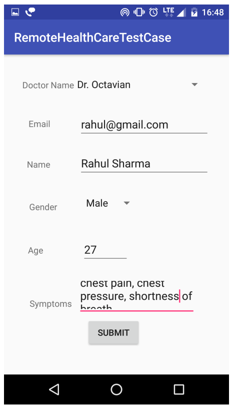
[5]The sources of the tool are freely accessible in https://github.com/GRL2APK/GRL2APKTool



(a) Home Screen



(b) Patient's Home.



Doctor Name Dr. Octavian

Email rahul@gmail.com

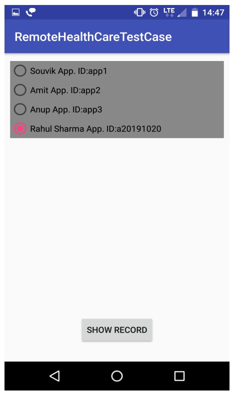
Name Rahul Sharma

Gender Male

Age 27

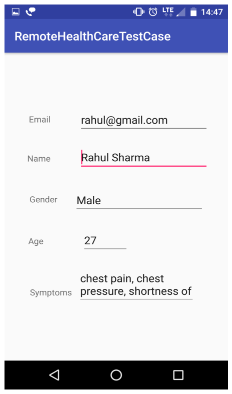
Symptoms chest pain, chest pressure, shortness of

(c) Medical Record Submission Screen.



○ Souvik App. ID:app1
○ Amit App. ID:app2
○ Anup App. ID:app3
● Rahul Sharma App. ID:a20191020

(d) Doctor's Screen to See Appointment List.



Email rahul@gmail.com

Name Rahul Sharma

Gender Male

Age 27

Symptoms chest pain, chest pressure, shortness of

(e) Doctor's Screen to See Patient's Medical Record.

Figure 16: Generated App Screens

4 Software and Hardware Requirements

4.1 Hardware Requirements

1. **Processor: Minimum Requirements-** Intel Core i3 or equivalent. **Recommended Requirements-** Intel Core i5 or higher.

2. **RAM: Minimum Requirements-** 4GB DDR3. **Recommended Requirements-** 6GB DDR3 or Higher.

3. **Disk Space: Minimum Requirements-** 5GB. **Recommended Requirements-** 10GB.

4.2 Software Requirements

1. **Operating System:** Microsoft Windows 7, 8, 10.

2. **Java:** jdk-1.7 or higher version.

3. **Eclipse** Eclipse Oxygen or higher version.

4. **Acceleo:** Installation of Acceleo plugin is required to run the *.mtl* files from Eclipse IDE.

   - **Installation:** Open *Eclipse IDE* Go to *Help* menu -> *Install New Software* -> Type *Acceleo* in search-bar.
     update also available at http://download.eclipse.org/acceleo/updates/releases /3.7.

5. **XText Plugin:** Installation of XText SDK plugin is required to execute Acceleo MTL files.

   - **Installation:** Open *Eclipse IDE* Go to *Help* menu -> *Install New Software* -> Type *Xtext* in search-bar.

6. **SQL Server:** SQL Server 2014 or Latest.

7. **Web Server:** WampServer Version 3.1.0 or XampServer Version 1.8.0.

8. **Android SDK:** Android Sdk Version 20 or Latest.

9. **Android Studio:** Android Studio Version 3.2 or Latest.
   https://developer.android.com/studio

10. **Google Firebase:** https://firebase.google.com

11. **Amazon Web Service S3:** https://aws.amazon.com