

LINKR

A
Project Report
Submitted for partial fulfillment
of B. Tech. Degree
In
Computer Science and Engineering

by
Aakanksha Verma [1405210001]
Abhishek Gupta [1405210004]
Himanshu Yadav [1405251022]

Under the supervision of
Prof. M.H. Khan and Mr. Ritendra Goyal



Department of Computer Science and Engineering
Institute of Engineering and Technology
Dr. A.P.J. Abdul Kalam Technical University,
Lucknow, Uttar Pradesh
8 June 2018

CERTIFICATE

This is to certify that the project report entitled **LINKR** presented by Aakanksha Verma(1405210001), Abhishek Gupta(1405210004) and Himanshu Yadav(1405251022) in the partial fulfillment for the award of Bachelor of Technology in Computer Science and Engineering, is a record of bona-fide work carried out by them under my supervision and guidance at the department of Computer Science and Engineering at Institute of Engineering and Technology, Lucknow.

(Prof. M.H. Khan)

Department of Computer Science and Engineering

Institute of Engineering and Technology, Lucknow

(Mr. Ritendra Goyal)

Department of Computer Science and Engineering

Institute of Engineering and Technology, Lucknow

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the project report undertaken during final year of **Bachelor of Technology** at **Institute of Engineering and Technology, Lucknow**. I owe special debt of gratitude to our project guides, Prof. M.H. Khan and Mr. Ritendra Goyal and the project coordinator, **Prof. Y. N. Singh**, for their constant support and guidance throughout the course of our work. Their sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only their cognizant efforts that our endeavour has seen light of the day.

I would not like to miss the opportunity to acknowledge the contribution of **Prof. S.P. Tripathi**, Head of Computer Science and Engineering department for his kind assistance and cooperation during the development of our project.

DECLARATION

We hereby declare that this project entitled **Linkr** submitted by us, in the partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering**, is a record of confide work carried out by us under the supervision and guidance of Prof. **M.H. Khan and Mr. Ritendra Goyal**, at the department of **Computer Science and Engineering** at **Institute of Engineering and Technology, Lucknow**.

Date: 08/06/2018

Aakanksha Verma(1405210001)

Abhishek Gupta(1405210004)

Himanshu Yadav(1405251022)

ABSTRACT

To stay linked socially we need to communicate with each other be it face-to-face conversation, video conferencing, telephone conversation, e-mails, handheld devices, blogs, written letters and memos, formal written documents or spreadsheets. For carrying out these communications successfully, we require a valid address or handle or some other identification such as residential address, phone number, IM handles, e-mail address, and many other such information, in other words the contact information of the people.

Whenever there is a change in one or more contact information, keeping everyone updated about the changes is a tedious task. Also, it may happen so that one may lose all the contact information of their acquaintances due to many reasons.

To tackle these issues we can store all the contact information of different people at a single place which is secure, consistent and up-to-date throughout the various dynamics of the people.

Linkr is a dynamic android application that is aimed at addressing the above issue with minimum complexity and memory and execution overhead.

Contents

INTRODUCTION	8
Overview of the application	8
Objective	8
Technology Used (Technology Stack)	8
Tools Required	8
Modules:	8
What all it does?	8
What does Dynamic mean?	9
1.4 More in less	9
1.5 Scope of the project	10
1.6 Features	10
1.7 Need	11
2. The Building Blocks	12
How does it work?	17
RecyclerView components	19
Adapter	21
View holder	22
Add the dependency to app/build.gradle	22
Add a RecyclerView to your activity's layout	23
Create the layout for one item	23
Create an adapter with a view holder	23
Implement the view holder class	24
Create the RecyclerView	25
Adding a user interface	29
Creating a layout	29
3. Installation guide	30
4. Android Studio	31

Project structure	31
Tool windows	34
Code completion	34
Find sample code	35
Navigation	35
Style and formatting	36
Version control basics	36
Gradle build system	36
Build variants	37
Multiple APK support	37
Resource shrinking	37
Managing dependencies	37
Debug and profile tools	37
Inline debugging	37
Performance profilers	38
Heap dump	38
Memory Profiler	38
Data file access	38
Code inspections	38
Annotations in Android Studio	39
5. Walkthrough	40
6. Low and High Level Design	57
8. Workflow	60
9. Database Hierarchy	61
10. Codes :	65
11. Conclusion	88
12. References	89

1. INTRODUCTION

1.1 Overview of the application

Objective

“To manage different types of contact details and other personal details of people and update the details of users in real-time along with custom privacy settings as well as providing the inbuilt chatting feature for the users.”

Technology Used (Technology Stack)

- Java (as programming language)
- XML (as programming language)
- Firebase (for real-time database and backend)

Tools Required

Android studio (for programming and debugging)
Adobe Photoshop (for interface and logo design)

Modules:

Profile Building: This module will enable the user to set up his profile by filling up details such as his name, his contact and address information, and hyperlinks to his social media accounts if any.

Interconnection: This module will connect the users of this application who wants to share the contact profile data with another . A contact profile can be modified and updated any number of times and the changes made by the user in his/her profile are automatically updated to all the users with whom his profile is shared.

Privacy: Two connected users need not share every contact information with each other. Every user can customize with whom he/she wants to share his particular contact details and with whom not.

Chat: The connected users can send text messages to each other using this application

1.2 What all it does?

- **Store contact details** like name, multiple numbers, address information, set a profile picture, etc.
- **Link various social networking sites with the application.**
- **Providing real time database connectivity.**
- **Chat**
- **Customized privacy settings**
- **Easy to understand and use interface**

1.3 What does Dynamic mean?

a. Static v/s Dynamic:

Static applications are not reliant on connection to an online server or database. Apps built in this way are downloaded once, usually periodically updated, and are able to function offline only with the device on which they are installed.

Dynamic applications are in some way reliant on an online server or database. When connected, these apps are loaded from a central server so that any iterative changes to development, design or functionality are rolled out across all devices simultaneously.

b. Why is dynamic application building approach used in Linkr ?

Although development costs for **dynamic** applications are slightly higher, they solve the problem of the rigid approach to product updates. A dynamic application can be optimized, fixed and tweaked across all devices. Additionally, dynamic apps are more likely to return insightful user data through engagement analytics. This allows you to build upon your UX design to deliver a product superior to the competition.

With dynamic applications, it is easy to bring new features online as development objects are not restricted by predetermined variables. New features can be added often without the need for software update, and when an update is required, it can be mandatory across all devices.

Another great benefit is that dynamic apps provide a platform for **user interaction**. Allowing users to interact not online with the software, but also with other users, can enrich your product exponentially. 21st-century tech is all about being social, so why should mobile applications be any different?

Finally, the basic requirement for a dynamic app is anything that is reliant on current data. In its most basic form, an app offering a service like Linkr, which is reliant on up-to-date information. Since, Linkr relies on a fresh stream of updates, dynamic is the only choice.

1.4 More in less

The application provides the services of a complete, reliable and secure contact information manager for all kinds of contact details along with an inbuilt chat feature and customized privacy settings.

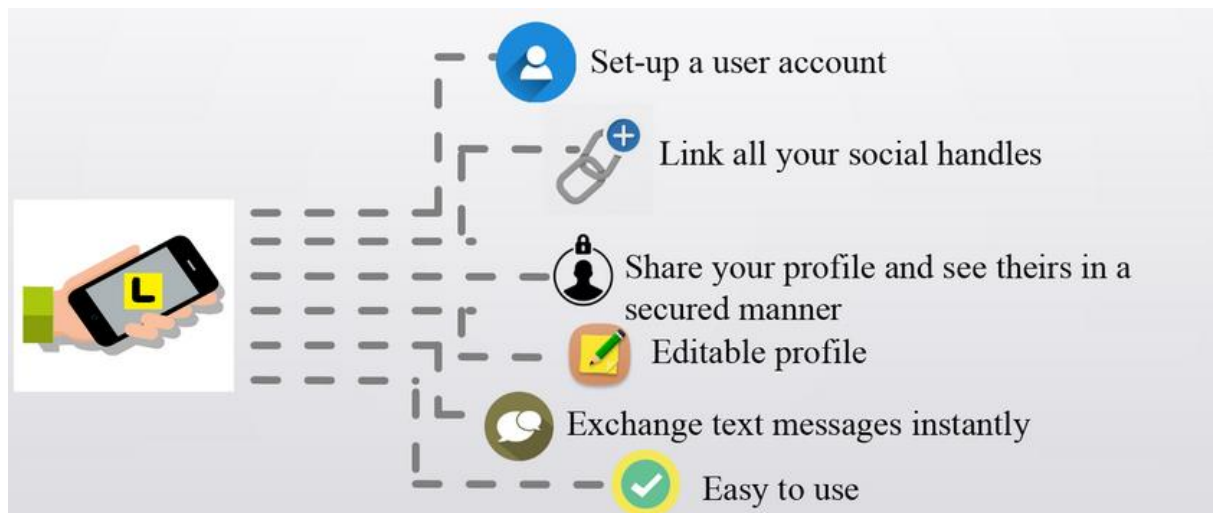
Despite numerous functions to perform Linkr, at the same time, ensures **less** memory requirement for the application storage, **less** RAM requirement, **less** CPU time demand and **less** internet bandwidth requirement to run the application.

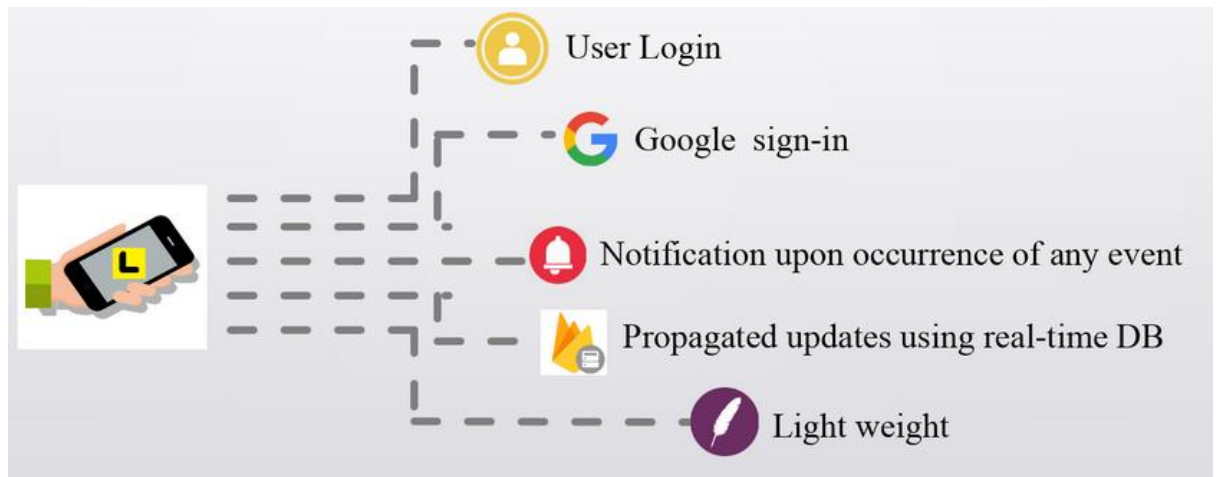
1.5 Scope of the project

The basic scope covers :

- Staying in touch with the friends and family.
 - Having latest contact information about them.
 - Sharing any updates in our contact details quickly and without much effort.
 - Easily handle all the social accounts from one place.
- ❖ Can be used for publicity of anything from a new shop to a new restaurant or a start-up needing to spread its contact details.
- ❖ Exclusive groups pertaining to a particular organization can be formed making use of this application, and share their contact details for smooth communication among themselves.

1.6 Features





1.7 Need

- Emergency numbers can be easily accessed in case of need.
- Collaboration of all the social media accounts makes ones life easy. No fuss in contacting people through various social accounts.
- Dynamic updation of the details maintains consistency all throughout which is needed for any storage system to be reliable.
- Security of data is ensured. (So, the user are confident that their data is available to all those whom he has allowed access to his personal details at the same time hiding the data from other users.)
- Invitations for any occasion can be sent on the fly with no last minute chaos in searching of the latest addresses and phone numbers of the long forgotten relatives whom one seldom interacts with on daily basis.
- No worries of the data being lost. Every is backed-up on the cloud.

2. The Building Blocks

- **Firestore:**

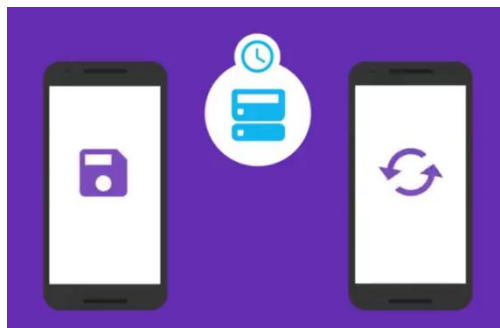
- Real-time Database:

- Store and sync data with our NoSQL cloud database.
 - Data is synced across all clients in real-time, and remains available when your app goes offline.

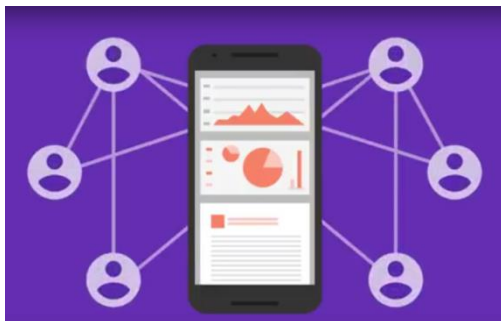
*Firestore



Real-time Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Real-time Database instance and automatically receive updates with the newest data.



Firestore lets users store and sync data in real-time.



Helps users collaborate with one another.



Whenever you update data in the database it stores data in the cloud and simultaneously updates the data on all the connected devices.



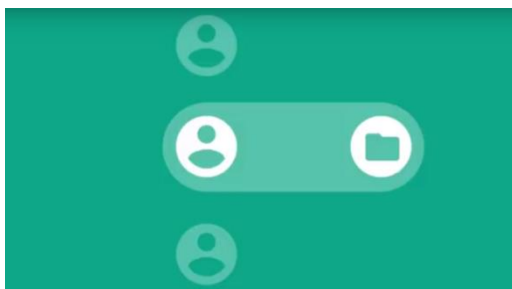
Whenever a device loses connection, the database STK uses the local cache on the device to serve and store changes



When the user comes back online the local data is automatically synchronized.



One can specify who has access to your data and how one's data needs to be structured.



We can ensure that the users can only access their own data.



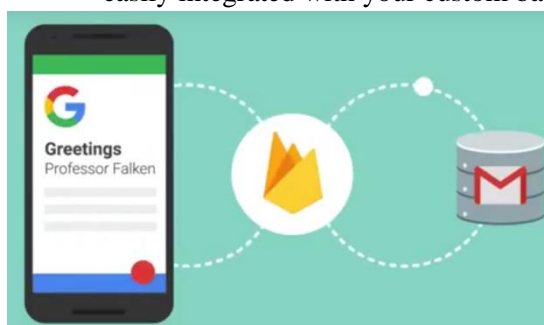
Limit on the maximum number of characters that can be used in the message can also be imposed.

➤ Authentication:

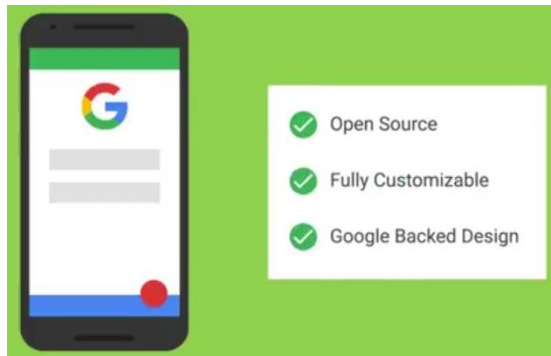
- Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.
- Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.



Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with your custom backend.



Most application requires identity of the user so that they can provide customized experience and keep their data secure. Firebase provides lots of different ways for the users to authenticate themselves. For example user can authenticate using email



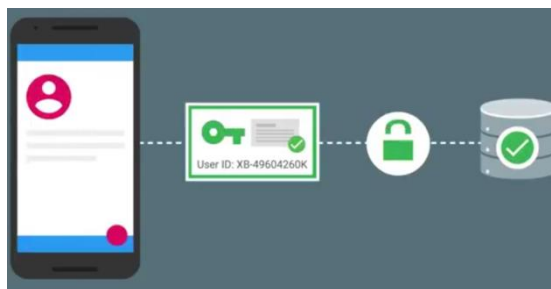
One can design their own interface or can use the firebase open source UI which is fully customizable and incorporates years of Google's experience in building simple sign-in UX.

Once a user authenticates:

- Info. about the user is returned to the device via callbacks.



The user information contains a unique ID which is guaranteed to be distinct across all providers, never changing for authenticated users.



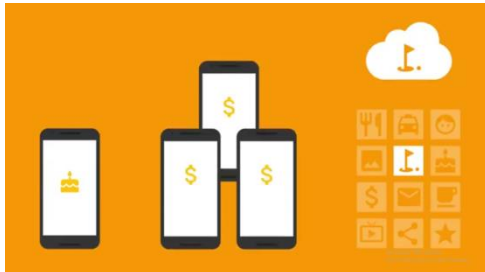
This unique ID is used to identify the user and what parts of the back-end system they're authorized to access.

Firebase also manages user session, so that users will remain logged in after the browser or application restarts. The firebase works on android, iOS and the web.

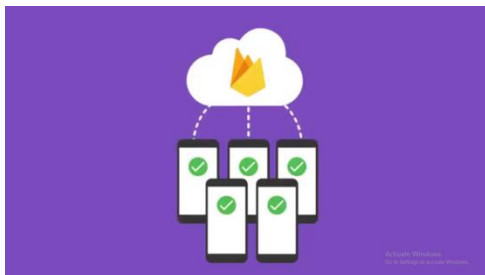
- Cloud Messaging
 - Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost.
 - Using FCM, you can notify a client app that new email or other data is available to sync. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app.



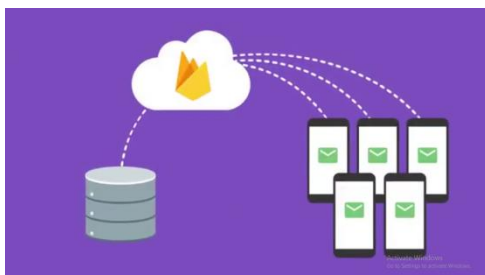
It's a free service that allows one to send messages to one's users' applications across a variety of platforms.



Messages can be addressed to single devices, groups of devices or even topics.



Building a notifications or messaging system with firebase is easy.
First, you register your users' app instances with the firebase cloud messaging servers.



Then on your server you write code that allows you to address these devices by ID, group or topic, and which tells the firebase cloud messaging server to send the messages for you.

Its powerful; its scalable delivering hundreds of billions of messages per day, with 95% of messages being delivered in 256 ms to the connect the devices across many different platforms.

➤ Cloud Storage:

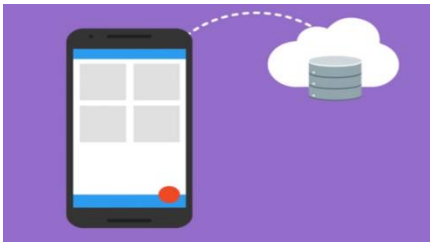
- Cloud Storage is built for app developers who need to store and serve user-generated content, such as photos or videos.
- Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content.

➤ How does it work?

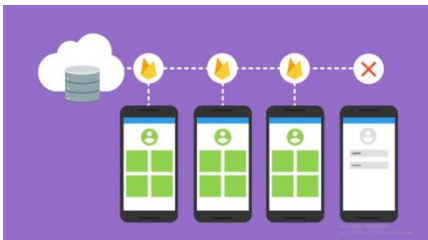
Developers use the Firebase SDKs for Cloud Storage to upload and download files directly from clients. If the network connection is poor, the client is able to retry the operation right where it left off, saving your users' time and bandwidth.

Cloud Storage stores your files in a Google Cloud Storage bucket, making them accessible through both Firebase and Google Cloud. This allows you the flexibility to upload and download files from mobile clients via the Firebase SDKs, and do server-side processing such as image filtering or video trans-coding using Google Cloud Platform. Cloud Storage scales automatically, meaning that there's no need to migrate to any other provider. Learn more about all the benefits of our integration with Google Cloud Platform.

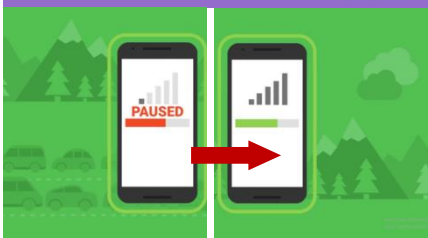
The Firebase SDKs for Cloud Storage integrate seamlessly with Firebase Authentication to identify users, and we provide a declarative security language that lets you set access controls on individual files or groups of files, so you can make files as public or private as you want.



Firebase API lets you upload your users' to its cloud so that it can be shared with anyone else.



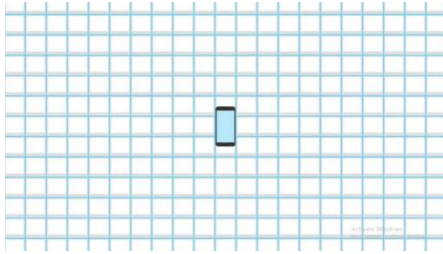
If there are specific rules for sharing files with certain users, one can protect these contents for users logged in with firebase authentication. Security is the first concern for firebase team. All transfers are performed over a secured connection.



All transfers with the firebase APIs are robust and will automatically resume in case the connection is broken. This is essential for transferring large files over slow or unreliable mobile connections.



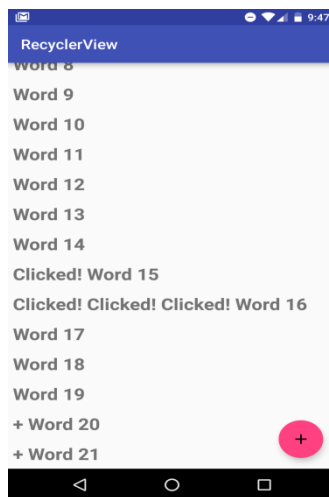
This type of storage, backed by Google Storage, scales to Petabytes.



That's billions of photos to meet the application needs. So one will never be out of space when needed.

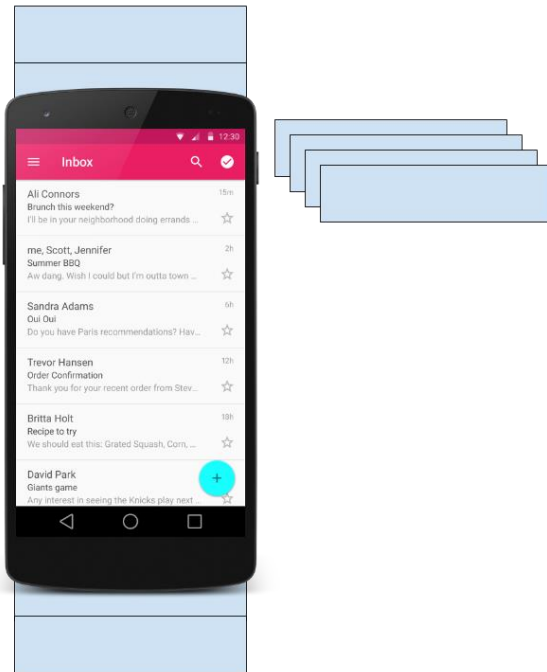
- ***Recycler View:***

- When you display a large number of items in a scrollable list, most items are not visible. For example, in a long list of words or many news headlines, the user only sees a small number of list item at a time.



Or you may have a dataset that changes as the user interacts with it. If you create a new view every time the data changes, that's also a lot of views, even for a small dataset.

From a performance perspective, you want to minimize the number of views kept around at any given point (Memory), and the number of views you have to create (Time). Both of these goals can be accomplished by creating somewhat more views than the user can see on the screen, and cache and reuse previously created views with different data as they scroll in and out of the view.



The RecyclerView class is a more advanced and flexible version of ListView. It is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.

Use the RecyclerView widget when you need to display a large amount of scrollable data, or data collections whose elements change at runtime based on user action or network events.

RecyclerView components

To display your data in a RecyclerView, you need the following parts:

Data. It doesn't matter where the data comes from. You can create the data locally, as you do in the practical, get it from a database on the device as you will do in a later practical, or pull it from the cloud.

A RecyclerView.

The scrolling list that contains the list items.

An instance of RecyclerView as defined in your activity's layout file to act as the container for the views.

Layout for one item of data. All list items look the same, so you can use the same layout for all of them. The item layout has to be created separately from the activity's layout, so that one item view at a time can be created and filled with data.

A layout manager.

The layout manager handles the organization (layout) of user interface components in a view. All view groups have layout managers. For the LinearLayout, the Android system handles the layout for you. RecyclerView requires an explicit layout manager to manage the arrangement of list items contained within it. This layout could be vertical, horizontal, or a grid.

The layout manager is an instance of RecyclerView.LayoutManager to organize the layout of the items in the RecyclerView

An adapter.

The adapter connects your data to the RecyclerView. It prepares the data and how will be displayed in a view holder. When the data changes, the adapter updates the contents of the respective list item view in the RecyclerView.

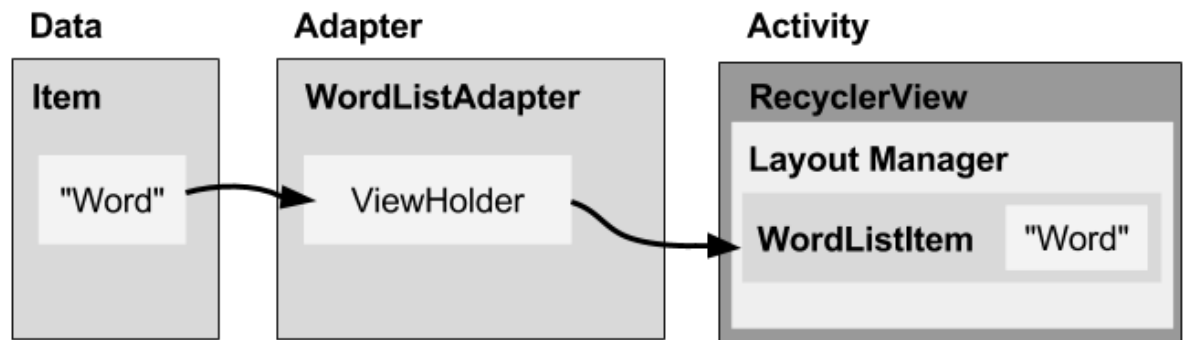
An adapter is an extension of `RecyclerView.Adapter`. The adapter uses a `ViewHolder` to hold the views that constitute each item in the `RecyclerView`, and to bind the data to be displayed into the views that display it.

A view holder.

The view holder extends the `ViewHolder` class. It contains the view information for displaying one item from the item's layout.

A view holder used by the adapter to supply data, which is an extension of “`RecyclerView.ViewHolder`”.

The diagram below shows the relationship between these components.



Data

Any displayable data can be shown in a RecyclerView.

Text , Images, Icons, etc.

Data can come from any source. For example news headlines.

RecyclerView is:

A View group for a scrollable container. Ideal for long lists of similar items. Uses only a limited number of views that are re-used when they go off-screen. This saves memory and makes it faster to update list items as the user scrolls through data, because it is not necessary to create a new view for every item that appears.

In general, the RecyclerView keeps as many item views as fit on the screen, plus a few extra at each end of the list to make sure that scrolling is fast and smooth.

Item Layout

The layout for a list item is kept in a separate file so that the adapter can create item views and edit their contents independently from the layout of the activity.

Layout Manager

A layout manager positions item views inside a view group, such as the RecyclerView and determines when to reuse item views that are no longer visible to the user. To reuse (or recycle) a view, a layout manager may ask the adapter to replace the contents of the view with a different element from the dataset. Recycling views in this manner improves performance by avoiding the creation of unnecessary views or performing expensive findViewById() lookups.

RecyclerView provides these built-in layout managers:

- LinearLayoutManager shows items in a vertical or horizontal scrolling list.
- GridLayoutManager shows items in a grid.
- StaggeredGridLayoutManager shows items in a staggered grid.

To create a custom layout manager, extend the RecyclerView.LayoutManager class.

Animations

Animations for adding and removing items are enabled by default in RecyclerView. To customize these animations, extend the RecyclerView.ItemAnimator class and use the RecyclerView.setItemAnimator() method.

Adapter

An Adapter helps two incompatible interfaces to work together. In the RecyclerView, the adapter connects data with views. It acts as an intermediary between the data and the view. The Adapter receives or retrieves the data, does any work required to make it displayable in a view, and places the data in a view.

For example, the adapter may receive data from a database as a Cursor object, extract the the word and its definition, convert them to strings, and place the strings in an item view that has two text views, one for the word and one for the definition. You will learn more about cursors in a later chapter.

The RecyclerView.Adapter implements a view holder, and must override the following callbacks:

- `onCreateViewHolder()` inflates an item view and returns a new view holder that contains it. This method is called when the `RecyclerView` needs a new view holder to represent an item.
- `onBindViewHolder()` sets the contents of an item at a given position in the `RecyclerView`. This is called by the `RecyclerView`, for example, when a new item scrolls into view.
- `getItemCount()` returns the total number of items in the data set held by the adapter.

View holder

A `RecyclerView.ViewHolder` describes an item view and metadata about its place within the `RecyclerView`. Each view holder holds one set of data. The adapter adds data to view holders for the layout manager to display.

You define your view holder layout in an XML resource file. It can contain (almost) any type of view, including clickable elements.

Implementing a `RecyclerView` requires the following steps:

- ❖ Add the `RecyclerView` dependency to the app's `app/build.gradle` file.
- ❖ Add the `RecyclerView` to the activity's layout
- ❖ Create a layout XML file for one item
- ❖ Extend `RecyclerView.Adapter` and implement `onCreateViewHolder` and `onBindViewHolder` methods.
- ❖ Extend `RecyclerView.ViewHolder` to create a view holder for your item layout. You can add click behavior by overriding the `onClick` method.
- ❖ In your activity, in the `onCreate` method, create a `RecyclerView` and initialize it with the adapter and a layout manager.
- ❖ **Add the dependency to `app/build.gradle`**
- ❖ Add the `recycler view` library to your `app/build.gradle` file as a dependency. Look at the chapter on support libraries or the `RecyclerView` practical, if you need detailed instructions.

```
dependencies {
    ...
    compile 'com.android.support:recyclerview-v7:24.1.1'
    ...
}
```

Add a RecyclerView to your activity's layout

```
<android.support.v7.widget.RecyclerView  
    android:id="@+id/recyclerview"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
</android.support.v7.widget.RecyclerView>
```

Use the recycler view from the support library to be compatible with older devices. The only required attributes are the id, along with the width and height. Customize the items, not this view group.

Create the layout for one item

Create an XML resource file and specify the layout of one item. This will be used by the adapter to create the view holder.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="vertical"  
    android:padding="6dp">
```

```
<TextView  
    android:id="@+id/word"  
    style="@style/word_title" />
```

```
</LinearLayout>
```

The text view has a @style element. A style is a collection of properties that specifies the look of a view. You can use styles to share display attributes with multiple views. An easy way to create a style is to extract the style of a UI element that you already created. For example, after styling a TextView, Right-click > Refactor > Extract > Style on the element and follow the dialog prompts. More details on styles are in the practical and in a later chapter.

Create an adapter with a view holder

Extend RecyclerView.Adapter and implement the onCreateViewHolder and onBindViewHolder methods.

Create a new Java class with the following signature:

```
public class WordListAdapter extends  
    RecyclerView.Adapter<WordListAdapter.WordViewHolder> { }
```

In the constructor, get an inflater from the current context, and your data.

```
public WordListAdapter(Context context, LinkedList<String> wordList) {  
    mInflater = LayoutInflater.from(context);  
    this.mWordList = wordList;  
}
```

For this adapter, you have to implement 3 methods.

onCreateViewHolder() creates a view and returns it.

```

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType){
    // Inflate an item view.
    View itemView = LayoutInflater.inflate(R.layout.wordlist_item, parent, false);
    return new ViewHolder(itemView, this);
}

```

onBindViewHolder() associates the data with the view holder for a given position in the RecyclerView.

```

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    // Retrieve the data for that position
    String mCurrent = mWordList.get(position);
    // Add the data to the view
    holder.itemView.setText(mCurrent);
}

```

getItemCount() returns to number of data items available for displaying.

```

@Override
public int getItemCount() {
    return mWordList.size();
}

```

Implement the view holder class

Extend RecyclerView.ViewHolder to create a view holder for your item layout. You can add click behavior by overriding the onClick method.

This class is usually defined as an inner class to the adapter and extends RecyclerView.ViewHolder.

```
class ViewHolder extends RecyclerView.ViewHolder { }
```

If you want to add click handling, you need to implement a click listener. One way to do this is to have the view holder implement the click listener methods.

```
// Extend the signature of ViewHolder to implement a click listener.
class ViewHolder extends RecyclerView.ViewHolder implements
View.OnClickListener { }
```

In its constructor, the view holder has to inflate its layout, associate with its adapter, and, if applicable, set a click listener.

```

public ViewHolder(View itemView, WordListAdapter adapter) {
    super(itemView);
    wordItemView = (TextView) itemView.findViewById(R.id.word);
    this.mAdapter = adapter;
    itemView.setOnClickListener(this);
}

```

And, if you implementing OnClickListener, you also have to implement onClick().

```

@Override
public void onClick(View v) {
    wordItemView.setText("Clicked! " + wordItemView.getText());
}

```


Note that to attach click listeners to other elements of the view holder, you do that dynamically in `onBindViewHolder`. (You will do this a later practical, when you will be extending the recycler view code from the practical.)

Create the RecyclerView

Finally, to tie it all together, in your activity's `onCreate()` method:

Get a handle to the `RecyclerView`.

```
mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview);
```

Create an adapter and supply the data to be displayed.

```
mAdapter = new WordListAdapter(this, mWordList);
```

Connect the adapter with the recycler view.

```
mRecyclerView.setAdapter(mAdapter);
```

Give the recycler view a default layout manager.

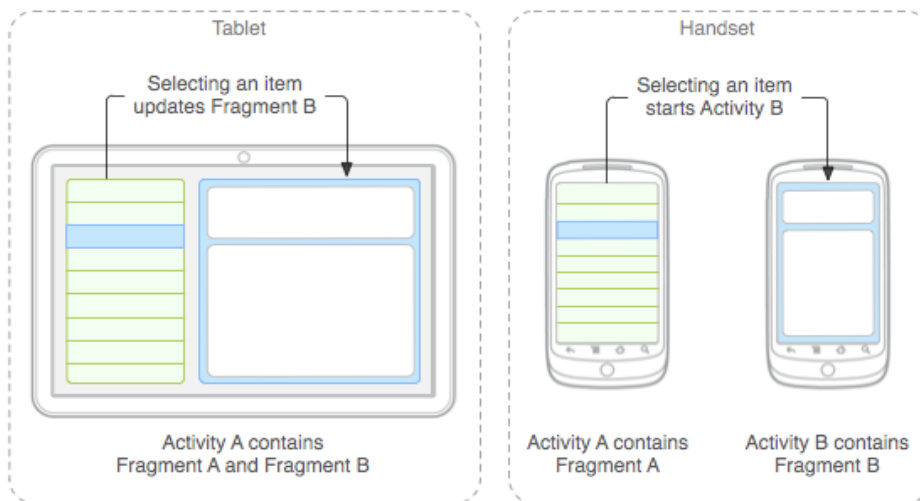
```
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

`RecyclerView` is an efficient way for displaying scrolling list data. It uses the adapter pattern to connect data with list item views. To implement a `RecyclerView` you need to create an adapter and a view holder, and the methods that take the data and add it to the list items.

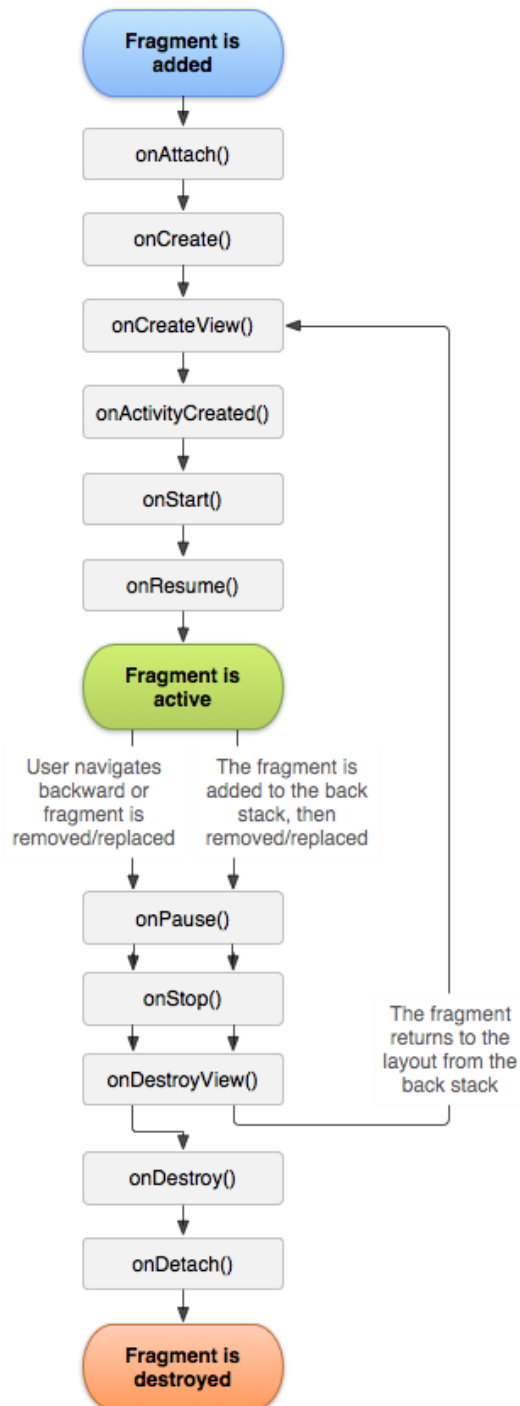
- ***Fragments:***

- A `Fragment` represents a behavior or a portion of user interface in a `FragmentActivity`. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).
- A fragment must always be hosted in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle. For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments. However, while an activity is running (it is in the *resumed* lifecycle state), you can manipulate each fragment independently, such as add or remove them. When you perform such a fragment transaction, you can also add it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred. The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the *Back* button.
- When you add a fragment as a part of your activity layout, it lives in a `ViewGroup` inside the activity's view hierarchy and the fragment defines its own view layout. You can insert a fragment into your activity layout by declaring the fragment in

the activity's layout file, as a `<fragment>` element, or from your application code by adding it to an existing `ViewGroup`.



Creating a Fragment.



To create a fragment, you must create a subclass of `Fragment` (or an existing subclass of it). The `Fragment` class has code that looks a lot like an `Activity`. It contains callback methods similar to an activity, such as `onCreate()`, `onStart()`, `onPause()`, and `onStop()`. In fact, if you're converting an existing Android application to use fragments, you might simply move code from your activity's callback methods into the respective callback methods of your fragment.

Usually, you should implement at least the following lifecycle methods:

`onCreate()`

The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

`onCreateView()`

The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a `View` from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.

`onPause()`

The system calls this method as the first indication that the user is leaving the fragment (though it doesn't always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

Most applications should implement at least these three methods for every fragment, but there are several other callback methods you should also use to handle various stages of the fragment lifecycle. All the lifecycle callback methods are discussed in more detail in the section about Handling the Fragment Lifecycle.

Note that the code implementing lifecycle actions of a dependent component should be placed in the component itself, rather than directly in the fragment callback implementations. See Handling Lifecycles with Lifecycle-Aware Components to learn how to make your dependent components lifecycle-aware.

There are also a few subclasses that you might want to extend, instead of the base `Fragment` class:

`DialogFragment`

Displays a floating dialog. Using this class to create a dialog is a good alternative to using the dialog helper methods in the `Activity` class, because you can incorporate a fragment dialog into the back stack of fragments managed by the activity, allowing the user to return to a dismissed fragment.

`ListFragment`

Displays a list of items that are managed by an adapter (such as a `SimpleCursorAdapter`), similar to `ListActivity`. It provides several methods for managing a list view, such as the `onListItemClick()` callback to handle click events. (Note that the preferred method for displaying a list is to use `RecyclerView` instead of `ListView`. In this case you would need to create a fragment that includes a `RecyclerView` in its layout. See [Create a List with RecyclerView](#) to learn how.)

PreferenceFragmentCompat

Displays a hierarchy of Preference objects as a list, similar to PreferenceActivity. This is useful when creating a "settings" activity for your application.

Adding a user interface

A fragment is usually used as part of an activity's user interface and contributes its own layout to the activity.

To provide a layout for a fragment, you must implement the `onCreateView()` callback method, which the Android system calls when it's time for the fragment to draw its layout. Your implementation of this method must return a `View` that is the root of your fragment's layout.

Note: If your fragment is a subclass of `ListFragment`, the default implementation returns a `ListView` from `onCreateView()`, so you don't need to implement it.

To return a layout from `onCreateView()`, you can inflate it from a layout resource defined in XML. To help you do so, `onCreateView()` provides a `LayoutInflater` object.

For example, here's a subclass of `Fragment` that loads a layout from the `example_fragment.xml` file:

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

Creating a layout

In the sample above, `R.layout.example_fragment` is a reference to a layout resource named `example_fragment.xml` saved in the application resources. For information about how to create a layout in XML, see the User Interface documentation.

The `container` parameter passed to `onCreateView()` is the parent `ViewGroup` (from the activity's layout) in which your fragment layout is inserted. The `savedInstanceState` parameter is a `Bundle` that provides data about the previous instance of the fragment, if the fragment is being resumed (restoring state is discussed more in the section about Handling the Fragment Lifecycle).

The `inflate()` method takes three arguments:

- The resource ID of the layout you want to inflate.
- The `ViewGroup` to be the parent of the inflated layout. Passing the container is important in order for the system to apply layout parameters to the root view of the inflated layout, specified by the parent view in which it's going.
- A boolean indicating whether the inflated layout should be attached to the `ViewGroup` (the second parameter) during inflation. (In this case, this is false because the system is already inserting the inflated layout into the container—passing true would create a redundant view group in the final layout.)

3. Installation guide

Linkr is an android application, which can be downloaded by the interested users from the standard android app store called “Google Play Store”.

So, step 1 is without any doubt to connect to the internet.

The size of the application is --- . Thus, the users are advised to check the storage space availability on their device for the successful download of the application

A search for “Linkr” is done on the play store followed by just a click on the “install” button.

After successful download and installation of Linkr on his/her device, the user is now ready to use the application.

As soon as the app icon is clicked a login page appears on the screen.

The application requires mandatory authentication for the the people who wish to access the application.

To ease and speed-up the authentication procedure, Linkr incorporates Google account based authentication which eliminates the long process of registering and verification of email addresses.

So after the user has signed-in the application using the google account, a screen is pops-up, requiring the user to set up his/her profile by completing the simple tasks of setting up the desired profile photo and user name.

The above steps are preliminary steps and are supposed to be followed by all the users.

Apart from the above steps, to utilise the application features in best way, one may further try out the useful functionalities that linkr offers.

These include :

Search among the list of all the users to find acquaintances, friends and family members.

Send requests to the desired people.

Accept the pending requests of other users.

Update the profile with all the relevant and current contact information.

4. Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on [IntelliJ IDEA](#). On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for [Google Cloud Platform](#), making it easy to integrate Google Cloud Messaging and App Engine

This page provides an introduction to basic Android Studio features. For a summary of the latest changes, see [Android Studio release notes](#).

Project structure

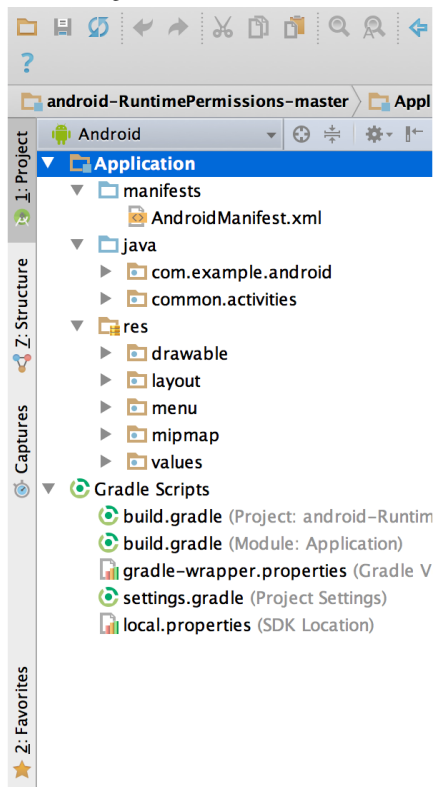


Figure 1. The project files in Android view.

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests:** Contains the AndroidManifest.xml file.
- **java:** Contains the Java source code files, including JUnit test code.
- **res:** Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as **Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

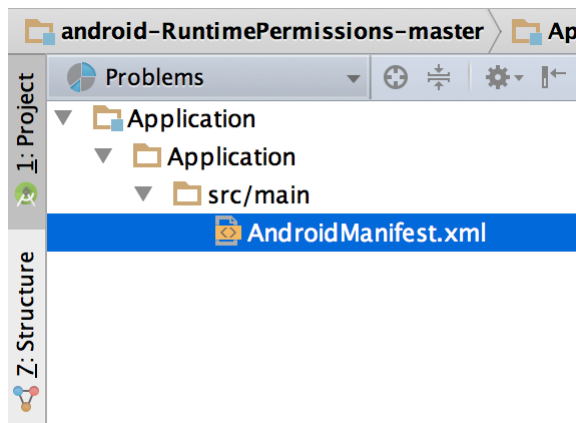


Figure 2. The project files in Problems view, showing a layout file with a problem.

The user interface

The Android Studio main window is made up of several logical areas identified in figure 3.

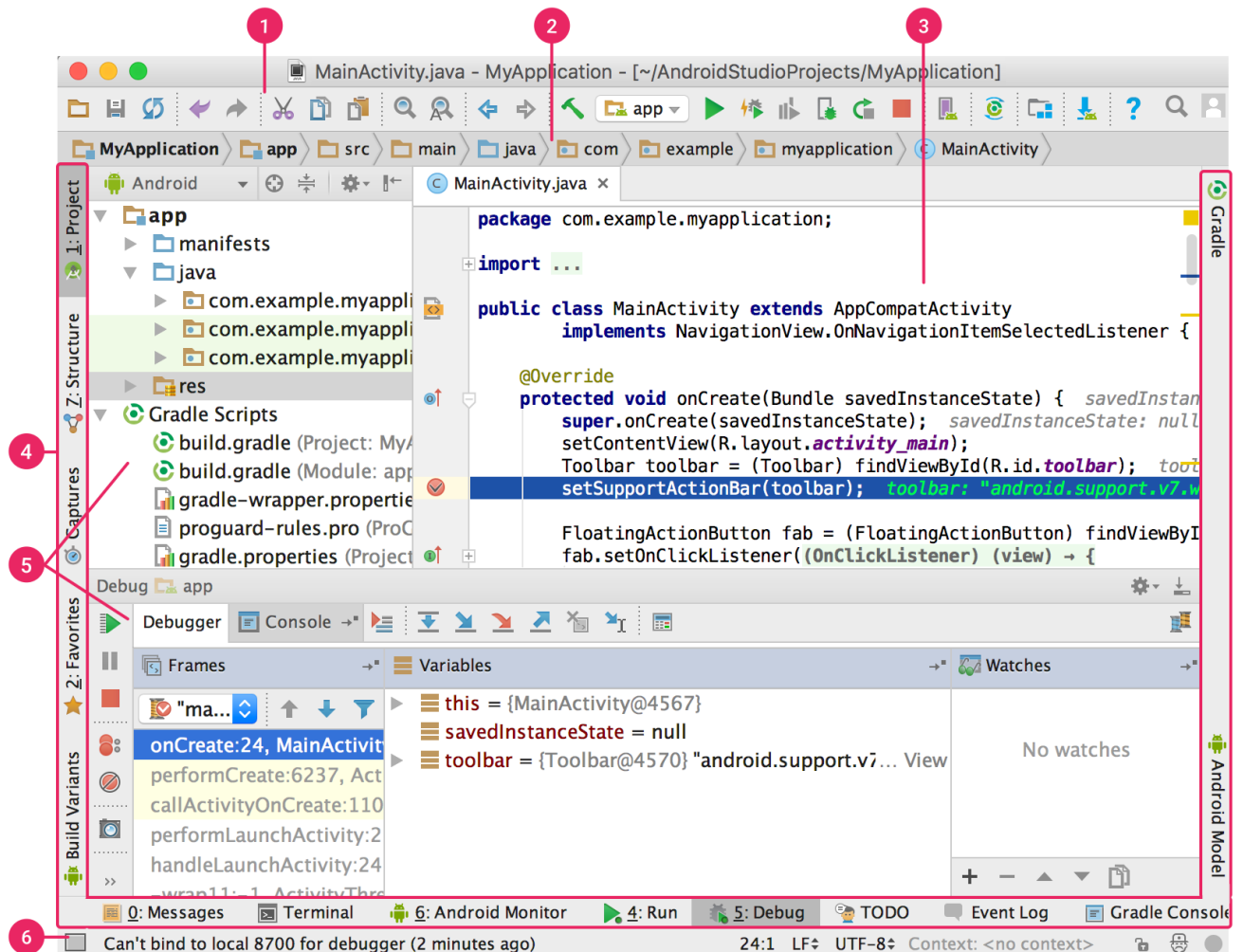


Figure 3. The Android Studio main window.

1. The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.


You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful if, for

example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

Tool windows

Instead of using preset perspectives, Android Studio follows your context and automatically brings up relevant tool windows as you work. By default, the most commonly used tool windows are pinned to the tool window bar at the edges of the application window.

- To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.
- To return to the current default tool window layout, click **Window > Restore Default Layout** or customize your default layout by clicking **Window > Store Current Layout as Default**.
- To show or hide the entire tool window bar, click the window icon  in the bottom left-hand corner of the Android Studio window.
- To locate a specific tool window, hover over the window icon and select the tool window from the menu.

You can also use keyboard shortcuts to open tool windows. Table 1 lists the shortcuts for the most common windows.

Table 1. Keyboard shortcuts for some useful tool windows.

Tool window	Windows and Linux	Mac
Project	Alt+1	Command+1
Version Control	Alt+9	Command+9
Run	Shift+F10	Control+R
Debug	Shift+F9	Control+D
Logcat	Alt+6	Command+6
Return to Editor	Esc	Esc
Hide All Tool Windows	Control+Shift+F12	Command+Shift+F12

If you want to hide all toolbars, tool windows, and editor tabs, click **View > Enter Distraction Free Mode**. This enables *Distraction Free Mode*. To exit *Distraction Free Mode*, click **View > Exit Distraction Free Mode**.

You can use *Speed Search* to search and filter within most tool windows in Android Studio. To use *Speed Search*, select the tool window and then type your search query.

Code completion

Android Studio has three types of code completion, which you can access using keyboard shortcuts.

Table 2. Keyboard shortcuts for code completion.

Type	Description	Windows and Linux	Mac
Basic Completion	Displays basic suggestions for variables, types, methods, expressions, and so on. If you call basic completion twice in a row, you see more results, including private members and non-imported static members.	Control+Space	Control+Space
Smart Completion	Displays relevant options based on the context. Smart completion is aware of the expected type and data flows. If you call Smart Completion twice in a row, you see more results, including chains.	Control+Shift+Space	Control+Shift+Space
Statement Completion	Completes the current statement for you, adding missing parentheses, brackets, braces, formatting, etc.	Control+Shift+Enter	Shift+Command+Enter

You can also perform quick fixes and show intention actions by pressing **Alt+Enter**.

Find sample code

The Code Sample Browser in Android Studio helps you find high-quality, Google-provided Android code samples based on the currently highlighted symbol in your project.

Navigation

Here are some tips to help you move around Android Studio.

- Switch between your recently accessed files using the *Recent Files* action. Press **Control+E** (**Command+E** on a Mac) to bring up the Recent Files action. By default, the last accessed file is selected. You can also access any tool window through the left column in this action.
- View the structure of the current file using the *File Structure* action. Bring up the File Structure action by pressing **Control+F12** (**Command+F12** on a Mac). Using this action, you can quickly navigate to any part of your current file.
- Search for and navigate to a specific class in your project using the *Navigate to Class* action. Bring up the action by pressing **Control+N** (**Command+O** on a Mac). Navigate to Class supports sophisticated expressions, including camel humps, paths, line navigate to, middle name matching, and many more. If you call it twice in a row, it shows you the results out of the project classes.
- Navigate to a file or folder using the *Navigate to File* action. Bring up the Navigate to File action by pressing **Control+Shift+N** (**Command+Shift+O** on a Mac). To search for folders rather than files, add a / at the end of your expression.
- Navigate to a method or field by name using the *Navigate to Symbol* action. Bring up the Navigate to Symbol action by pressing **Control+Shift+Alt+N** (**Command+Option+O** on a Mac).
- Find all the pieces of code referencing the class, method, field, parameter, or statement at the current cursor position by pressing **Alt+F7** (**Option+F7** on a Mac).

Style and formatting

As you edit, Android Studio automatically applies formatting and styles as specified in your code style settings. You can customize the code style settings by programming language, including specifying conventions for tabs and indents, spaces, wrapping and braces, and blank lines. To customize your code style settings, click **File > Settings > Editor > Code Style (Android Studio > Preferences > Editor > Code Style on a Mac.)**

Although the IDE automatically applies formatting as you work, you can also explicitly call the *Reformat Code* action by pressing **Control+Alt+L (Opt+Command+L on a Mac)**, or auto-indent all lines by pressing **Control+Alt+I (Control+Option+I on a Mac)**.

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mActionBar = getSupportActionBar();  
    mActionBar.setDisplayHomeAsUpEnabled(true);  
}
```

Figure 4. Code before formatting.

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mActionBar = getSupportActionBar();  
    mActionBar.setDisplayHomeAsUpEnabled(true);  
    // Get reference to the drawer layout and set event listener  
}
```

Formatted 7 lines
Show reformat dialog: ⌘⇧⌘L

Figure 5. Code after formatting.

Version control basics

Android Studio supports a variety of version control systems (VCS's), including Git, GitHub, CVS, Mercurial, Subversion, and Google Cloud Source Repositories.

After importing your app into Android Studio, use the Android Studio VCS menu options to enable VCS support for the desired version control system, create a repository, import the new files into version control, and perform other version control operations:

1. From the Android Studio **VCS** menu, click **Enable Version Control Integration**.
2. From the drop-down menu, select a version control system to associate with the project root, and then click **OK**.

The VCS menu now displays a number of version control options based on the system you selected.

Note: You can also use the **File > Settings > Version Control** menu option to set up and modify the version control settings.

Gradle build system

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the [Android plugin for Gradle](#). This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app, with different features using the same project and modules.
- Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are named `build.gradle`. They are plain text files that use [Groovy](#) syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

Build variants

The build system can help you create different versions of the same application from a single project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs for different device configurations on Google Play.

Multiple APK support

Multiple APK support allows you to efficiently create multiple APKs based on screen density or ABI. For example, you can create separate APKs of an app for the `hdpi` and `mdpi` screen densities, while still considering them a single variant and allowing them to share test APK, `javac`, `dx`, and ProGuard settings.

Resource shrinking

Resource shrinking in Android Studio automatically removes unused resources from your packaged app and library dependencies. For example, if your application is using [Google Play services](#) to access Google Drive functionality, and you are not currently using [Google Sign-In](#), then resource shrinking can remove the various drawable assets for the `SignInButton` buttons.

Note: Resource shrinking works in conjunction with code shrinking tools, such as ProGuard.

Managing dependencies

Dependencies for your project are specified by name in the `build.gradle` file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your `build.gradle` file. Android Studio configures projects to use the Maven Central Repository by default.

Debug and profile tools

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

Inline debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object

- Method return values
- Lambda and operator expressions
- Tooltip values

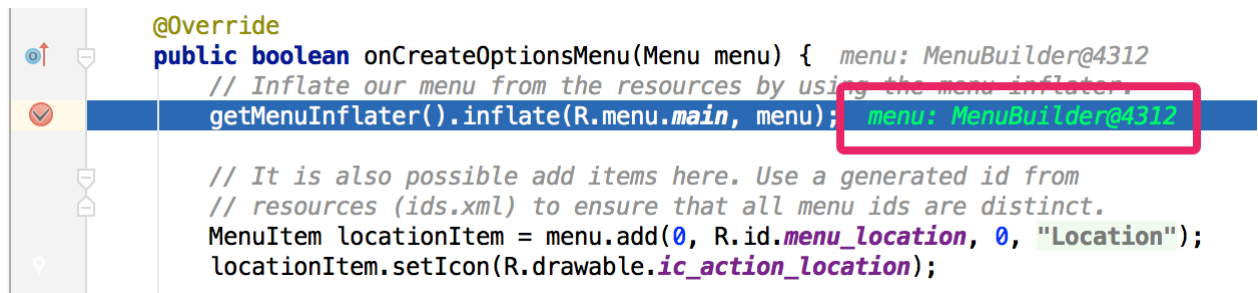



Figure 6. An inline variable value.

To enable inline debugging, in the **Debug** window, click **Settings**  and select the checkbox for **Show Values Inline**.

Performance profilers

Android Studio provides performance profilers so you can more easily track your app's memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyze network requests. With your app running on a device or emulator, open the **Android Profiler** tab.

Heap dump

When you're profiling memory usage in Android Studio, you can simultaneously initiate garbage collection and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks.

Memory Profiler

You can use Memory Profiler to track memory allocation and watch where objects are being allocated when you perform certain actions. Knowing these allocations enables you to optimize your app's performance and memory use by adjusting the method calls related to those actions.

Data file access

The Android SDK tools, such as [Systrace](#), and [logcat](#), generate performance and debugging data for detailed app analysis.

To view the available generated data files, open the Captures tool window. In the list of the generated files, double-click a file to view the data. Right-click any .hprof files to convert them to the standard [Investigate your RAM usage](#) file format.

Code inspections

Whenever you compile your program, Android Studio automatically runs configured [Lint](#) and other [IDE inspections](#) to help you easily identify and correct problems with the structural quality of your code.

The Lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.

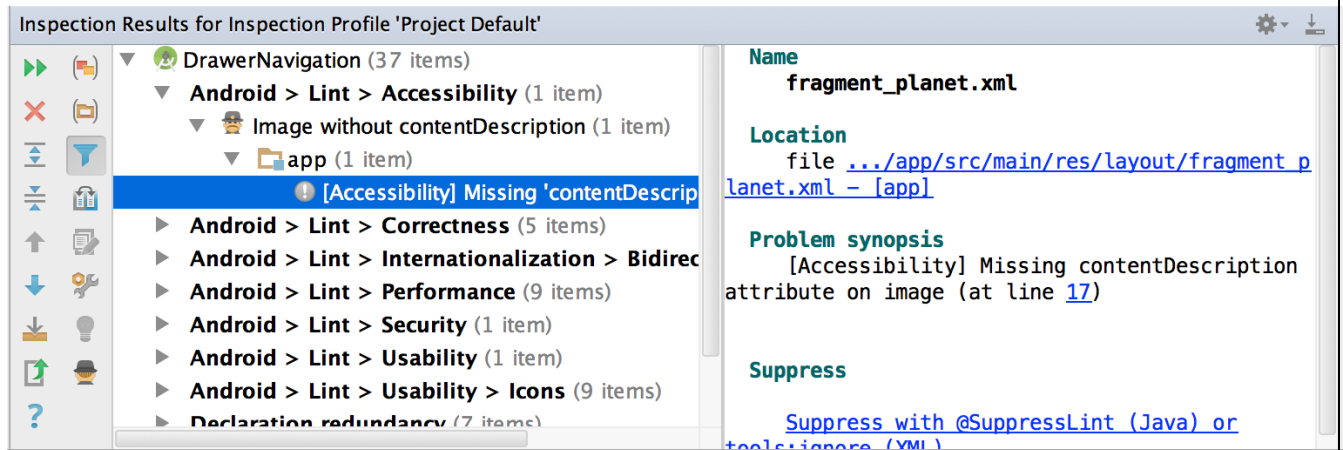


Figure 7. The results of a Lint inspection in Android Studio.

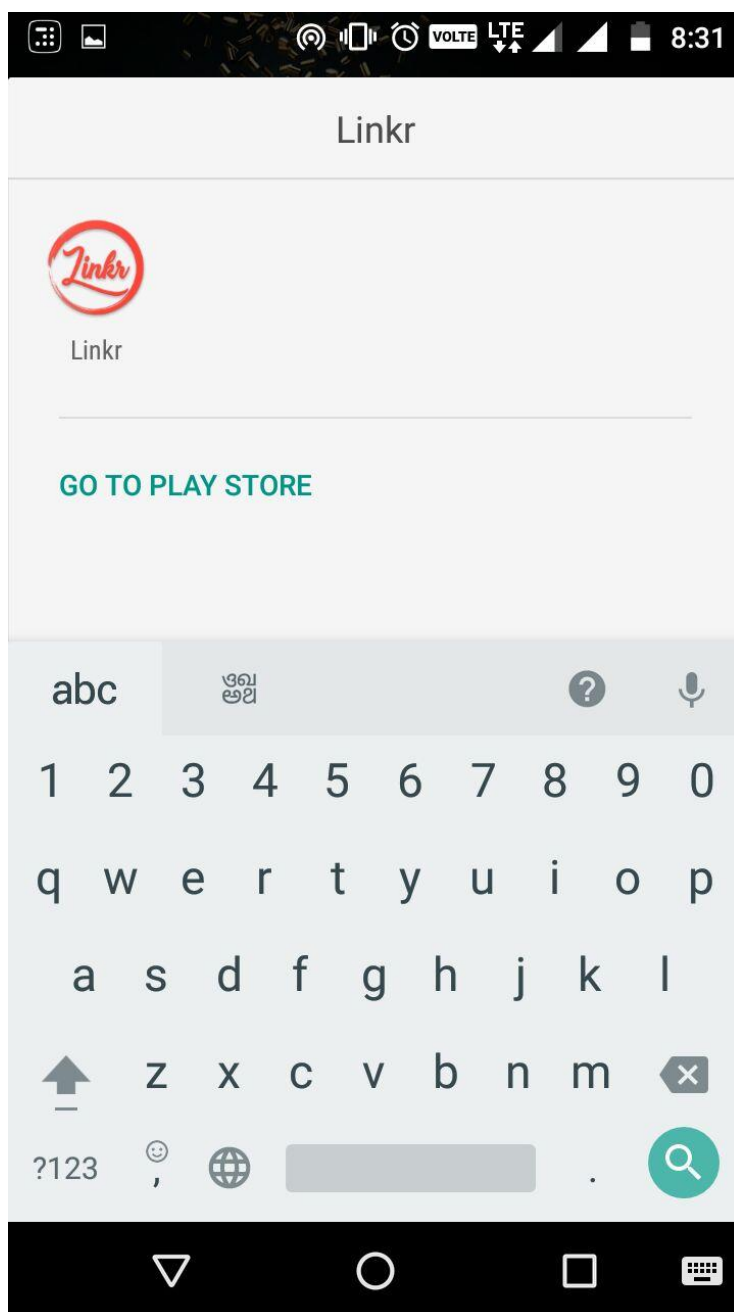
In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline your coding workflow.

Annotations in Android Studio

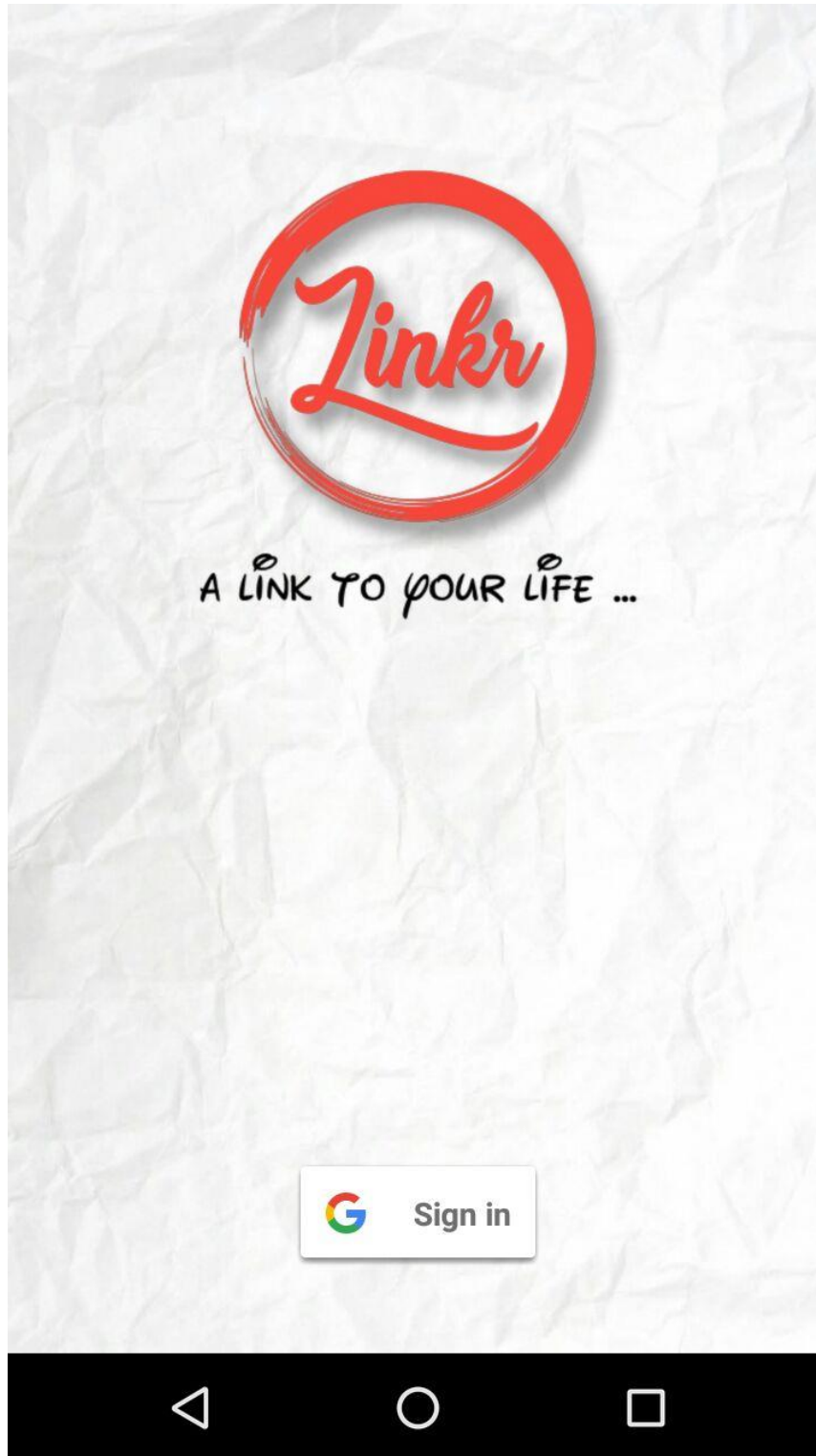
Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the Support-Annotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection.

5. Walkthrough

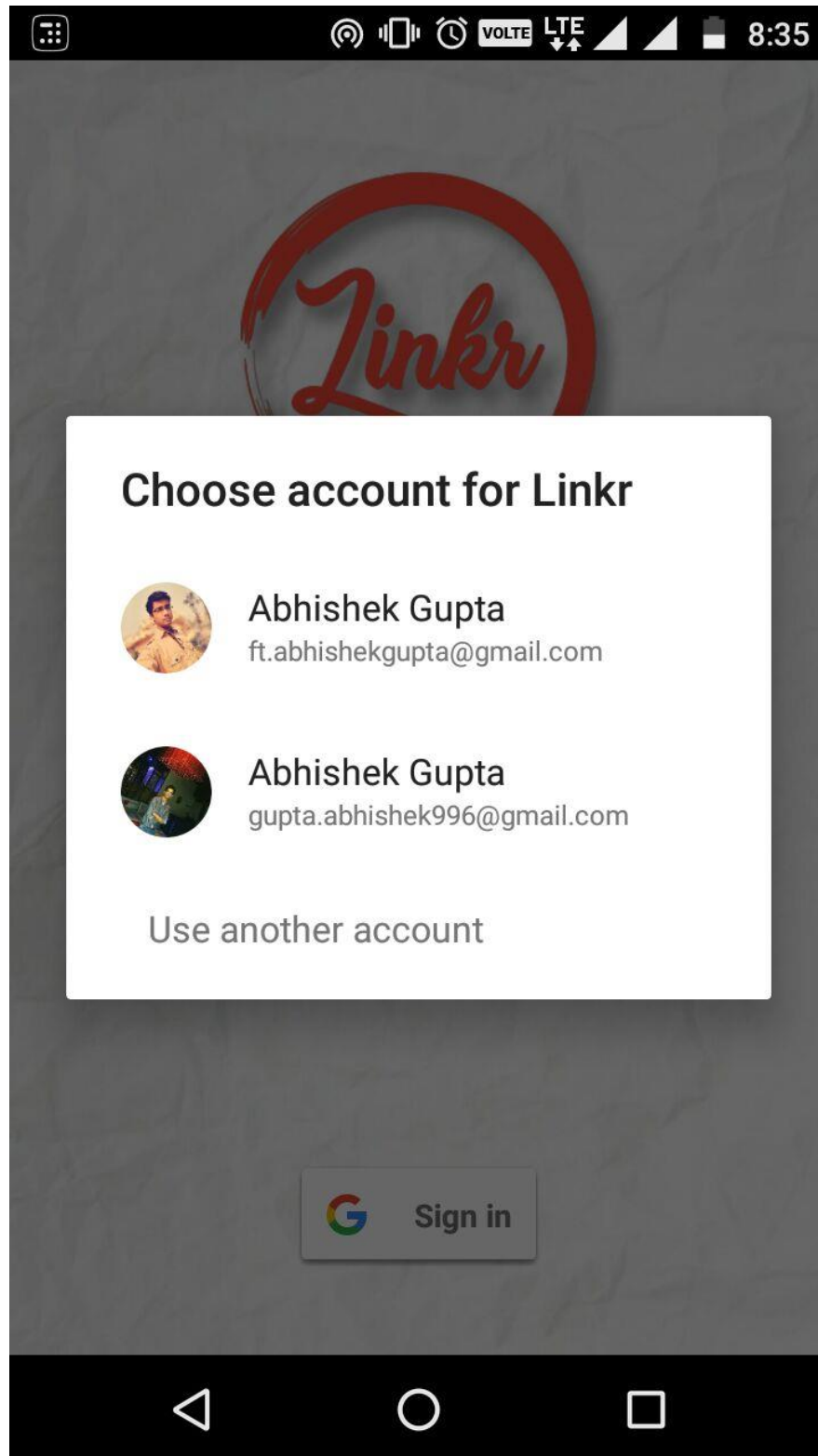
1. Open App from your Home Screen



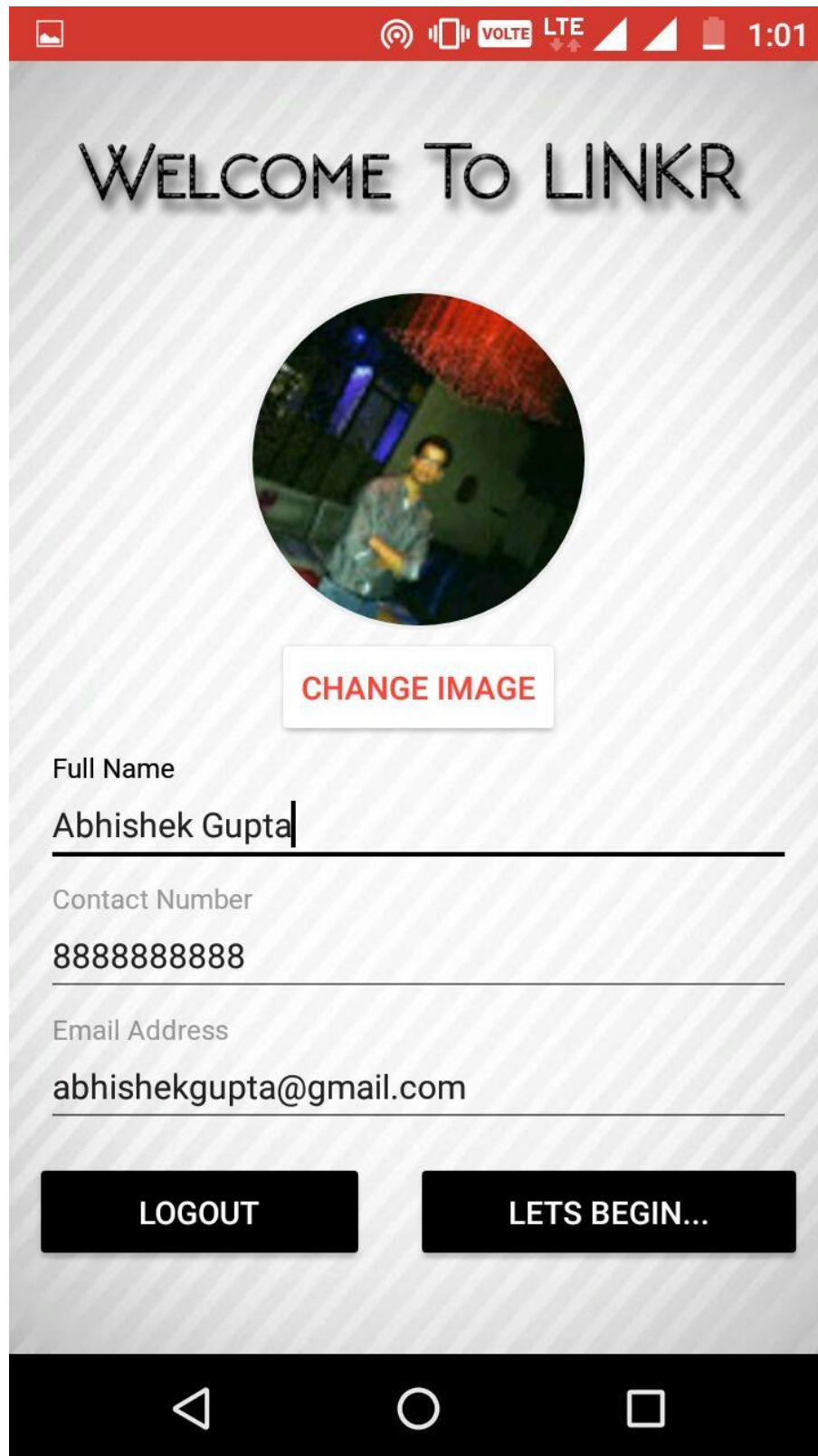
2. Login Screen Appears , Click on Sign In Button to choose account to login



3. Dialog Box Appears , choose the account you want to associate with your profile



4. For the first time use of the app , you have to setup your profile by entering your name , email , phone number and your profile picture



WELCOME TO LINKR

CHANGE IMAGE

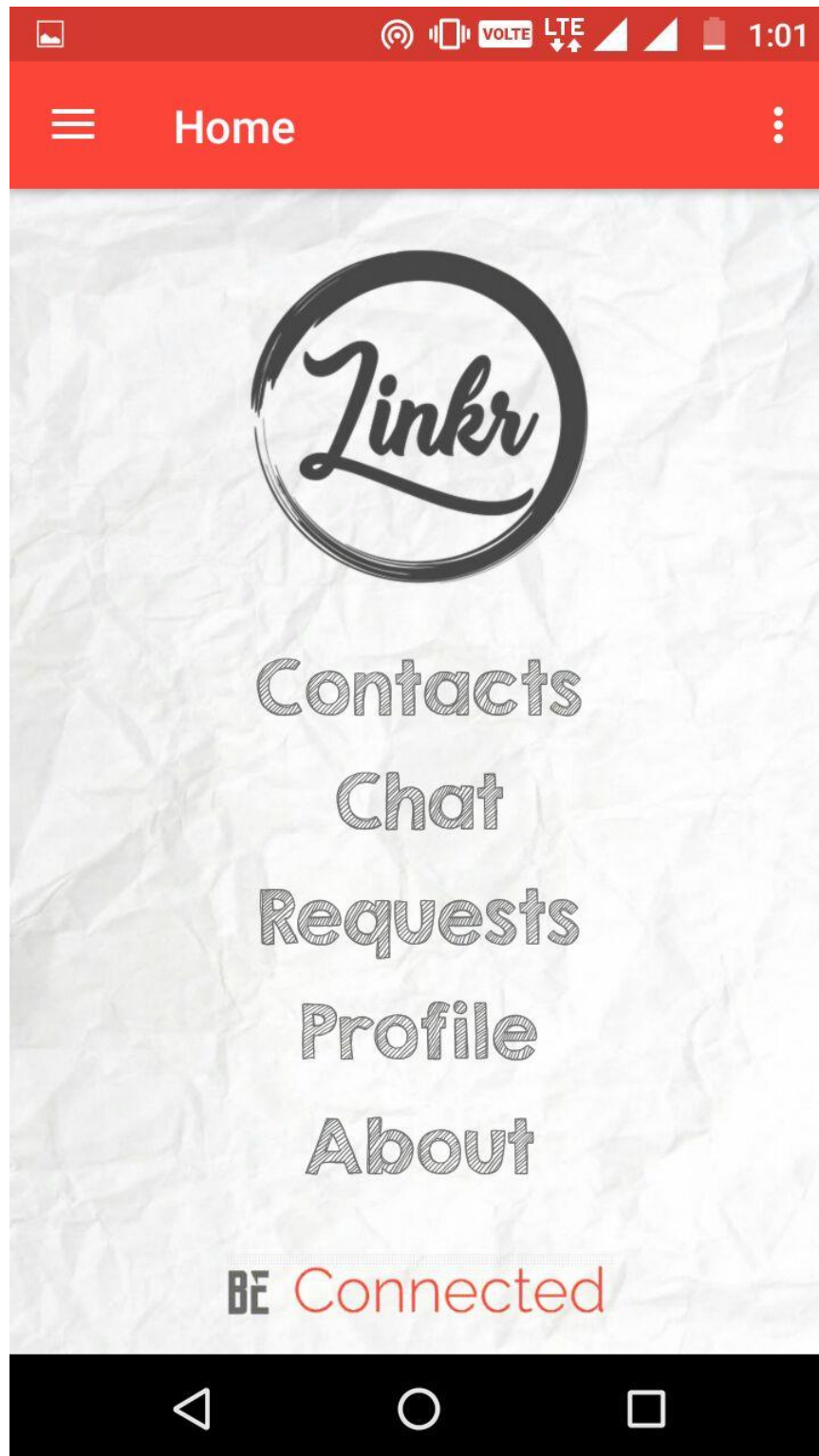
Full Name
Abhishek Gupta

Contact Number
8888888888

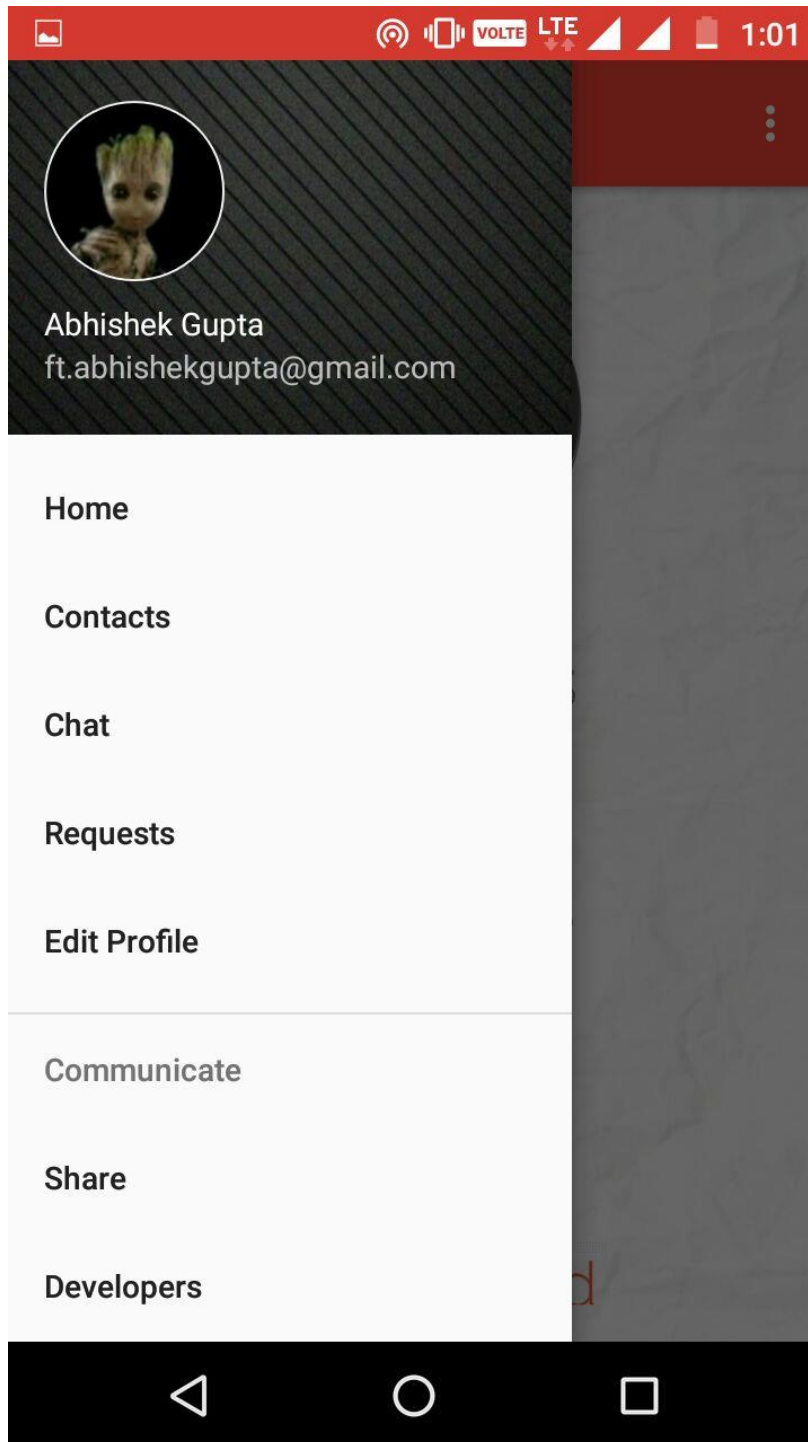
Email Address
abhishekgupta@gmail.com

LOGOUT LETS BEGIN...

5. Click on Lets Begin... Button when you are ready. Next Home Screen of the app will be displayed.

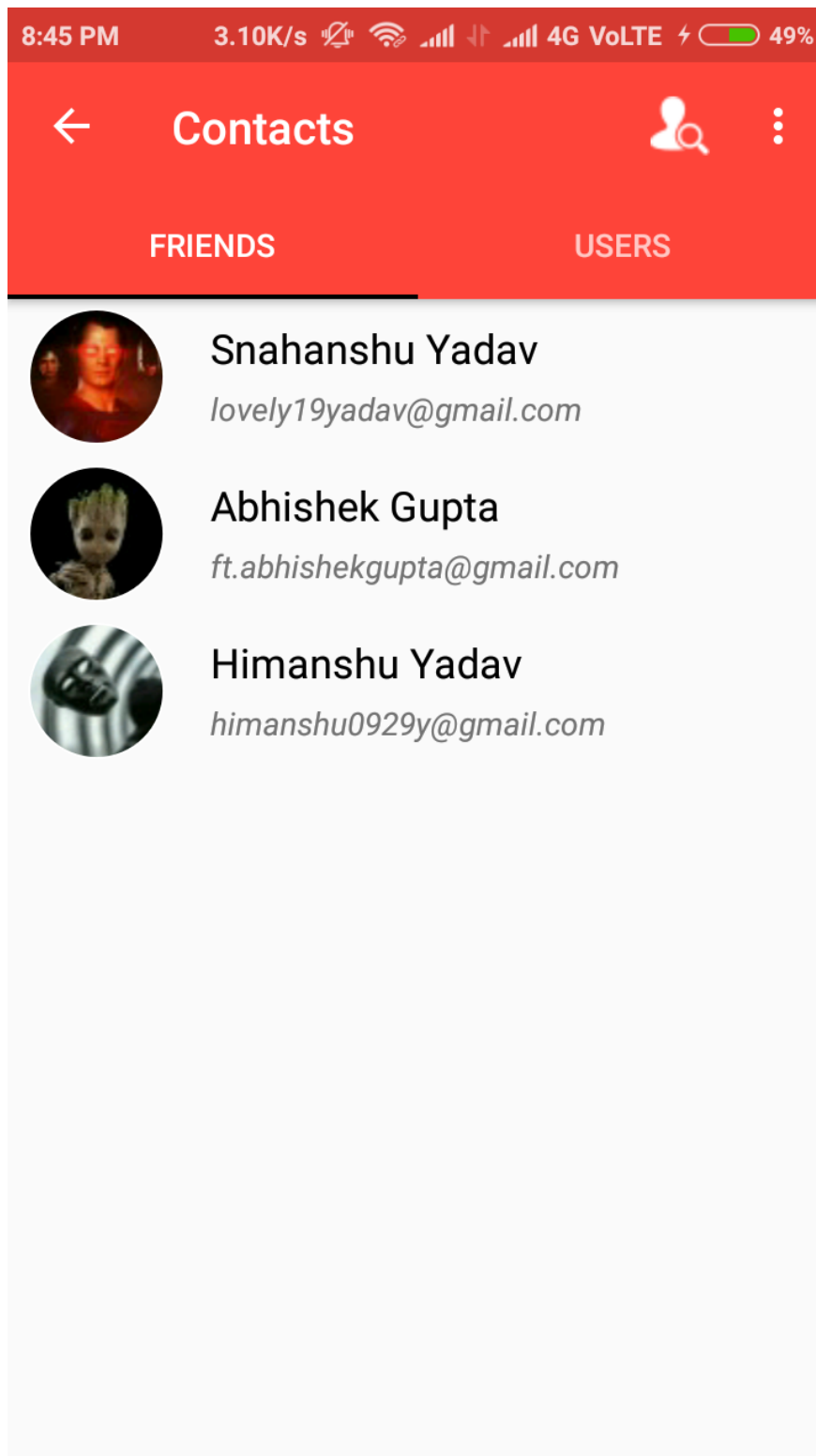


6. Navigation Drawer is also provided for ease of access to different sections of the app



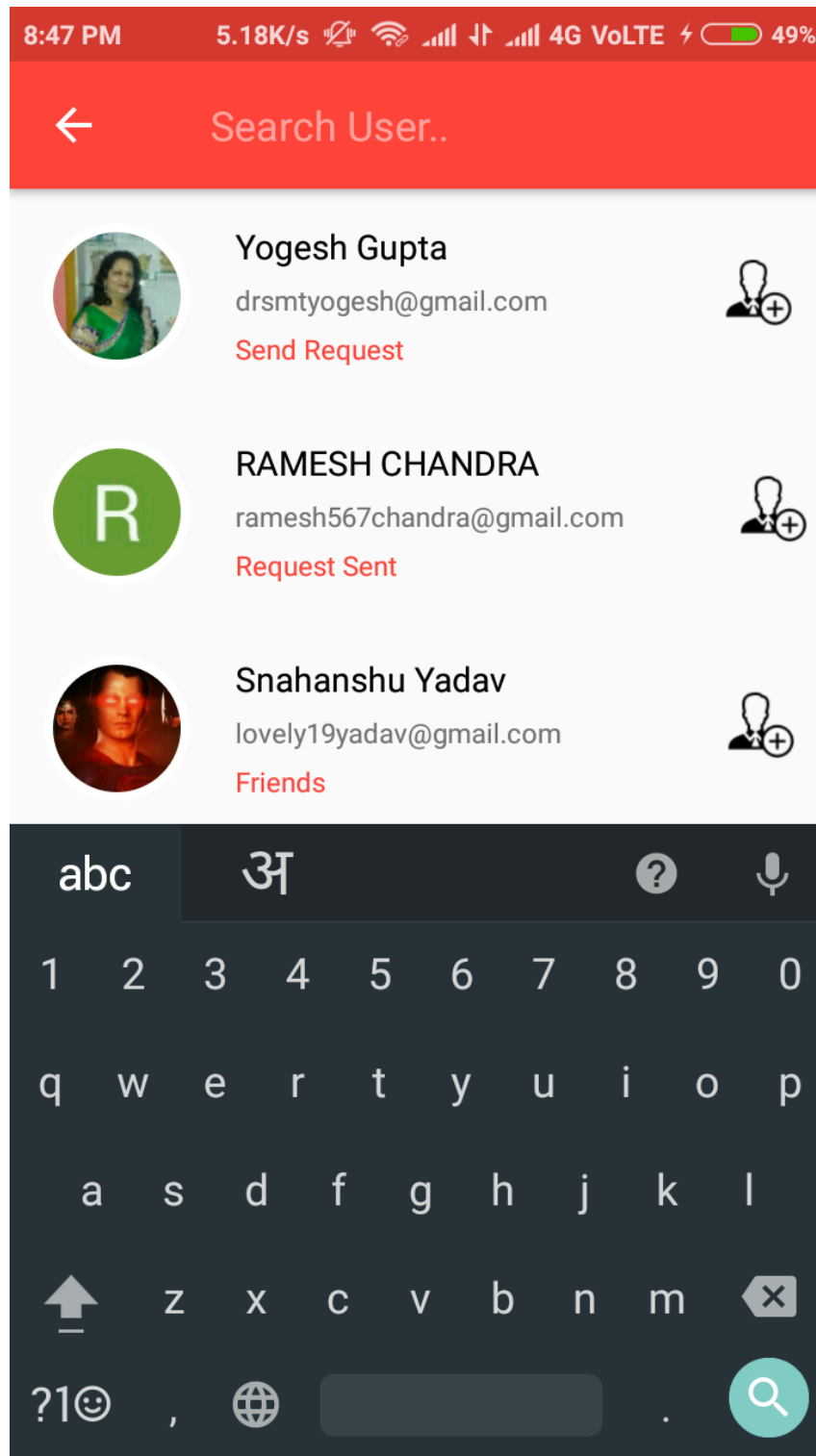
7. Contacts section has two tabs

- One for the users that are friends
- Other for all the users using the Linkr App



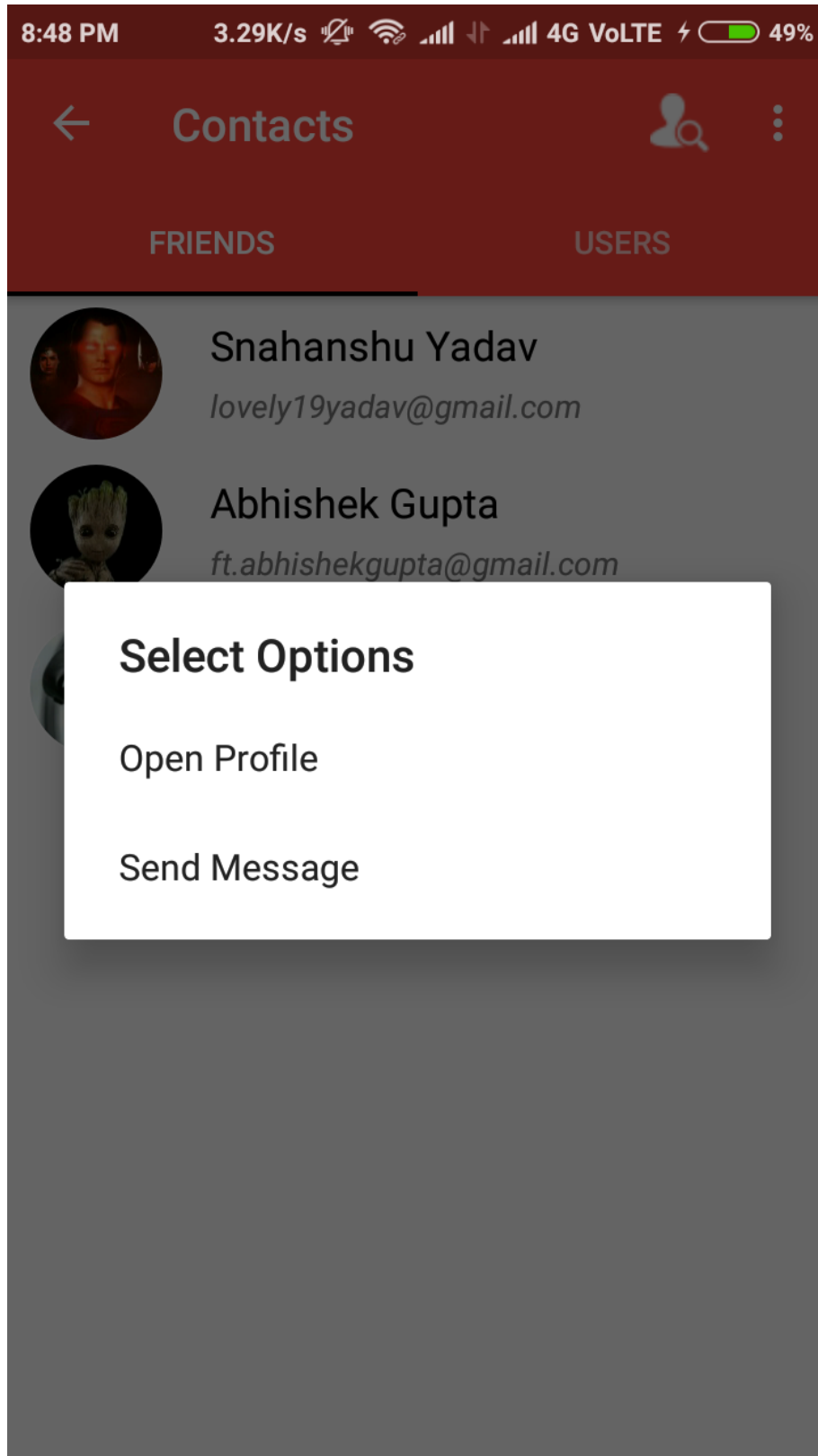
○

8. Search feature is also provided in the app for easily searching the desired user for a long list.



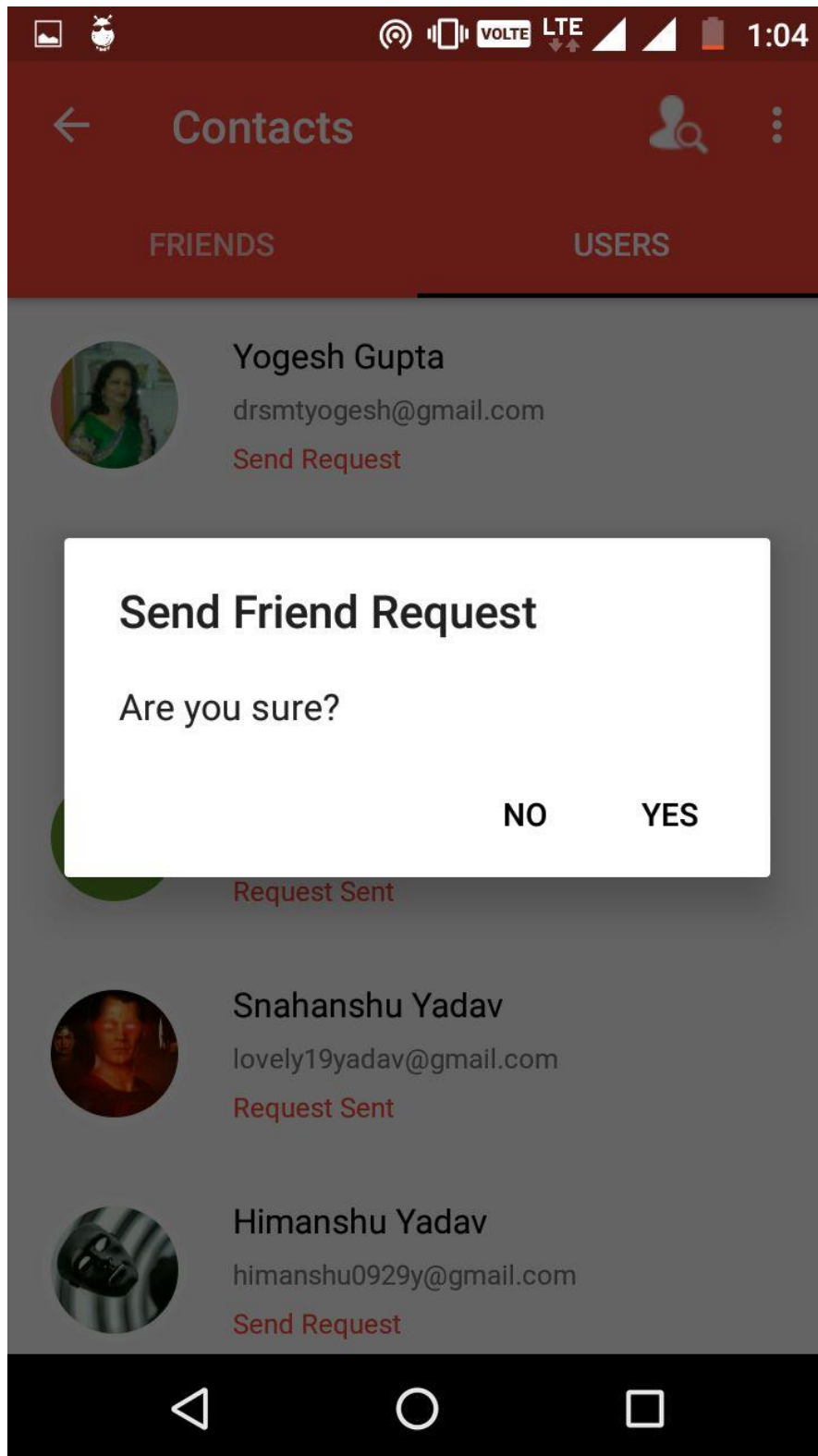
9. On clicking a user there are two cases

- If the user is a friend then we can view his profile or start chatting with him



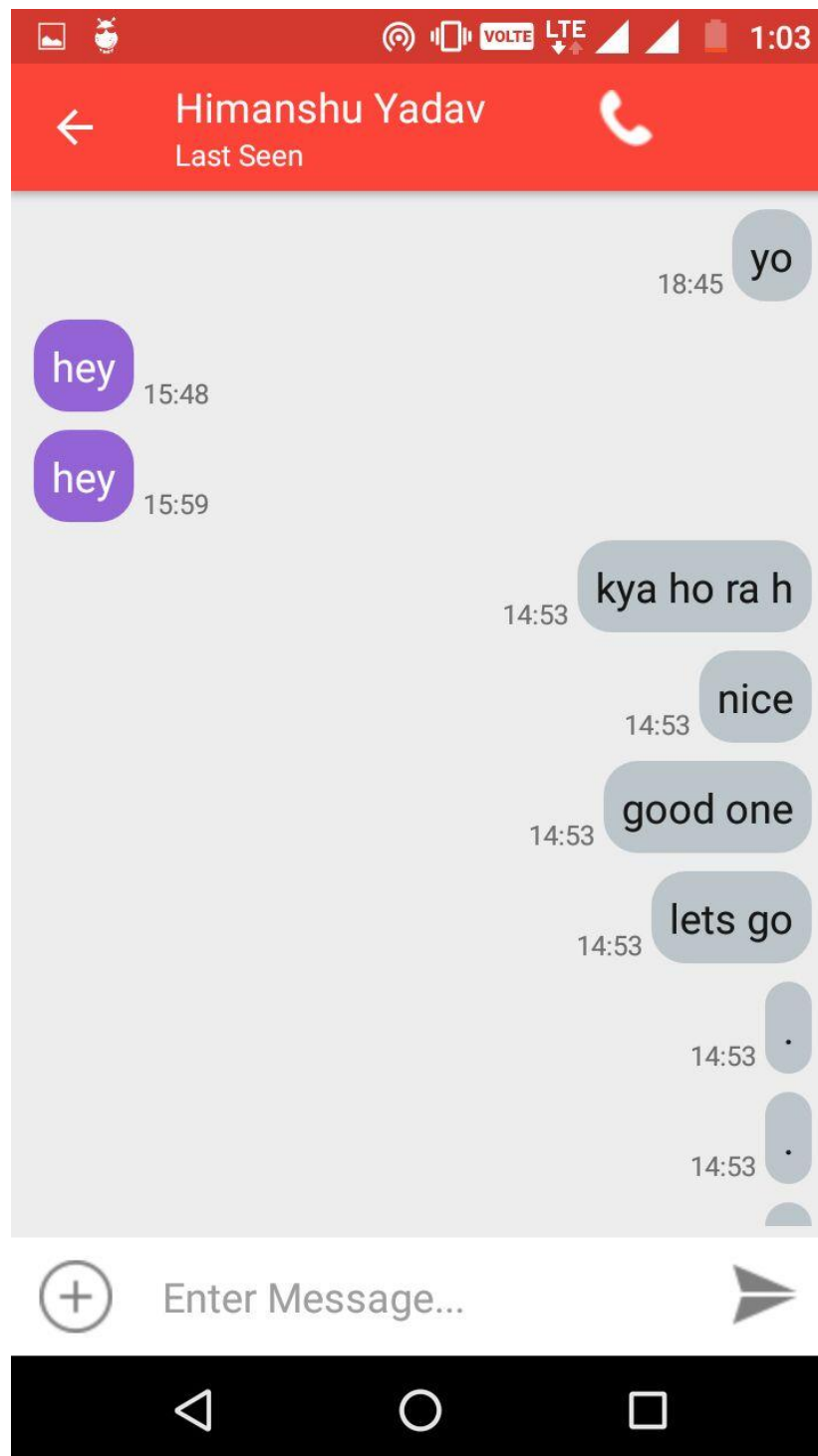
○

- If the user is not a friend in the app , then we can view his public profile and send him friend request

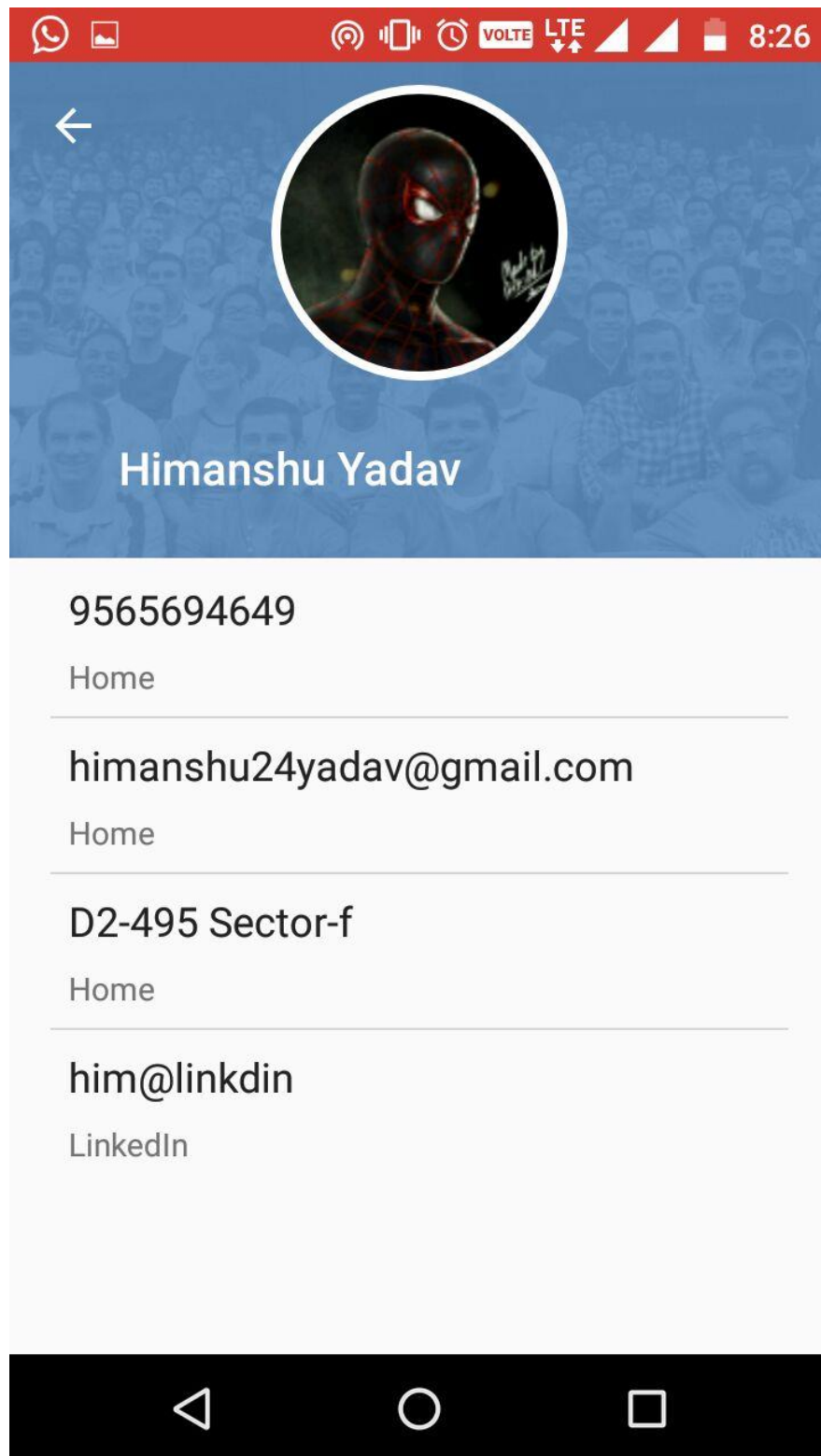


○

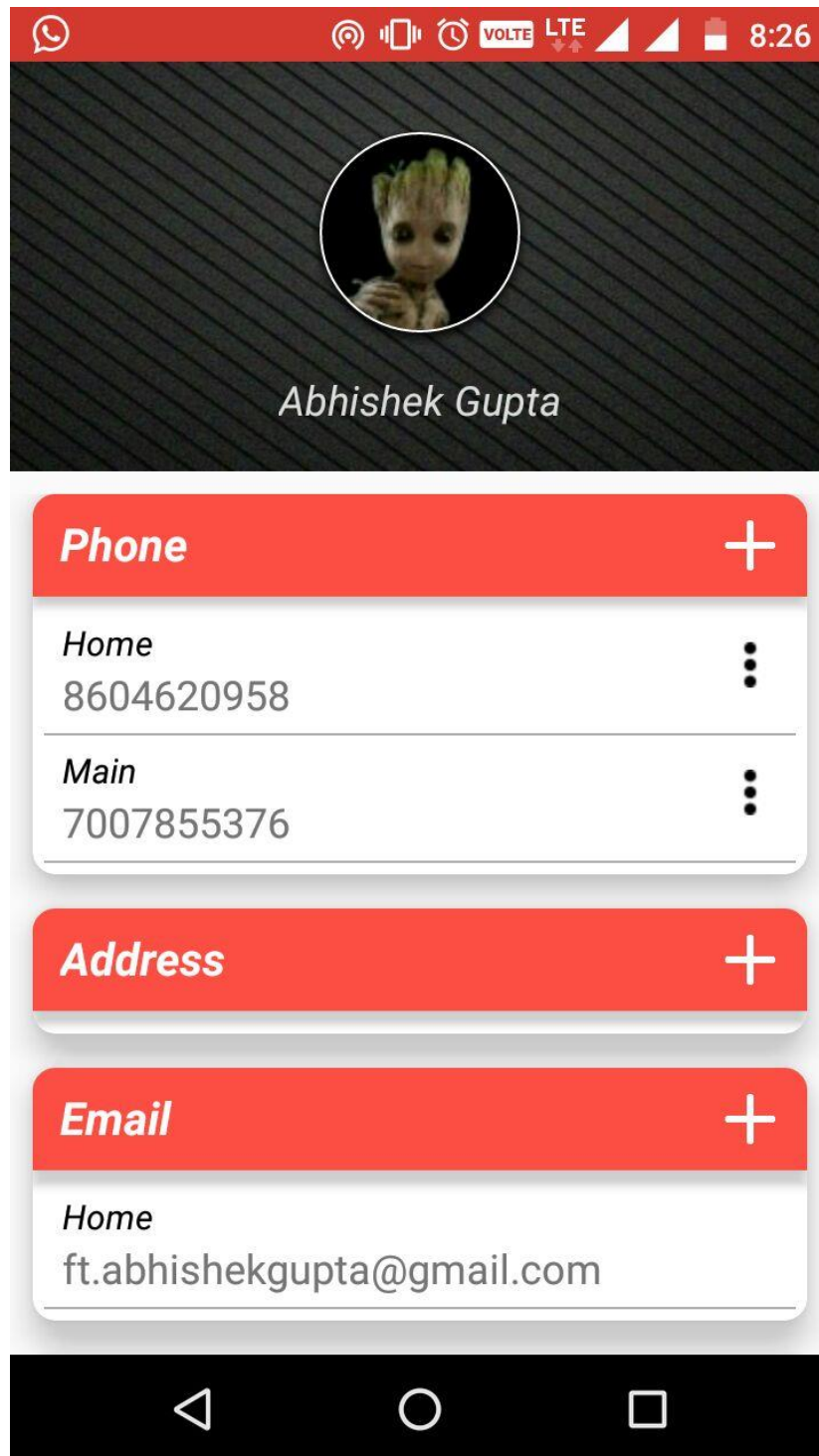
10. In the chat window , we can do text based e-messaging. As the app is using dynamic database , streamless chatting is established



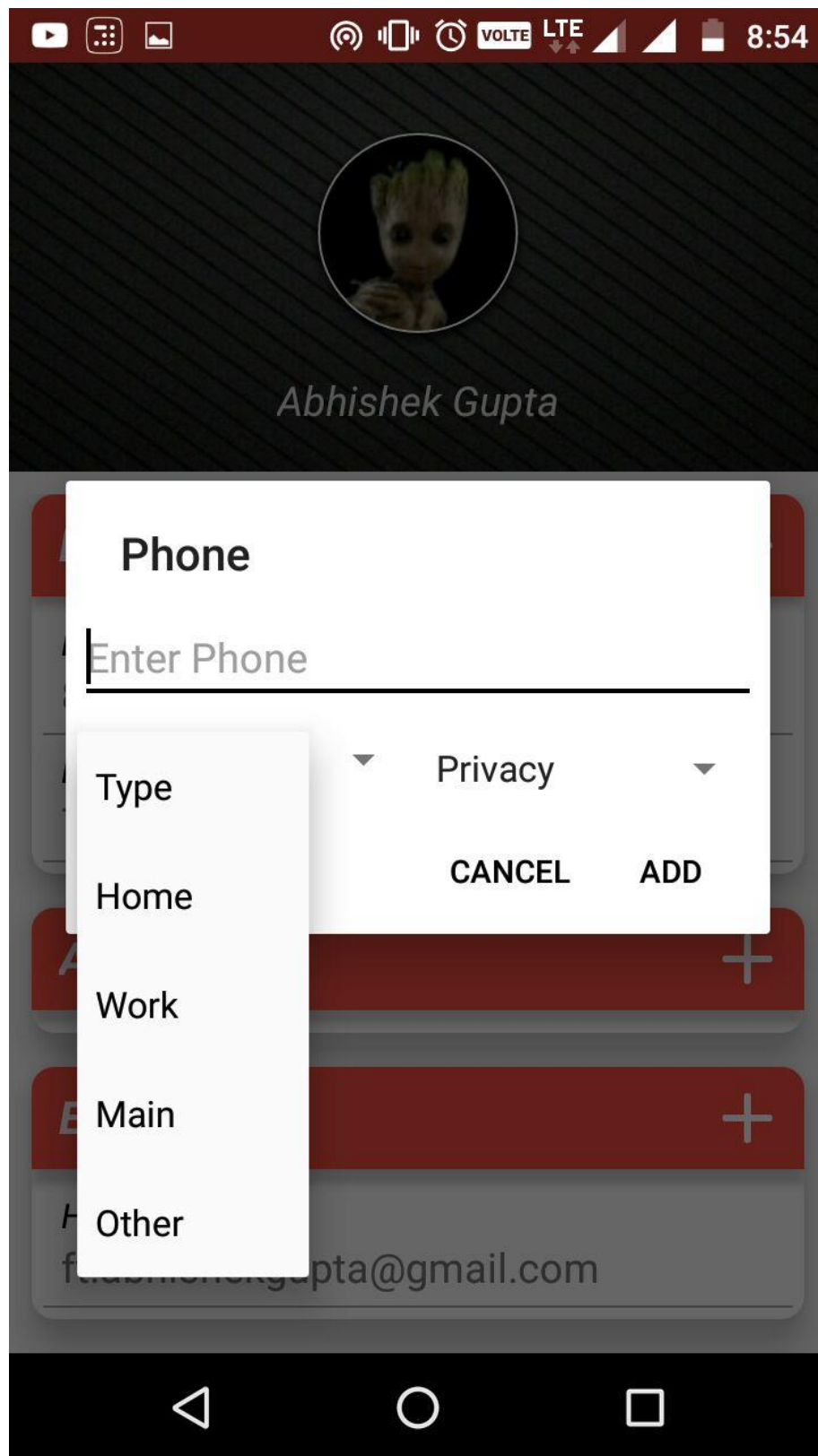
11. When Opening a users profile , we can view the dynamic links that he made visible to us. Those link can be clicked upon for further actions like , opening call dialog for phone numbers or opening browser with url as different social handles of the user.



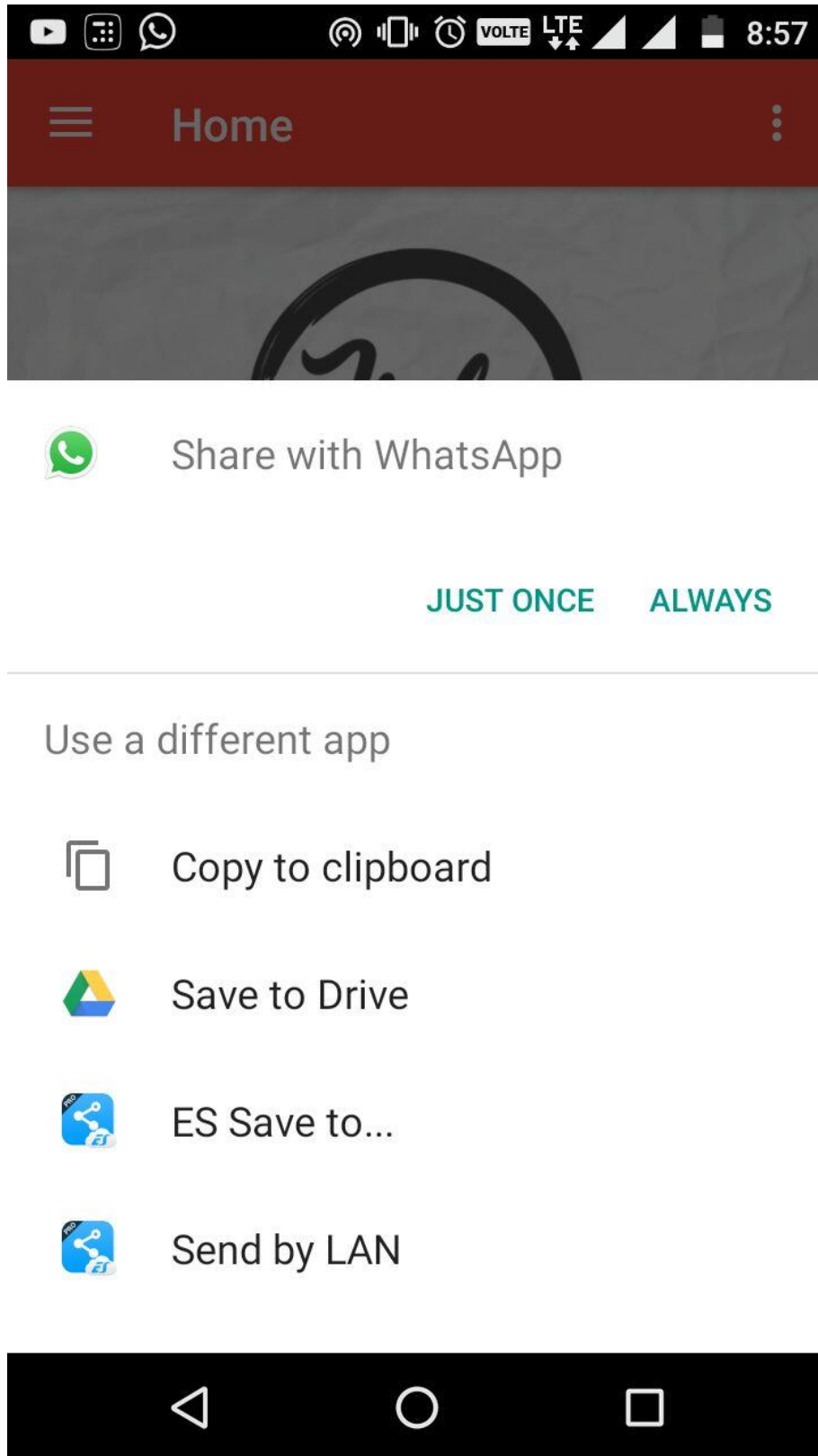
12. The edit profile window gives user the access to manage his/her profile. Each field in the profile can be customised with editable data and restricted privacy settings



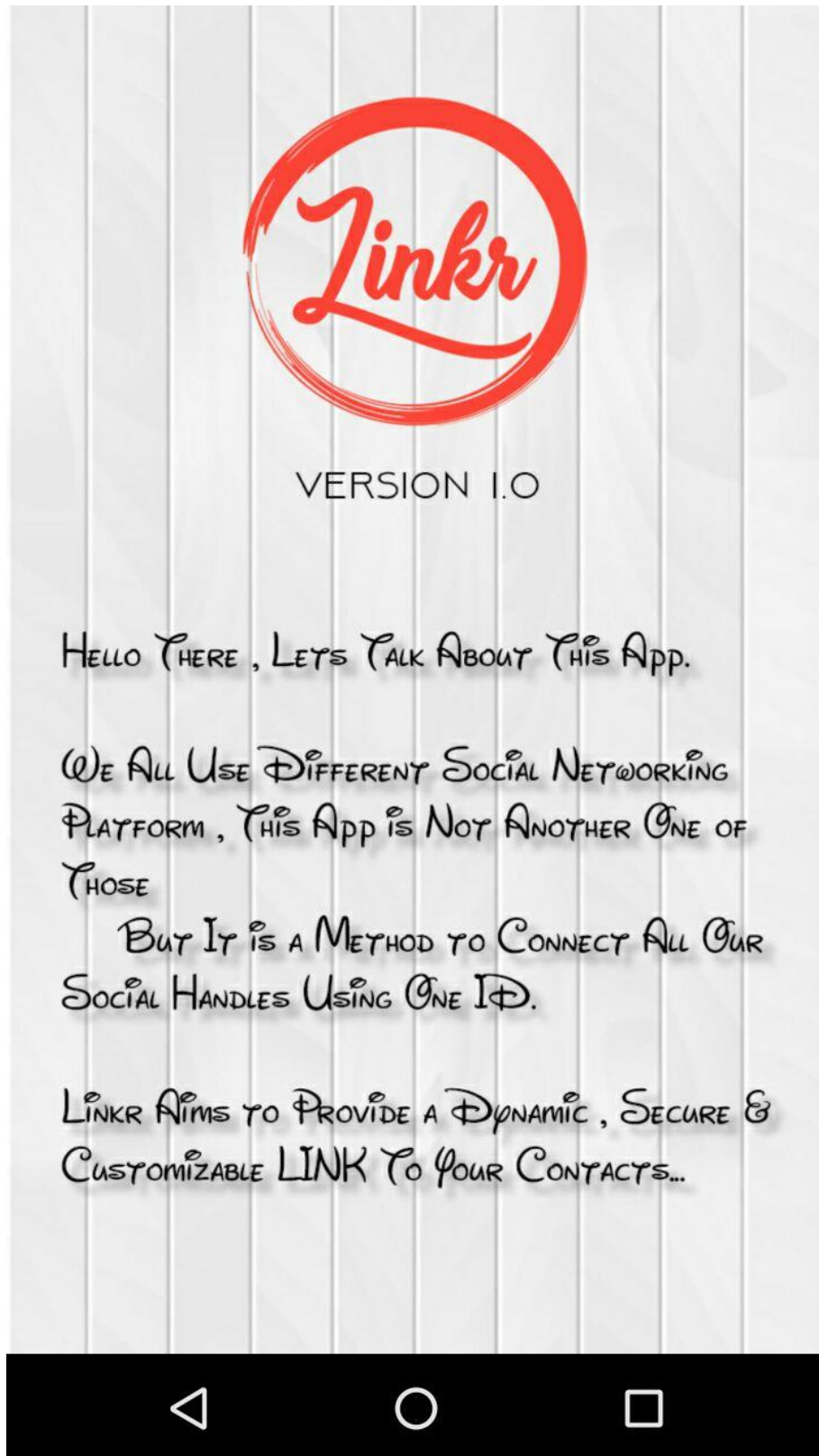
13. Plus sign can be clicked upon to add fields using a dialog box where we can set the type of data and set privacy settings



14. Share Button is provided in the navigation drawer to easily share the app and increase your network



15. About Sections gives a quick overview of the app and its current version



16. Developer Sections gives a brief description of the developers of the app

DEVELOPERS



HIMANSHU YADAV
CSE FINAL YEAR
IET LUCKNOW
himanshu24yadav@gmail.com



ABHISHEK GUPTA
CSE FINAL YEAR
IET LUCKNOW
ft.abhishekgupta@gmail.com



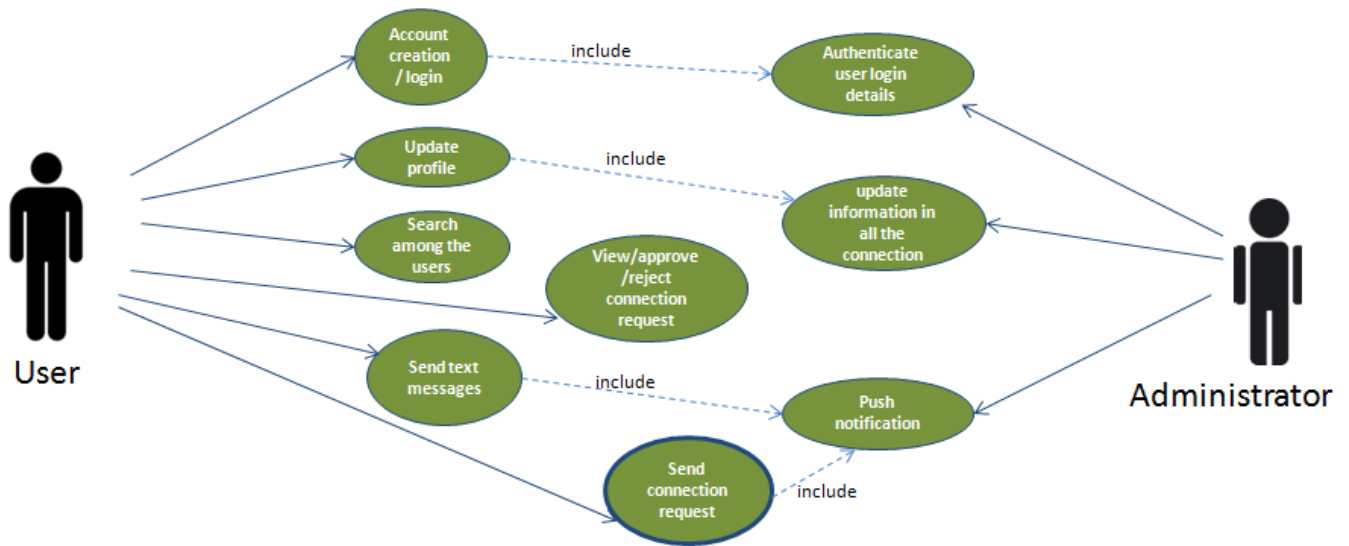
AAKANKSHA VERMA
CSE FINAL YEAR
IET LUCKNOW
cse.aakanksha@gmail.com

UNDER GUIDANCE OF
PROFF. M. H. KHAN (IET LUCKNOW)
MR. RITENDRA GOYAL (IET LUCKNOW)

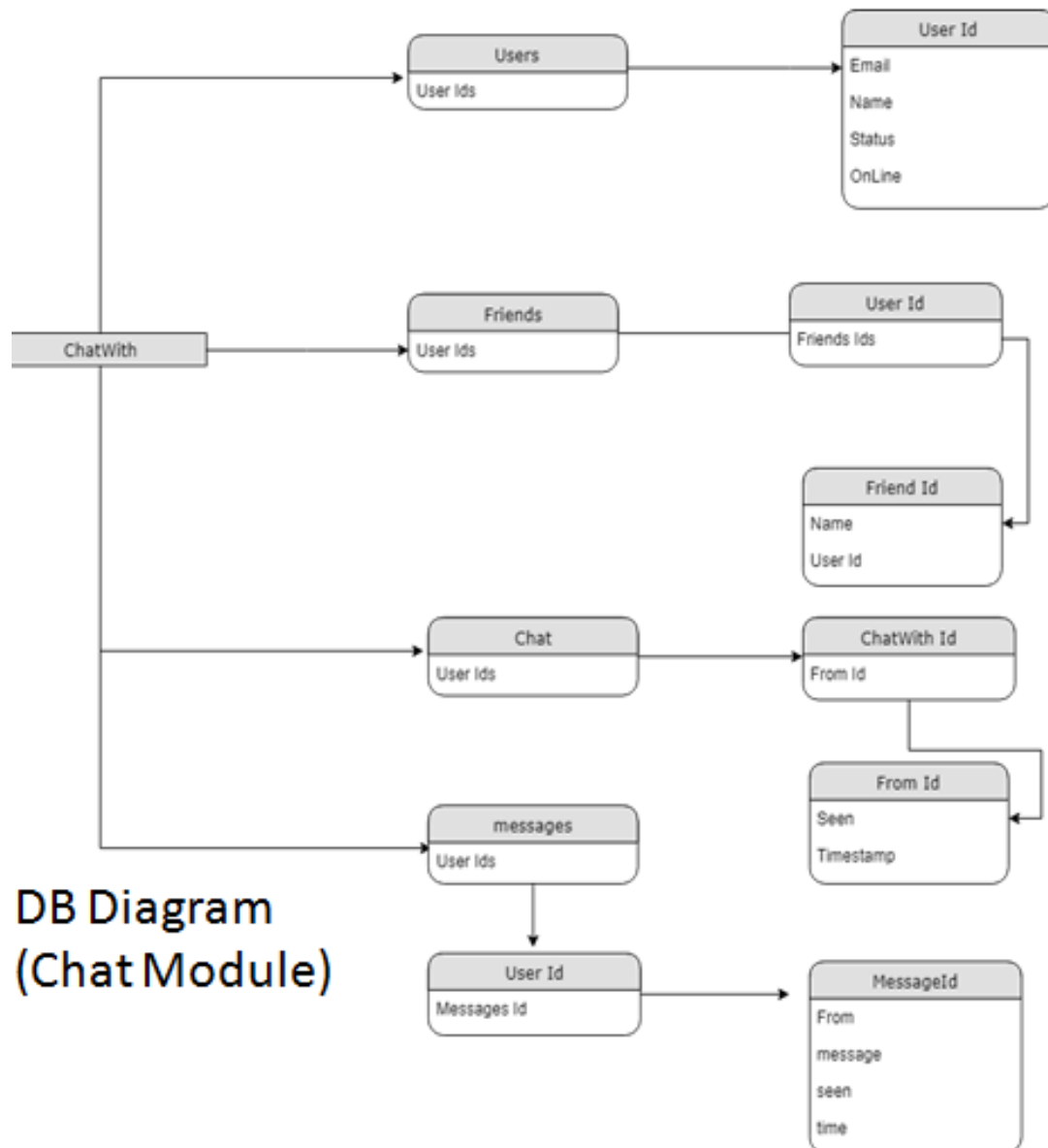
o

6. Low and High Level Design

Use case diagram



Use case diagram, a part of UML diagrams needed to demonstrate the different ways that a user might interact with the system. Same goes for the administrator. The above use case diagram for Linkr models the basic flow of events.

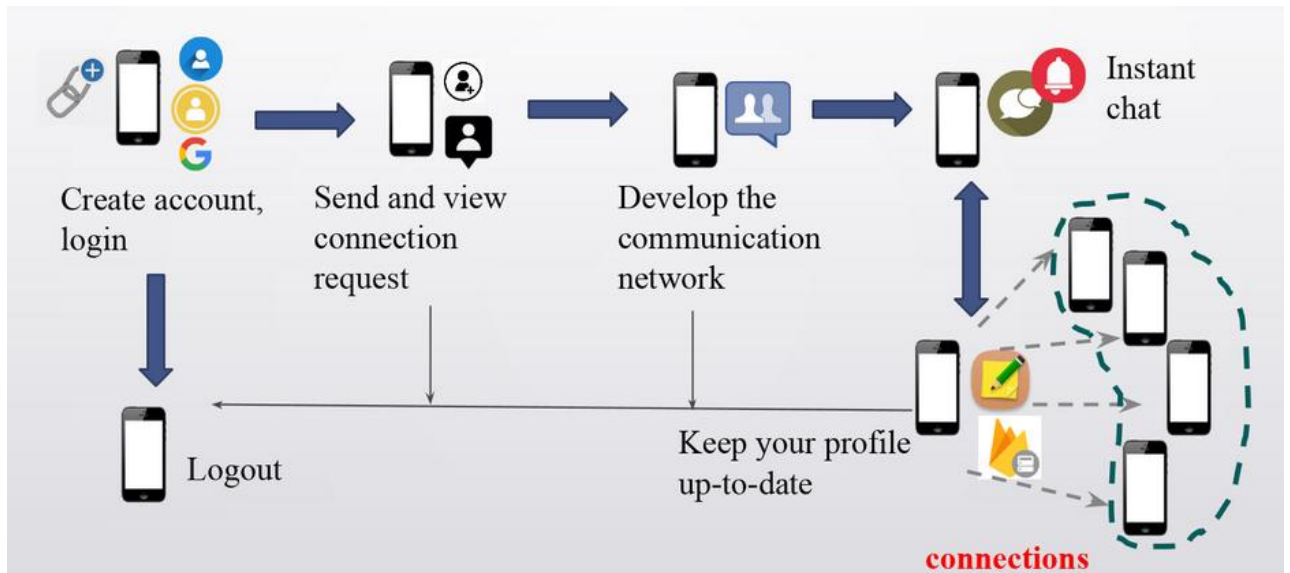


The database model shows the logical structure of a database, including the relationships and constraints that determine how data can be stored and accessed. A data model is in general represented by the accompanying database diagram shown above.

7. Classes Used in App

- Ⓢ About
- Ⓢ AdapterFriendList
- Ⓢ AdapterMessageList
- Ⓢ AdapterPendingRequests
- Ⓢ AdapterProfileData
- Ⓢ AdapterUserList
- Ⓢ AdapterUsersProfile
- Ⓢ ChatAct
- Ⓢ Contacts
- Ⓢ Developer
- Ⓢ EditProfile
- Ⓢ FilterFriendList
- Ⓢ FilterUserList
- Ⓢ home
- Ⓢ Login
- Ⓢ MainActivity
- Ⓢ ModelDetailedInfo
- Ⓢ ModelFriendRequest
- Ⓢ ModelMessage
- Ⓢ ModelUserProfile
- Ⓢ PagerContacts
- Ⓢ PagerPendingRequests
- Ⓢ PendingRequests
- Ⓢ TabFriendList
- Ⓢ TabRequestRecieved
- Ⓢ TabRequestSent
- Ⓢ TabUserList
- Ⓢ UpdateProfile
- Ⓢ UserProfile

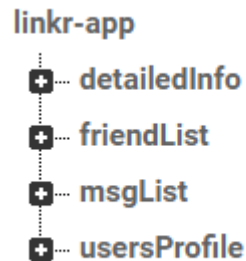
8. Workflow



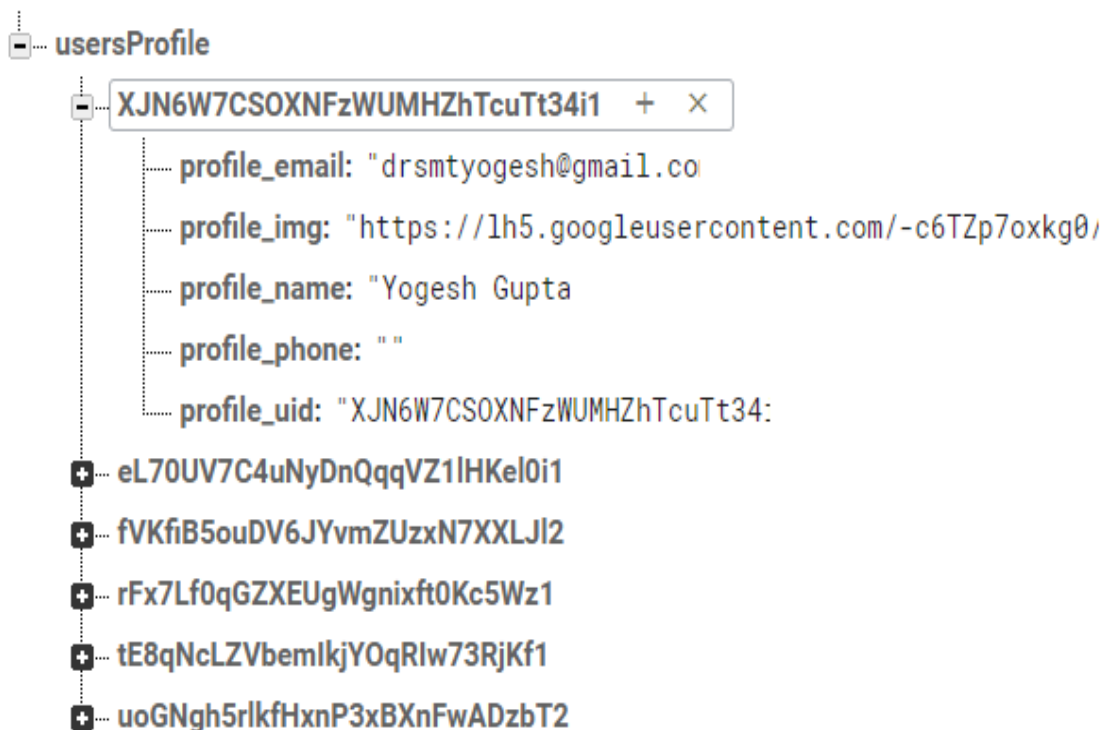
The above figure illustrates the workflow considering some scenarios possible in the execution of the application.

9. Database Hierarchy

Linkr contains four basic modules which are then subdivided to provide other useful applications.



- **Module usersProfile:** Contains basic details of each user such as:
 1. profile_email: Contains Email of the user to contact
 2. profile_img: Contains link to the profile image of user
 3. profile_name: Username displayed on the application
 4. profile_phone: Contains contact details
 5. profile_uid: Unique key of the user in the database



- **Module detailedInfo :** Contains other information about each user such as email, contact details, address and links.

Each user has four other details sub-modules

1. other_emails : Contains emails added by user other than its primary email privacy level provided.
2. other_address: Contains address details of users privacy level provided.
3. other_phone: Contains contact details added by user privacy level provided.
4. other_link: Contains social links with its privacy level provided



- **Module friendList:** Contains details of friends of each user with their details such as status of their friendship.

Each friend of user contains information such as user_email , unique key, user name and user status.



- **Module msgList:** Contains details of messages sent and received by each user with timestamp.

Each message that is sent by the user has unique id provide by the database administrator and each of them contains details such as:

1. sent_by: Contains unique key of user who has sent the message.
2. text: message written.
3. time: Contains timestamp of the message(time and date).



10. Codes :

Login.java

```
package com.thesocialnetwork.linkr;

import android.content.Intent;

import android.os.Bundle;

import android.support.annotation.NonNull;

import android.support.v7.app.AppCompatActivity;

import android.util.Log;

import android.view.View;

import android.view.Window;

import android.view.WindowManager;

import android.view.animation.Animation;

import android.view.animation.AnimationUtils;

import android.widget.ImageView;

import android.widget.Toast;

import com.google.android.gms.auth.api.Auth;

import com.google.android.gms.auth.api.signin.GoogleSignInAccount;

import com.google.android.gms.auth.api.signin.GoogleSignInOptions;

import com.google.android.gms.auth.api.signin.GoogleSignInResult;

import com.google.android.gms.common.api.GoogleApiClient;

import com.google.android.gms.tasks.OnCompleteListener;

import com.google.android.gms.tasks.Task;

import com.google.firebase.auth.AuthCredential;

import com.google.firebase.auth.AuthResult;

import com.google.firebase.auth.FirebaseAuth;
```

```

import com.google.firebase.auth.FirebaseAuth;

import com.google.firebase.auth.GoogleAuthProvider;

import com.google.firebase.database.DataSnapshot;

import com.google.firebase.database.DatabaseError;

import com.google.firebase.database.DatabaseReference;

import com.google.firebase.database.FirebaseDatabase;

import com.google.firebase.database.ValueEventListener;


public class Login extends AppCompatActivity implements View.OnClickListener{

    private static final String TAG = "SignInActivity";

    private static final int RC_SIGN_IN = 9001;

    private FirebaseAuth mAuth;

    private GoogleApiClient mGoogleApiClient;

    private FirebaseAuth.AuthStateListener mAuthListener;

    private DatabaseReference du;

    Animation anim;

    ImageView logo;


    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        requestWindowFeature(Window.FEATURE_NO_TITLE);


        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,Window
        Manager.LayoutParams.FLAG_FULLSCREEN);

        setContentView(R.layout.activity_login);

        getSupportActionBar().hide();

```

```
        GoogleSignInOptions gso = new
        GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN).requestEmail().r
        equestIdToken(getString(R.string.default_web_client_id)).build();
```

```
        mGoogleApiClient = new GoogleApiClient.Builder(this).enableAutoManage(this /*
        FragmentActivity */, null /* OnConnectionFailedListener
        */).addApi(Auth.GOOGLE_SIGN_IN_API, gso).build();
```

```
        findViewById(R.id.sign_in_button).setOnClickListener(this);
```

```
        mAuth = FirebaseAuth.getInstance();
```

```
        du= FirebaseDatabase.getInstance().getReference().child("usersProfile");
```

```
        du.keepSynced(true);
```

```
        mAuthListener = new FirebaseAuth.AuthStateListener() {
```

```
            @Override
```

```
            public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
```

```
                FirebaseUser user = firebaseAuth.getCurrentUser();
```

```
                if (user != null) {
```

```
                    Log.d(TAG, "onAuthStateChanged:signed_in:" + user.getId());
```

```
                } else {
```

```
                    Log.d(TAG, "onAuthStateChanged:signed_out");
```

```
                }
```

```
            }
```

```
        };
```

```
        mAuth.addAuthStateListener(mAuthListener);
```

```
    }
```

```
    @Override
```

```
    public void onStart() {
```

```

        super.onStart();

        logo= (ImageView) findViewById(R.id.imageView);

        anim = AnimationUtils.loadAnimation(getApplicationContext(),R.anim.fadein);

        logo.startAnimation(anim);

    }

    @Override

    protected void onDestroy() {

        super.onDestroy();

        System.gc();

        Runtime.getRuntime().gc();

    }

    @Override

    public void onBackPressed() {

        super.onBackPressed();

        System.exit(0);

    }


    @Override

    public void onStop() {

        super.onStop();

        if (mAuthListener != null) {

            mAuth.removeAuthStateListener(mAuthListener);

        }

    }

    @Override

```

```

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.sign_in_button:
            {
                FirebaseUser currentUser = mAuth.getCurrentUser();
                if(currentUser!=null) checkUserExist();
                else signIn();
                break;
            }
    }
}

private void signIn() {
    Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);
    startActivityForResult(signInIntent, RC_SIGN_IN);
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == RC_SIGN_IN) {
        GoogleSignInResult result =
Auth.GoogleSignInApi.getSignInResultFromIntent(data);
        handleSignInResult(result);
        if (result.isSuccess()) {
            GoogleSignInAccount account = result.getSignInAccount();
            firebaseAuthWithGoogle(account);
        } else {

```

```

    }
}

private void handleSignInResult(GoogleSignInResult result) {
    Log.d(TAG, "handleSignInResult:" + result.isSuccess());
    if (result.isSuccess()) {
        GoogleSignInAccount acct = result.getSignInAccount();
    } else {
    }
}

public void updateUI(boolean ans){
    if(ans){
        Intent i=new Intent(this,MainActivity.class);
        finish();
        startActivity(i);
    }
}

private void firebaseAuthWithGoogle(GoogleSignInAccount acct) {
    Log.d(TAG, "firebaseAuthWithGoogle:" + acct.getId());
    AuthCredential credential = GoogleAuthProvider.getCredential(acct.getIdToken(), null);
    mAuth.signInWithCredential(credential)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "signInWithCredential:onComplete:" + task.isSuccessful());
                if (!task.isSuccessful()) {

```

```

        Log.w(TAG, "signInWithCredential", task.getException());

        Toast.makeText(Login.this, "Authentication failed.",
                        Toast.LENGTH_SHORT).show();

    }else

        checkUserExist();

    }

});

}

private void checkUserExist() {

    if (mAuth.getCurrentUser()!=null)

    {

        final String uid=mAuth.getCurrentUser().getUid();

        du.addListenerForSingleValueEvent(new ValueEventListener() {

            @Override

            public void onDataChange(DataSnapshot dataSnapshot) {

                if (dataSnapshot.hasChild(uid))

                {

                    updateUI(true);

                }

                else

                {

                    Intent i = new Intent(Login.this, UpdateProfile.class);

                    finish();

                    startActivity(i);

                }

            }

        }

    }

}

```

```

        @Override

        public void onCancelled(DatabaseError databaseError) {

        }

    });}}

}

activity_login.XML

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:background="@drawable/background"

    tools:context="com.thesocialnetwork.linkr.Login">

    <com.google.android.gms.common.SignInButton

        android:id="@+id/sign_in_button"

        android:layout_width="111dp"

        android:layout_height="48dp"

        android:layout_alignParentBottom="true"

        android:layout_centerHorizontal="true"

        android:layout_marginBottom="38dp" />

    <ImageView

        android:id="@+id/imageView"

```



```
android:layout_width="200dp"  
android:layout_height="250dp"  
android:layout_alignParentTop="true"  
android:layout_centerHorizontal="true"  
android:layout_marginTop="47dp"  
app:srcCompat="@drawable/logotext" />
```

```
</RelativeLayout>
```

MainActivity.java

```
package com.thesocialnetwork.linkr;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.NavigationView;
import android.support.v4.app.FragmentManager;
import android.support.v4.view.GravityCompat;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarDrawerToggle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
import com.bumptech.glide.Glide;
import com.bumptech.glide.load.engine.DiskCacheStrategy;
import com.google.android.gms.auth.api.Auth;
import com.google.android.gms.auth.api.signin.GoogleSignInOptions;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.ResultCallback;
import com.google.android.gms.common.api.Status;
```

```

import com.google.firebase.auth.FirebaseAuth;

import com.google.firebase.database.DataSnapshot;

import com.google.firebase.database.DatabaseError;

import com.google.firebase.database.DatabaseReference;

import com.google.firebase.database.FirebaseDatabase;

import com.google.firebase.database.ValueEventListener;

import com.mikhaellopez.circularimageview.CircularImageView;


import java.util.ArrayList;


public class MainActivity extends AppCompatActivity
    implements NavigationView.OnNavigationItemSelectedListener {


    private GoogleApiClient GoogleApiClient;

    private DatabaseReference databaseReference;

    private FirebaseAuth FirebaseAuth;

    String userId="";

    TextView mName ,mEmail;

    CircularImageView mImg;

    ValueEventListener valueEventListener;

    ArrayList<ModelFriendRequest> arrayList=new ArrayList<>();

    String currentUserKey;

    DatabaseReference db_friendList;


    @Override

    protected void onCreate(Bundle savedInstanceState) {

```

```

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);

DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
    this, drawer, toolbar, R.string.navigation_drawer_open,
    R.string.navigation_drawer_close);

drawer.addDrawerListener(toggle);

toggle.syncState();

NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);

FirebaseAuth = FirebaseAuth.getInstance();

userId = FirebaseAuth.getCurrentUser().getUid();

currentUserKey= FirebaseAuth.getInstance().getCurrentUser().getUid();

db_friendList=
FirebaseDatabase.getInstance().getReference().child("friendList").child(currentUserKey);

FragmentManager fragmentManager = getSupportFragmentManager();

fragmentManager.beginTransaction().replace(R.id.content_frame, new
home()).commit();

View header = navigationView.getHeaderView(0);

mName = header.findViewById(R.id.textview1);

```

```

        mEmail = header.findViewById(R.id.textView2);

        mImg = header.findViewById(R.id.circleImageView);

        databaseReference =
FirebaseDatabase.getInstance().getReference().child("usersProfile").child(userId);

        databaseReference.keepSynced(true);

        countRequests();
    }

    private void countRequests() {

        db_friendList.addValueEventListener(new ValueEventListener() {

            @Override

            public void onDataChange(DataSnapshot dataSnapshot) {

                arrayList.clear();

                for (DataSnapshot ds : dataSnapshot.getChildren())

                {

                    if(dataSnapshot.child(ds.getKey()).child("user_status").getValue().toString().equals("request
Recieved") ||
dataSnapshot.child(ds.getKey()).child("user_status").getValue().toString().equals("requestSe
nt"))

                    {

                        ModelFriendRequest model=ds.getValue(ModelFriendRequest.class);

                        arrayList.add(model);

                    }

                }

                if(arrayList.size()>0)

                    Toast.makeText(MainActivity.this, "You have " +arrayList.size()+" pending
requests", Toast.LENGTH_SHORT).show();

                    //showDialogBox(arrayList.size());

```

```

    }

    @Override

    public void onCancelled(DatabaseError databaseError) {

    }

});

}

private void showDialogBox(int size) {

    AlertDialog.Builder alert = new AlertDialog.Builder(

        MainActivity.this);

    alert.setTitle("Pending Requests");

    alert.setMessage("You have " +size+" pending requests. Do you want to process?");

    alert.setPositiveButton("Yes", new DialogInterface.OnClickListener() {

        @Override

        public void onClick(DialogInterface dialog, int which) {

            Intent intent = new Intent(MainActivity.this, PendingRequests.class);

            startActivity(intent);

            dialog.dismiss();

        }

    });

    alert.setNegativeButton("No", new DialogInterface.OnClickListener() {

        @Override

        public void onClick(DialogInterface dialog, int which) {

```

```

        dialog.dismiss();
    }
});

    alert.show();
}

@Override

public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}

@Override

protected void onResume() {
    ValueEventListener = new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            String name = dataSnapshot.child("profile_name").getValue().toString();
            String email = dataSnapshot.child("profile_email").getValue().toString();
            String img = dataSnapshot.child("profile_img").getValue().toString();
            mName.setText(name);

```

```

        mEmail.setText(email);

        Glide.with(getApplicationContext()).load(img)

            .thumbnail(0.5f)

            .crossFade()

            .diskCacheStrategy(DiskCacheStrategy.ALL)

            .into(mImg);
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
};

databaseReference.addValueEventListener(valueEventListener);

super.onResume();
}

@Override

public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

```



```

// Handle action bar item clicks here. The action bar will
// automatically handle clicks on the Home/Up button, so long
// as you specify a parent activity in AndroidManifest.xml.
int id = item.getItemId();

//noinspection SimplifiableIfStatement
if (id == R.id.action_logout) {
    signOut();
    return true;
} else if(id==R.id.action_exit){
    System.exit(0);
}

return super.onOptionsItemSelected(item);
}

private void signOut() {
    Auth.GoogleSignInApi.signOut(GoogleApiClient).setResultCallback(
        new ResultCallback<Status>() {
            @Override
            public void onResult(Status status) {
                FirebaseAuth.getInstance().signOut();
                Intent i=new Intent(MainActivity.this,Login.class);
                finish();
                startActivity(i);
            }
        });
}

```

```

    }

    @SuppressWarnings("StatementWithEmptyBody")
    @Override
    public boolean onNavigationItemSelected(MenuItem item) {

        // Handle navigation view item clicks here.

        int id = item.getItemId();

        if (id == R.id.nav_about) {

            Intent i=new Intent(this,About.class);

            startActivity(i);

        }else if (id == R.id.nav_dev) {

            Intent i=new Intent(this,Developer.class);

            startActivity(i);

        }else if(id==R.id.nav_share){

            final String appPackageName = getApplicationContext().getPackageName();

            Intent sendIntent = new Intent();

            sendIntent.setAction(Intent.ACTION_SEND);

            sendIntent.putExtra(Intent.EXTRA_TEXT, "Linkr - Profile Linking Android App :
https://play.google.com/store/apps/details?id= + appPackageName);

            sendIntent.setType("text/plain");

            startActivity(sendIntent);

        }

        else if(id==R.id.nav_edit){

            Intent intent = new Intent(this, EditProfile.class);

            startActivity(intent);

            return true;

```

```

    }

    else if(id==R.id.nav_contacts){

        Intent intent = new Intent(this, Contacts.class);

        startActivity(intent);

        return true;

    }


    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

    drawer.closeDrawer(GravityCompat.START);

    return true;

}

@Override

protected void onStart() {

    GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)

        .requestEmail()

        .build();

    GoogleApiClient = new GoogleApiClient.Builder(this)

        .addApi(Auth.GOOGLE_SIGN_IN_API, gso)

        .build();

    GoogleApiClient.connect();

    super.onStart();

}

}

```

home.XML

```

<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.thesocialnetwork.linkr.home"
    tools:showIn="@layout/app_bar_main">

    <RelativeLayout

        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <com.sdsmgd.harjot.rotatingtext.RotatingTextWrapper

            android:id="@+id/custom_switcher"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_centerHorizontal="true"
            android:paddingBottom="15dp" />

        <pl.droidsonroids.gif.GifTextView

            android:background="@drawable/gif"
            android:layout_width="170dp"

```

```
android:layout_height="30dp"
android:layout_alignParentBottom="true"
app:srcCompat="@mipmap/ic_launcher"
android:layout_centerHorizontal="true"
android:id="@+id/imageView2"
android:layout_marginBottom="15dp" />
```

<ImageView

```
android:layout_width="match_parent"
android:layout_height="200dp"
android:padding="25dp"
android:src="@drawable/logo"/>
```

<LinearLayout

```
android:layout_width="270dp"
android:layout_height="match_parent"
android:orientation="vertical"
android:layout_centerHorizontal="true"
android:weightSum="5"
android:layout_marginBottom="60dp"
android:layout_marginTop="200dp">
```

<TextView

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/hContacts"
android:layout_weight="1"
android:text="Contacts"
```

```

        android:textSize="30dp"
        android:textAlignment="center"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/hChat"
    android:layout_weight="1"
    android:text="Chat"
    android:textSize="30dp"
    android:textAlignment="center"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/hRequests"
    android:layout_weight="1"
    android:text="Requests"
    android:textSize="30dp"
    android:textAlignment="center"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/hProfile"
    android:layout_weight="1"
    android:text="Profile"
    android:textSize="30dp"
    android:textAlignment="center"/>

```

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/hAbout"  
    android:layout_weight="1"  
    android:text="About"  
    android:textSize="30dp"  
    android:textAlignment="center"/>
```

```
</LinearLayout>
```

```
</RelativeLayout>
```

```
</android.support.constraint.ConstraintLayout>
```

11. Conclusion

Linkr has been developed to make people's life easier. It is free from any complexity of usage, size overhead, functions perfectly fine even in slow data connection and is freely available and easily downloadable from the Google Play Store within seconds.

Linkr lets its users integrate, collaborate, share and update their contact information in real-time which increases its usefulness in this world where change is the only constant.

Linkr links all your accounts at one place and also links you to all your cared ones hence, indeed acting as a **link to your life!**

To summarise linkr is:

**“Simple enough that anyone can use it,
versatile enough that everyone has some
use for it.”**

12. References

- <https://firebase.google.com/docs/storage/android/start>
- <https://developer.android.com/studio/write/firebase>
- <https://firebase.google.com/docs/cloud-messaging/>
- <https://firebase.google.com/docs/auth/>
- <https://firebase.google.com/docs/auth/android/google-signin>
- <https://firebase.google.com/docs/database/android/offline-capabilities>
- <https://firebase.google.com/docs/database/>
- <https://developer.android.com/studio/workflow>
- <https://developer.android.com/reference/android/support/v7/widget/RecyclerView>
- <https://developer.android.com/guide/components/fragments>