



Graphic Era HILL UNIVERSITY

Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

BHIMTAL CAMPUS



REPORT FILE

Mini Project

ON

“Notes and Password Manager app”

B.TECH-IIIrd

(2021-2022)

SUBMITTED TO -

MR. RAVINDRA KORANGA
Assistant Professor
(Dept. of CSE)

SUBMITTED BY-

PRAMOD JOSHI
Mukul Kandpal
Hem Upadhyay



Graphic Era HILL UNIVERSITY

Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

BHIMTAL CAMPUS

THIS IS TO CERTIFY THAT MR./MS. PRAMOD JOSHI,
Mukul Kandpal and Hem Upadhyay HAS SATISFACTORILY
COMPLRTED ALL THE EXPERIMENTS IN THE
LABORATORY OF THIS COLLEGE. THE EXPERIMENTS /
TEAM WORK NOTES AND PASSWORD MANAGER APP IN
PARTIAL FULLFILLMENTS OF THE REQUIREMENT IN
THIRD OF B.TECH.()/ M.TECH()/ BCA()/ MCA / BBA /
MBA DEGREE COURSE PRESCRIBED BY GRAHIC ERA HILL
UNIVERSITY, DEHRADUN DURING THE YEAR .

CONCERNED FACULTY

HEAD OF DEPATMENT

NAME OF EXAMINER

SIGNATURE OF EXAMINER

Notes & Password Manager App

Overview

Objective

You will be creating an android application which helps to take notes at any time and helps to generate a very strong password for the security of your account and makes them available whenever you need. It's going to have two major functionalities at the same time.

Project Context

Password managing & note-taking are one of the most overlooked virtual activities in today's digital age. Various companies are constantly adding new features to note-taking and password management. Have you ever considered building your own note-taking and password managing application?

So, instead of having and maintaining two different apps in your system, we'll be developing a single app which can perform both of these activities.

Project Stages

The project consists of the following stages:



High-Level Approach:

- The first task, once we get the development environment ready, will be to set up **Android Studio & Firebase**.
- Once we have everything in place, we can start off with creating the application, which will basically start with the login authentication.

- Next up is the home page building. In this project, we'll be keeping it simple by showing two options i.e Notes & Password

- In our project, we'll need to manage three states: **Note** (to manage notes data), **Password** (to manage password data), and **user** (for managing the details of the currently logged-in user).

- For showing all the notes or passwords, we'll be using **List views or recycle views** .

- Handling our database and authentication needs to be supported and we'll be using **Firestore** for the same. Basically, the database will be used to store the login information for the users, but the resource can be used for storing notes and password information as well.

- Successful implementation of the above requirements will lead to the completion of the core implementation of our application. Next up, deploy!

Primary goals:

- Create a login page for user login.
- Create a home page to display notes & password options.
- Create a note page to display all notes.
- Create a password page to display all passwords.
- Add functionalities like add note or password, delete note or password, and update note (if required).

Task 1:

Environment & Firebase setup:

Before the start of any development procedure, we need to set up the environment according to our application needs. Then connect your Android Studio with Firebase.

Requirements:

- Install Android Studio on your machine.
- Install and set up JDK.
- Create a new Android project.
- Like any typical application, the source code of Java should be in a java folder and the source code of XML should be in a res folder.
- Connect with Firebase.

System Requirements for Installing the JDK and the JRE on 64-Bit Windows Platform:

The JDK and the JRE have minimum processor, disk space, and memory requirements for 64-bit Windows platform.

Before installing the JDK or the JRE on your 64-bit Windows platform, you must verify that it meets the following minimum processor, disk space, and memory requirements.

Processor Requirements:

Both the JDK and JRE require at minimum a Pentium 2 266 MHz processor.

Disk Space Requirements:

For JDK 10, you are given the option of installing the following features:

- Development Tools
- Source Code
- Public Java Runtime Environment

When you install 64-bit JDK, then 64-bit public JRE also gets installed. The following table provides the disk requirements for the installed features:

JDK	Installed Image
Development Tools: 64-bit platform	500 MB
Source Code	54.2 MB
JRE	Installed Image
Public Java Runtime Environment	200 MB
Java Update	2 MB

Memory Requirements

On Windows 64-bit operating systems, the Java runtime requires a minimum of 128 MB of memory.

Note:

The minimum physical RAM is required to run graphically based applications. More RAM is recommended for applets running within a browser using the Java Plug-in. Running with less memory may cause disk swapping, which has a severe effect on performance. Very large programs may require more RAM for adequate performance.

Note:

For supported processors and browsers, see [Oracle JDK Certified Systems Configurations](#).

Create an Android project: ☐

This lesson shows you how to create a new Android project with Android Studio, and it describes some of the files in the project.

To create your new Android project, follow these steps:

1. Install the latest version of Android Studio.
2. In the Welcome to Android Studio window, click Create New Project.

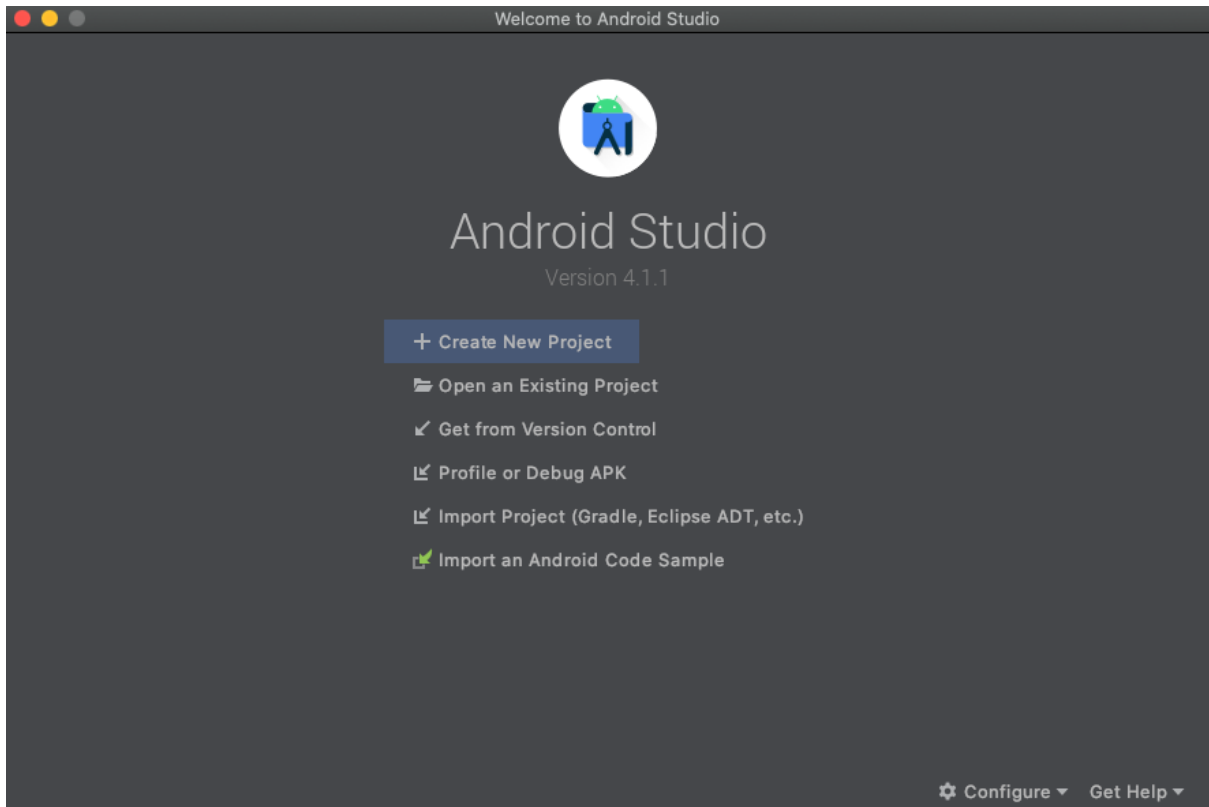


Figure 1. Android Studio welcome screen.

If you have a project already opened, select File > New > New Project.

3. In the **Select a Project Template** window, select **Empty Activity** and click **Next**.
4. In the **Configure your project** window, complete the following:
 - Enter "My First App" in the **Name** field.
 - Enter "com.example.myfirstapp" in the **Package name** field.
 - If you'd like to place the project in a different folder, change its **Save** location.
 - Select either Java or Kotlin from the Language drop-down menu.
 - Select the lowest version of Android you want your app to support in the Minimum SDK field.

Note: The **Help me choose** link opens the **Android Platform/API Version Distribution** dialog. This dialog provides information about the various versions of Android that are distributed among devices. The key tradeoff to consider is the percentage of Android devices you want to support versus the amount of work to maintain your app on each of the different versions that those devices run on. For example, if you choose to make your app compatible with many different versions of Android, you increase the effort that's required to maintain compatibility between the oldest and newest versions.

 - If your app will require legacy library support, mark the **Use legacy android.support libraries** checkbox.
 - Leave the other options as they are.
5. Click **Finish**.

After some processing time, the Android Studio main window appears.

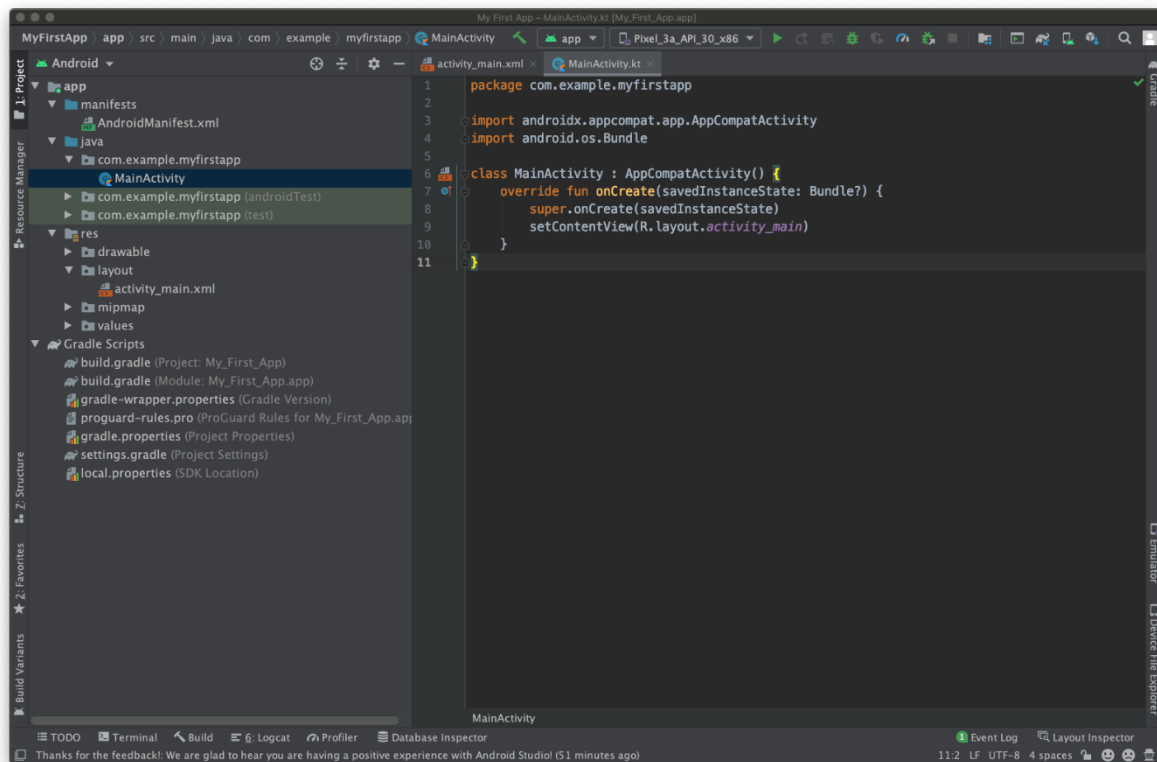


Figure 2. Android Studio main window.

Now take a moment to review the most important files.

First, be sure the **Project** window is open (select **View > Tool Windows > Project**) and the Android view is selected from the drop-down list at the top of that window. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity

This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml

This XML file defines the layout for the activity's user interface (UI). It contains a TextView element with the text "Hello, World!"

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts > build.gradle

There are two files with this name: one for the project, "Project: My_First_App," and one for the app module, "Module: My_First_App.app." Each module has its own build.gradle file, but this project currently has just one module. Use each module's build.gradle file to control

how the Gradle plugin builds your app. For more information about this file, see [Configure your build](#).

Create and manage virtual devices:



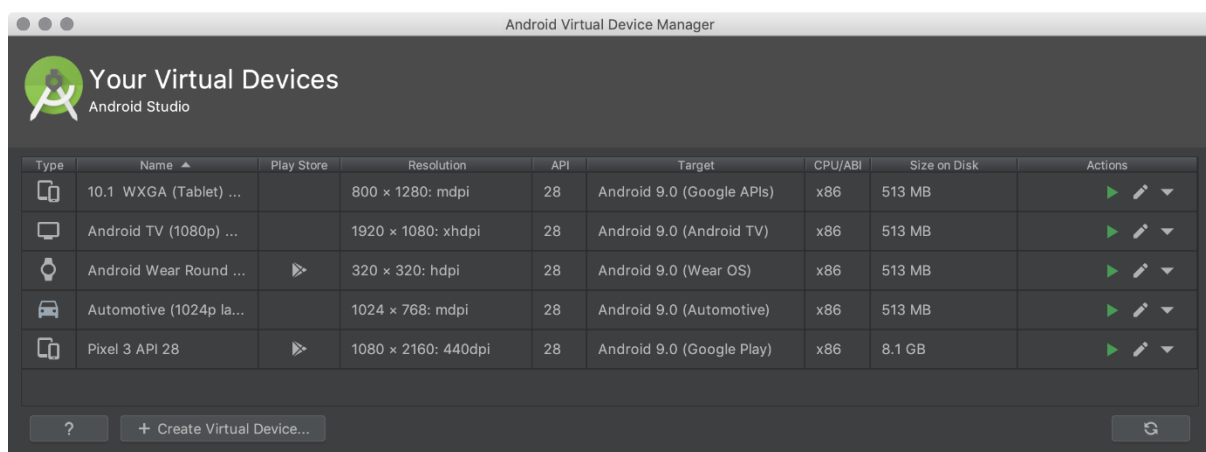
An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator. The AVD Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.

To open the AVD Manager, do one of the following:

- Select **Tools > AVD Manager**.



- Click **AVD Manager** in the toolbar.



About AVDs:

An AVD contains a hardware profile, system image, storage area, skin, and other properties.

We recommend that you create an AVD for each system image that your app could potentially support based on the [uses-sdk](#) setting in your manifest.

Hardware profile:

The hardware profile defines the characteristics of a device as shipped from the factory. The AVD Manager comes preloaded with certain hardware profiles, such as Pixel devices, and you can define or customize the hardware profiles as needed.

Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.

System images:

A system image labeled with **Google APIs** includes access to Google Play services. A system image labeled with the Google Play logo in the **Play Store** column includes the Google Play Store app and access to Google Play services, including a **Google Play** tab in the **Extended controls** dialog that provides a convenient button for updating Google Play services on the device.

To ensure app security and a consistent experience with physical devices, system images with the Google Play Store included are signed with a release key, which means that you cannot get elevated privileges (root) with these images. If you require elevated privileges (root) to aid with your app troubleshooting, you can use the Android Open Source Project (AOSP) system images that do not include Google apps or services.

Storage area:

The AVD has a dedicated storage area on your development machine. It stores the device user data, such as installed apps and settings, as well as an emulated SD card. If needed, you can use the AVD Manager to wipe user data, so the device has the same data as if it were new.

Skin

An emulator skin specifies the appearance of a device. The AVD Manager provides some predefined skins. You can also define your own, or use skins provided by third parties.

AVD and app features

Be sure your AVD definition includes the device features your app depends on. See **Hardware Profile Properties** and **AVD Properties** for lists of features you can define in your AVDs.

Create an AVD:

Tip: If you want to launch your app into an emulator, instead run your app from Android Studio and then in the **Select Deployment Target** dialog that appears, click **Create New Virtual Device**.

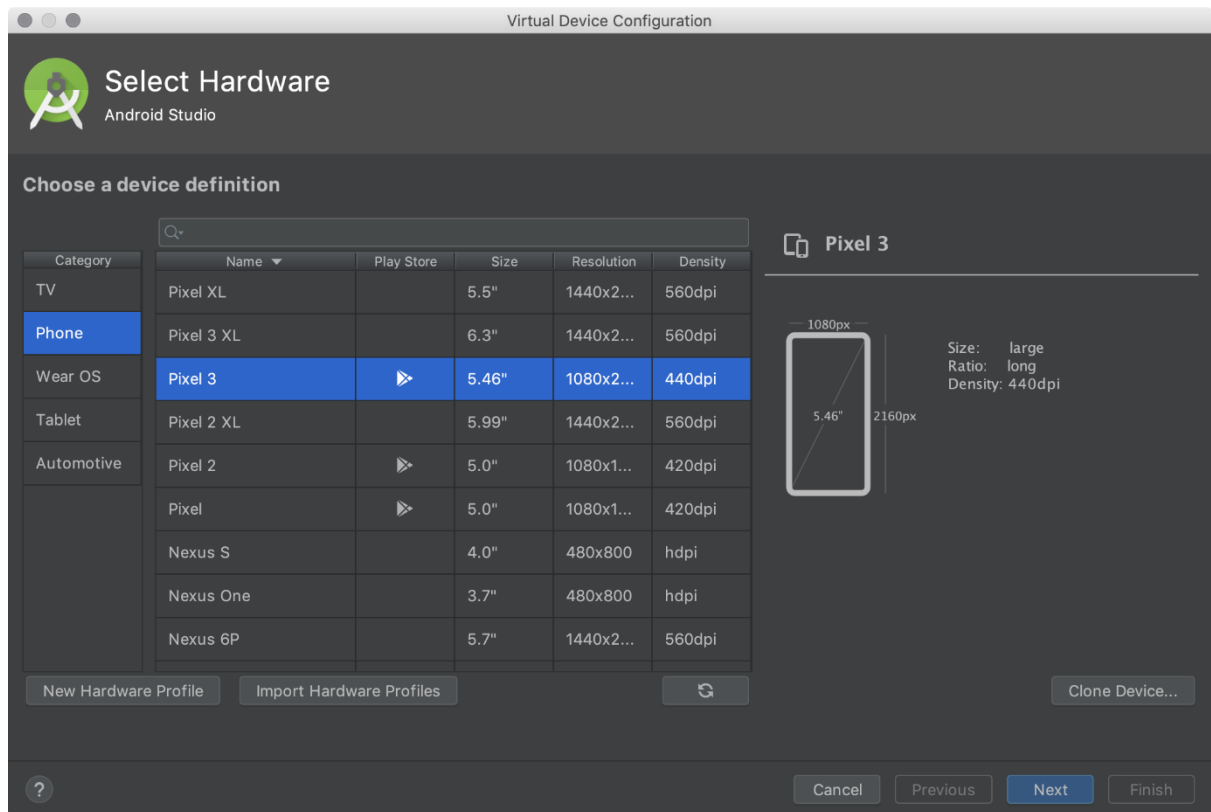
To create a new AVD:

1. Open the AVD Manager by clicking **Tools > AVD Manager**.



2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog.

The **Select Hardware** page appears.

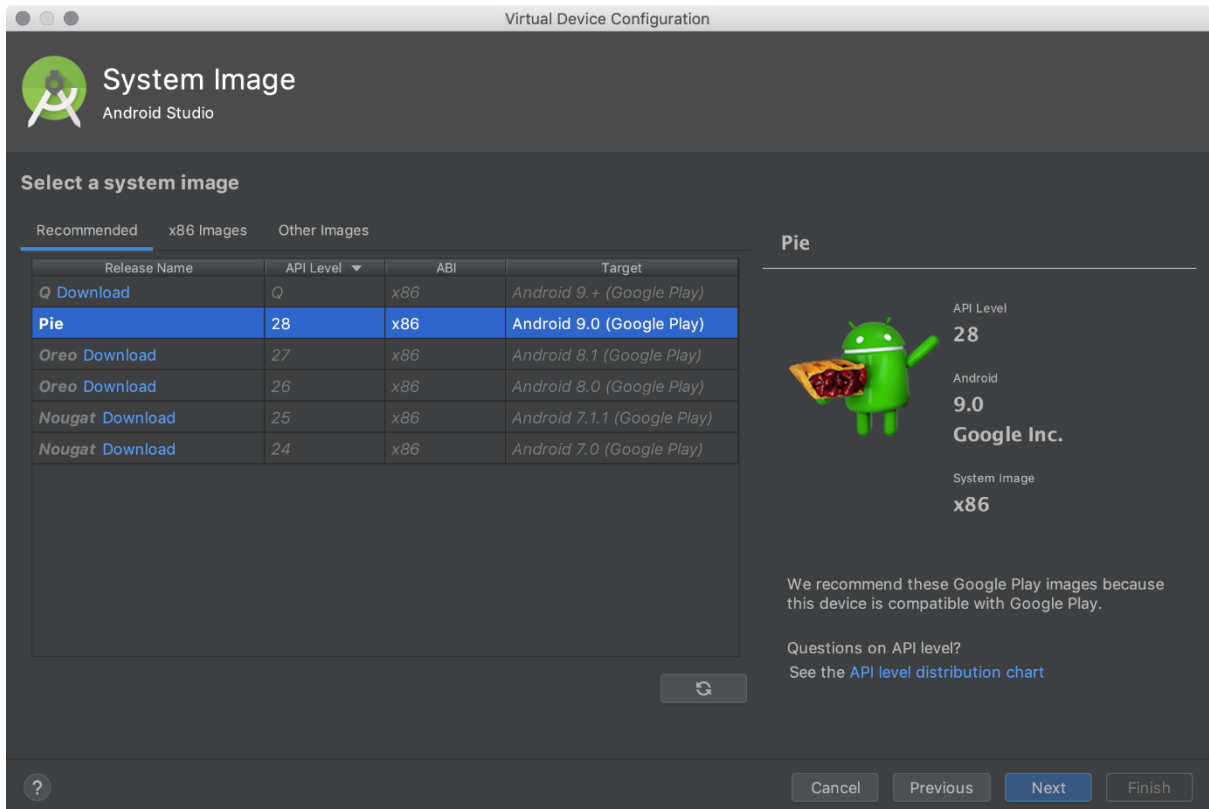


Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.

3. Select a hardware profile, and then click Next.

If you don't see the hardware profile you want, you can create or import a hardware profile.

The **System Image** page appears.



4. Select the system image for a particular API level, and then click Next.

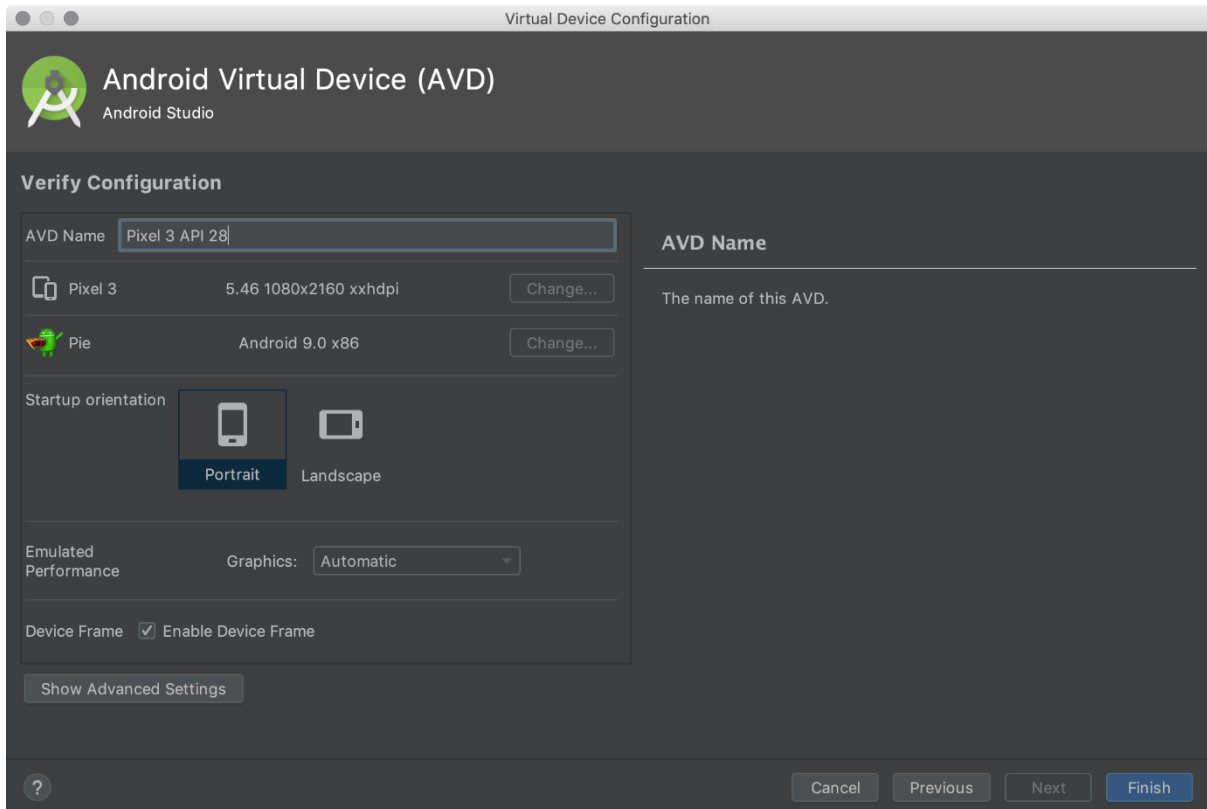
The Recommended tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

If you see Download next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the [minSdkVersion](#) attribute of the app manifest file. For more information about the relationship between system API level and minSdkVersion, see [Versioning Your Apps](#).

If your app declares a [<uses-library>](#) element in the manifest file, the app requires a system image in which that external library is present. If you want to run your app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

The **Verify Configuration** page appears.



5. Change AVD properties as needed, and then click Finish.

Click Show Advanced Settings to show more settings, such as the skin.

The new AVD appears in the **Your Virtual Devices** page or the **Select Deployment Target** dialog.

To create an AVD starting with a copy:

1. From the **Your Virtual Devices** page of the AVD Manager, right-click an AVD and select Duplicate.

Or click Menu ▼ and select Duplicate.

The **Verify Configuration** page appears.

2. Click Change or Previous if you need to make changes on the **System Image** and **Select Hardware** pages.
3. Make your changes, and then click Finish.

The AVD appears in the **Your Virtual Devices** page.

Create a hardware profile:

The AVD Manager provides predefined hardware profiles for common devices so you can easily add them to your AVD definitions. If you need to define a different device, you can create a new hardware profile. You can define a new hardware profile from the beginning, or copy a hardware profile as a start. The preloaded hardware profiles aren't editable.

To create a new hardware profile from the beginning:

1. In the **Select Hardware** page, click New Hardware Profile.

2. In the **Configure Hardware Profile** page, change the hardware profile properties as needed.
3. Click Finish.

Your new hardware profile appears in the **Select Hardware** page. You can optionally create an AVD that uses the hardware profile by clicking Next. Or, click Cancel to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.


To create a hardware profile starting with a copy:

1. In the **Select Hardware** page, select a hardware profile and click Clone Device.
Or right-click a hardware profile and select Clone.
2. In the **Configure Hardware Profile** page, change the hardware profile properties as needed.
3. Click Finish.

Your new hardware profile appears in the **Select Hardware** page. You can optionally create an AVD that uses the hardware profile by clicking Next. Or, click Cancel to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.

Edit existing AVDs:

From the **Your Virtual Devices** page, you can perform the following operations on an existing AVD:

- To edit an AVD, click **Edit this AVD**  and make your changes.
- To delete an AVD, right-click an AVD and select Delete. Or click Menu ▼ and select Delete.
- To show the associated AVD .ini and .img files on disk, right-click an AVD and select Show on Disk. Or click Menu ▼ and select Show on Disk.
- To view AVD configuration details that you can include in any bug reports to the Android Studio team, right-click an AVD and select View Details. Or click Menu ▼ and select View Details.

Edit existing hardware profiles:




From the **Select Hardware** page, you can perform the following operations on an existing hardware profile:

- To edit a hardware profile, select it and click Edit Device. Or right-click a hardware profile and select Edit. Next, make your changes.
- To delete a hardware profile, right-click it and select Delete.

You can't edit or delete the predefined hardware profiles.

Run and stop an emulator, and clear data

From the **Your Virtual Devices** page, you can perform the following operations on an emulator:

- To run an emulator that uses an AVD, double-click the AVD. Or click **Launch** .
- To stop a running emulator, right-click an AVD and select Stop. Or click Menu  and select Stop.
- To clear the data for an emulator, and return it to the same state as when it was first defined, right-click an AVD and select Wipe Data. Or click Menu  and select Wipe Data.

Import and export hardware profiles:

From the **Select Hardware** page, you can import and export hardware profiles:

- To import a hardware profile, click Import Hardware Profiles and select the XML file containing the definition on your computer.
- To export a hardware profile, right-click it and select Export. Specify the location where you want to store the XML file containing the definition.

Hardware profile properties:

You can specify the following properties of hardware profiles in the **Configure Hardware Profile** page. AVD configuration properties override hardware profile properties, and emulator properties that you set while the emulator is running override them both.

The predefined hardware profiles included with the AVD Manager aren't editable. However, you can copy them and edit the copies.

Hardware Profile Property	Description
Device Name	Name of the hardware profile. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (), and spaces. The name of the file storing the hardware profile is derived from the hardware profile name.
Device Type	Select one of the following: <ul style="list-style-type: none">• Phone/Tablet• Wear OS• Android TV• Chrome OS Device• Android Automotive
Screen Size	The physical size of the screen, in inches, measured at the diagonal. If the size is larger than your computer screen, it's reduced in size at launch.
Screen Resolution	Type a width and height in pixels to specify the total number of pixels on the simulated screen.

Round	Select this option if the device has a round screen, such as some Wear OS devices.
Memory: RAM	Type a RAM size for the device and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Input: Has Hardware Buttons (Back/Home/Menu)	Select this option if your device has hardware navigation buttons. Deselect it if these buttons are implemented in software only. If you select this option, the buttons won't appear on the screen. You can use the emulator side panel to "press" the buttons, in either case.
Input: Has Hardware Keyboard	Select this option if your device has a hardware keyboard. Deselect it if it doesn't. If you select this option, a keyboard won't appear on the screen. You can use your computer keyboard to send keystrokes to the emulator, in either case.
Navigation Style	<p>Select one of the following:</p> <ul style="list-style-type: none"> • None - No hardware controls. Navigation is through the software. • D-pad - Directional Pad support. • Trackball • Wheel <p>These options are for actual hardware controls on the device itself. However, the events sent to the device by an external controller are the same.</p>
Supported Device States	<p>Select one or both options:</p> <ul style="list-style-type: none"> • Portrait - Oriented taller than wide. • Landscape - Oriented wider than tall. <p>If you select both, you can switch between orientations in the emulator. You must select at least one option to continue.</p>
Cameras	<p>To enable the camera, select one or both options:</p> <ul style="list-style-type: none"> • Back-Facing Camera - The lens faces away from the user. • Front-Facing Camera - The lens faces toward the user. <p>Later, you can use a webcam or a photo provided by the emulator to simulate taking a photo with the camera.</p>
Sensors: Accelerometer	Select if the device has hardware that helps the device determine its orientation.
Sensors: Gyroscope	Select if the device has hardware that detects rotation or twist. In combination with an accelerometer, it can provide smoother orientation detection and support a six-axis orientation system.
Sensors: GPS	Select if the device has hardware that supports the Global Positioning System (GPS) satellite-based navigation system.
Sensors: Proximity Sensor	Select if the device has hardware that detects if the device is close to your face during a phone call to disable input from the screen.
Default Skin	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too

big for the resolution can mean that the screen is cut off, so you can't see the whole screen. See [Create an emulator skin](#) for more information.

AVD properties:

You can specify the following properties for AVD configurations in the **Verify Configuration** page. The AVD configuration specifies the interaction between the development computer and the emulator, as well as properties you want to override in the hardware profile.

AVD configuration properties override hardware profile properties. Emulator properties that you set while the emulator is running override them both.

AVD Property	Description
AVD Name	Name of the AVD. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (), dashes (-), and spaces. The name of the file storing the AVD configuration is derived from the AVD name.
AVD ID (Advanced)	The AVD filename is derived from the ID, and you can use the ID to refer to the AVD from the command line.
Hardware Profile	Click Change to select a different hardware profile in the Select Hardware page.
System Image	Click Change to select a different system image in the System Image page. An active internet connection is required to download a new image.
Startup Orientation	Select one option for the initial emulator orientation: <ul style="list-style-type: none">• Portrait - Oriented taller than wide.• Landscape - Oriented wider than tall. An option is enabled only if it's selected in the hardware profile. When running the AVD in the emulator, you can change the orientation if both portrait and landscape are supported in the hardware profile.
Camera (Advanced)	To enable a camera, select one or both options: <ul style="list-style-type: none">• Front - The lens faces away from the user.• Back - The lens faces toward the user. The Emulated setting produces a software-generated image, while the Webcam setting uses your development computer webcam to take a picture. This option is available only if it's selected in the hardware profile; it's not available for Wear OS and Android TV.
Network: Speed (Advanced)	Select a network protocol to determine the speed of data transfer: <ul style="list-style-type: none">• GSM - Global System for Mobile Communications• HSCSD - High-Speed Circuit-Switched Data

	<ul style="list-style-type: none"> • GPRS - Generic Packet Radio Service • EDGE - Enhanced Data rates for GSM Evolution • UMTS - Universal Mobile Telecommunications System • HSDPA - High-Speed Downlink Packet Access • LTE - Long-Term Evolution • Full (default) - Transfer data as quickly as your computer allows.
Network: Latency (Advanced)	Select a network protocol to set how much time (delay) it takes for the protocol to transfer a data packet from one point to another point.
Emulated Performance: Graphics	<p>Select how graphics are rendered in the emulator:</p> <ul style="list-style-type: none"> • Hardware - Use your computer graphics card for faster rendering. • Software - Emulate the graphics in software, which is useful if you're having a problem with rendering in your graphics card. • Automatic - Let the emulator decide the best option based on your graphics card.
Emulated Performance: Boot option (Advanced)	<ul style="list-style-type: none"> • Cold boot - Start the device each time by powering up from the device-off state. • Quick boot - Start the device by loading the device state from a saved snapshot. For details, see Run the emulator with Quick Boot.
Emulated Performance: Multi-Core CPU (Advanced)	Select the number of processor cores on your computer that you'd like to use for the emulator. Using more processor cores speeds up the emulator.
Memory and Storage: RAM	The amount of RAM on the device. This value is set by the hardware manufacturer, but you can override it, if needed, such as for faster emulator operation. Increasing the size uses more resources on your computer. Type a RAM size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: VM Heap	The VM heap size. This value is set by the hardware manufacturer, but you can override it, if needed. Type a heap size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). For more information on Android VMs, see Memory Management for Different Virtual Machines .
Memory and Storage: Internal Storage	The amount of nonremovable memory space available on the device. This value is set by the hardware manufacturer, but you can override it, if needed. Type a size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: SD Card	The amount of removable memory space available to store data on the device. To use a virtual SD card managed by Android Studio, select Studio-managed , type a size, and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). A minimum of 100 MB is recommended to use the camera. To manage the space in a file, select External file and click ... to specify the file and location. For more information, see mkSDcard and AVD data directory .

Device Frame: Enable Device Frame	Select to enable a frame around the emulator window that mimics the look of a real device.
Custom Skin Definition (Advanced)	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the skin can mean that the screen is cut off, so you can't see the whole screen. See Create an emulator skin for more information.
Keyboard: Enable Keyboard Input (Advanced)	Select this option if you want to use your hardware keyboard to interact with the emulator. It's disabled for Wear OS and Android TV.

Create an emulator skin:

An Android emulator skin is a collection of files that define the visual and control elements of an emulator display. If the skin definitions available in the AVD settings don't meet your requirements, you can create your own custom skin definition, and then apply it to your AVD.

Each emulator skin contains:

- A hardware.ini file
- Layout files for supported orientations (landscape, portrait) and physical configuration
- Image files for display elements, such as background, keys and buttons

To create and use a custom skin:

1. Create a new directory where you will save your skin configuration files.
2. Define the visual appearance of the skin in a text file named layout. This file defines many characteristics of the skin, such as the size and image assets for specific buttons. For example:
3. parts {
4. device {
5. display {
6. width 320
7. height 480
8. x 0
9. y 0
10. }
11. }
- 12.
13. portrait {

```
14.     background {
15.         image background_port.png
16.     }
17.
18.     buttons {
19.         power {
20.             image button_vertical.png
21.             x 1229
22.             y 616
23.         }
24.     }
25. }
26. ...
27. }
```

28. Add the bitmap files of the device images in the same directory.

29. Specify additional hardware-specific device configurations in a hardware.ini file for the device settings, such as hw.keyboard and hw.lcd.density.

30. Archive the files in the skin folder and select the archive file as a custom skin.

For more detailed information about creating emulator skins, see the [Android Emulator Skin File Specification](#) in the tools source code.

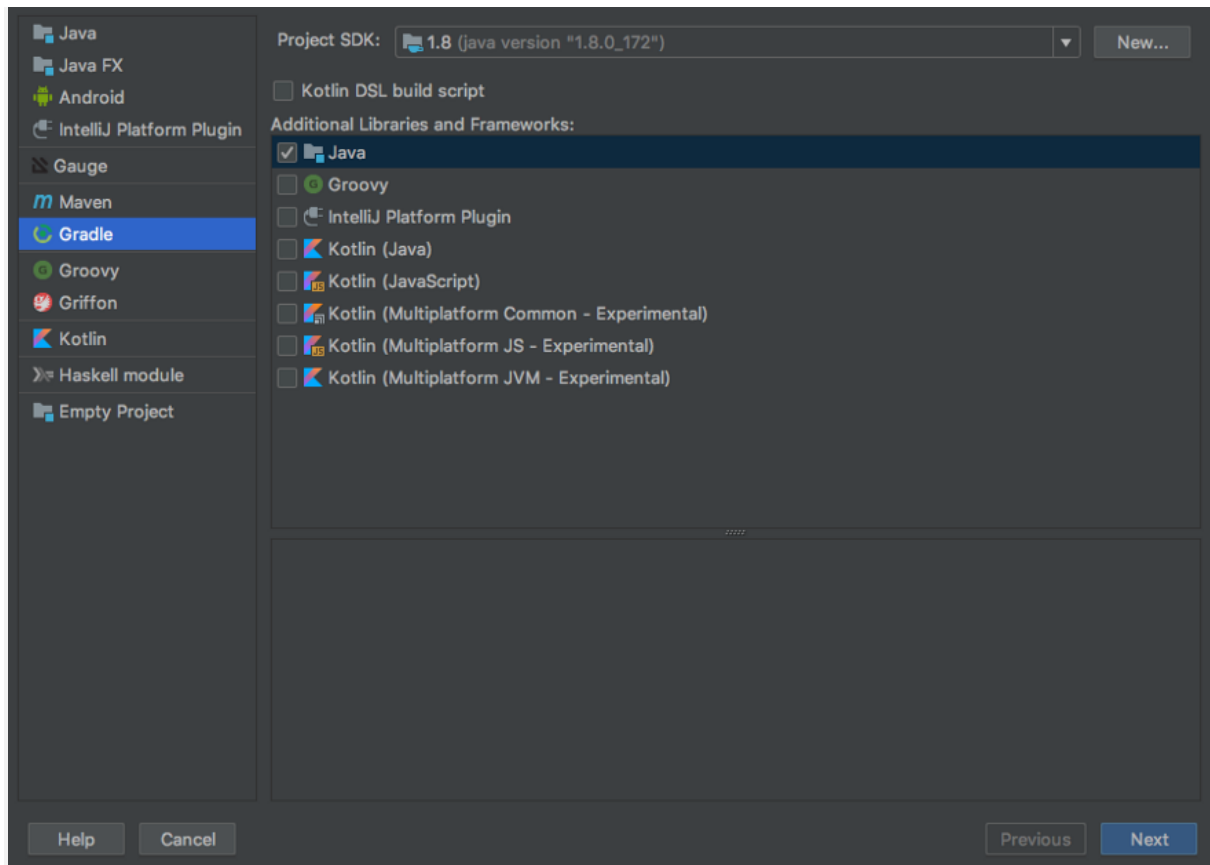
A beginners guide to Gradle:

What is Gradle?

Gradle is a build automation tool often used for JVM languages such as Java, Groovy or Scala. Gradle can be configured to run `Tasks` which do things like compile jars, run tests, create documentation and much more.

Starting a new project with Gradle using IntelliJ:

When creating a new project with intelij you can select the `gradle` option from the left and choose which language you'd like to use with it



After creating your project, IntelliJ will create a `build.gradle` file with some helpful defaults for that language, so let's have a look at the `build.gradle` file. I'll be using Java in the examples but the same concepts apply for other languages.

The `build.gradle` file is where all the magic happens. This is the default java one created by IntelliJ:

```
1  plugins {
2      id 'java'
3  }
4
5      version '1.0-SNAPSHOT'
6
7      sourceCompatibility = 1.8
8
9  repositories {
10     mavenCentral()
11 }
12
13 dependencies {
14     testCompile group: 'junit', name: 'junit', version: '4.12'
15 }
16
```

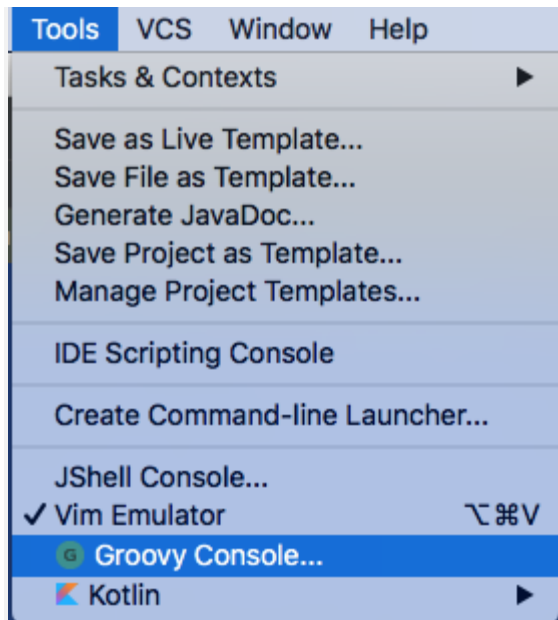
But what does it all mean? And where does this syntax come from?

Groovy 101:

Gradle build scripts are written in **Groovy**, a JVM language similar to Java but with a more concise syntax — let's learn just enough to have a better understanding of what's going on:

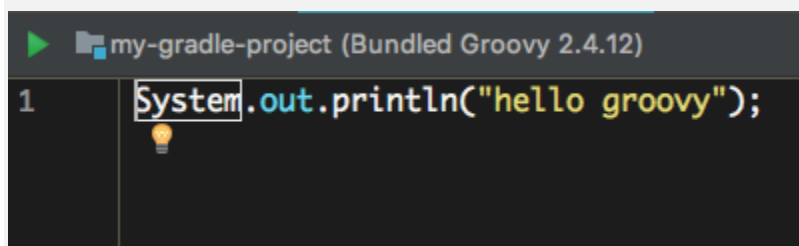
The Groovy console

You can try out groovy scripts using the groovy console in **IntelliJ** under **tools** -
> **groovy console**

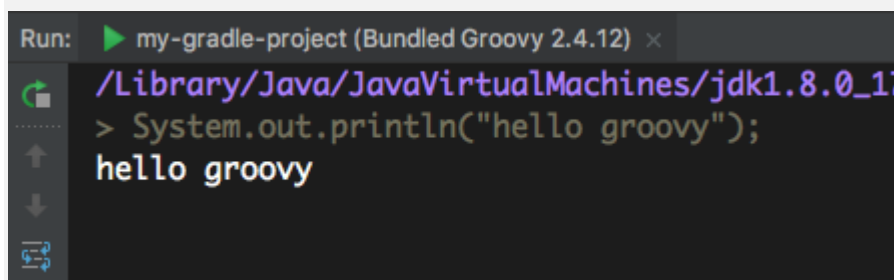


Valid Java is valid Groovy

In the console area we can type in any valid Java code:



Pressing play (the green button) we'll see the output in a window underneath



One thing to note already is that we don't need a surrounding `class` and `main` method to execute our code 🐣. But we can go further. Groovy automatically imports `System.out` so we can omit that:

```
my-gradle-project (Bundled Groovy 2.4.12)
1 println("hello groovy");
2 |
```

Parens for single argument method calls and semicolons are also optional in Groovy so we can reduce `System.out.println("hello groovy");` to:

```
my-gradle-project (Bundled Groovy 2.4.12)
1 println "hello groovy"
2 |
```

snazzy ✨

Groovy Closures:

Understanding Groovy Closures will demystify a lot of what's going on in `build.gradle`

If you've used Java 8's lambdas, groovy closures will feel familiar — they effectively let you treat methods as regular values (so they can be used as arguments to other methods). Let's have a look at an example:

In the groovy console we can define a class like this:

```
my-gradle-project (Bundled G
1 class MyClass {
2
3 }
```

Let's add a method that takes a closure:

```
my-gradle-project (Bundled Groovy 2.4.12)
1  class MyClass {
2
3      void doSomething(Closure closure) {
4          closure.call()
5      }
6
7  }
```

We can “run” whatever code is in the closure with `closure.call()`, but how would we use this? We’ll create an instance of `MyClass` and call `doSomething` like so:

```
my-gradle-project (Bundled Groovy 2.4.12)
1  class MyClass {
2
3      void doSomething(Closure closure) {
4          closure.call()
5      }
6
7  }
8
9  myClass = new MyClass()
10
11 myClass.doSomething {
12     println "doing something"
13 }
14
```

Which outputs:

```
> myClass = new MyClass()
>
> myClass.doSomething {
>     println "doing something"
> }
doing something
```

Does this look familiar now?


```
dependencies {  
    testCompile group: 'junit', name: 'junit', version: '4.12'  
}
```

`dependencies` is a method which takes a “runnable” block of code (a closure). Inside that block we’ve called the `testCompile` method with `group: 'junit'` etc as an argument (the `group`, `name`, `version` section is actually short hand for a groovy map, essentially a list of key value pairs).

The Core of Gradle: Projects and Tasks:

Now we have a better understanding of how groovy syntax works let’s take a deeper look at the `build.gradle` file.

The `build.gradle` file has a one to one relationship with something called the project object: It’s an object representing information about our project. If you’ve used Maven before (another popular build tool Gradle aims to be compatible with) you’ll recognise this idea from the `pom.xml` file (project-object-model), this is roughly equivalent to `build.gradle`.

Each project is made up of a collection of `Tasks`: these are atomic units of work that represent the things that need to be done to build our project.

Gradle has a number of default Tasks on the project object model — we can have a look at these by running the task:
> gradle tasks

In a blank `build.gradle` file (i.e. if you deleted everything from the one IntelliJ made) would output this:

```

> Task :tasks

-----
All tasks runnable from root project
-----

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Help tasks
-----
buildEnvironment - Displays all buildscript dependencies declared in root project 'my-gradle-project'.
components - Displays the components produced by root project 'my-gradle-project'. [incubating]
dependencies - Displays all dependencies declared in root project 'my-gradle-project'.
dependencyInsight - Displays the insight into a specific dependency in root project 'my-gradle-project'.
dependentComponents - Displays the dependent components of components in root project 'my-gradle-project'. [incubating]
help - Displays a help message.
model - Displays the configuration model of root project 'my-gradle-project'. [incubating]
projects - Displays the sub-projects of root project 'my-gradle-project'.
properties - Displays the properties of root project 'my-gradle-project'.
tasks - Displays the tasks runnable from root project 'my-gradle-project'.

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

```

The `tasks` task shows all the Tasks available on a project. The real magic starts to happen at this line:

```

1  plugins {
2      id 'java'
3  }

```

Applying the Java plugins adds a bunch of extra tasks we can use. Running `gradle tasks` now includes these lines:

```
Build tasks
-----
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles test classes.

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Documentation tasks
-----
javadoc - Generates Javadoc API documentation for the main source code.
```

These are some helpful default tasks that many Java projects are likely to use.

The key takeaway here is that we can **modify** the project object model via the `build.gradle` script — Gradle is **code as configuration**.

Check out the full docs for the Java plugin here.

Properties:

As well as Tasks, our project object also has “properties” that represent information about our project. These can be simple properties such as the `version` or the `sourceCompatibility`, you can see these being modified here:

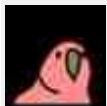
```
5    version '1.0-SNAPSHOT'
6
7    sourceCompatibility = 1.8
```

Or they can be more complex properties that are modified by methods that take `Closures`. A piece of magic that confused me when I started using Gradle was

the `sourceSets` property — this gets added by the `Java` plugin. The `sourceSets` property defines where Gradle should look for your source code (plus a couple of other snazzy things you can check out [here](#)). By default Gradle follows the Maven convention of putting your Java source code in `src/main/java` and tests in `src/test/java`.

But say we want to locate our java source code in `my-java-directory` (I wouldn't recommend this in a real project), or maybe we're working on a legacy project; we can point Gradle at a different directory like this:

```
17 sourceSets {
18     main {
19         java {
20             srcDir "my-java-directory"
21         }
22     }
23 }
```



Just like `Tasks` we can see a full list of properties for our project by running the task (be warned, it can be pretty large).

```
> gradle properties
```

Customising Tasks

We can do the same customising magic with `Tasks` too. When building my Tic Tac Toe command line application I was looking for some way of running it conveniently from Gradle — the `application` plugin had a handy task called `run`.

However, it needed to be configured to point `System.in` as its `standardInput`. Adding this line let me reconfigure the `run` task:

```
18  run {  
19      standardInput = System.in  
20  }
```

Another thing this highlights is that Tasks can also have their own properties which we're able to configure.

Making a new Task

We can also create our own custom tasks, either from scratch or more often from extending existing Tasks. The `run` task from above actually extends a lower level Task type called `JavaExec`. We could have defined the same Task as above like this:

```
22  task customRun(type: JavaExec) {  
23      classpath = sourceSets.main.runtimeClasspath  
24      standardInput = System.in  
25      main = 'tictactoe.cli.Main'  
26  }
```

This adds a `customRun` task to the project object which we can use:
> gradle customRun

And would have the same effect as the `run` task from before.

Conclusion:

The Gradle api is HUGE and quite overwhelming at first. This post only scratches the surface but hopefully it demystifies some of the syntax and concepts in Gradle to make understanding the docs a little easier.

There are some excellent guides on the Gradle website for more specific cases. This online course from [Lynda.com \(Gradle for Java Developers\)](https://www.lynda.com/Gradle/Gradle-for-Java-Developers) was particularly helpful for me for getting a better idea of what was going on.

Connect your App to Firebase:

If you haven't already, [add Firebase to your Android project](#).

Create a Database:

1. Navigate to the **Realtime Database** section of the [Firebase console](#). You'll be prompted to select an existing Firebase project. Follow the database creation workflow.
2. Select a starting mode for your Firebase Security Rules:

Test mode

Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data. After testing, **make sure to review the [Understand Firebase Realtime Database Rules](#) section.**

Note: If you create a database in Test mode and make no changes to the default world-readable and world-writeable Rules within a trial period, you will be alerted by email, then your database rules will deny all requests. Note the expiration date during the Firebase console setup flow.

To get started with the web, Apple, or Android SDK, select testmode.

Locked mode

Denies all reads and writes from mobile and web clients. Your authenticated application servers can still access your database.

3. Choose a region for the database. Depending on your choice of region, the database namespace will be of the form <databaseName>.firebaseio.com or <databaseName>.<region>.firebasedatabase.app. For more information, see [select locations for your project](#).
4. Click **Done**.

When you enable Realtime Database, it also enables the API in the [Cloud API Manager](#).

Add the Realtime Database SDK to your app:

Using the [Firebase Android BoM](#), declare the dependency for the Realtime Database Android library in your **module (app-level) Gradle file** (usually app/build.gradle).
[JavaKotlin+KTX](#)

```
dependencies {  
    // Import the BoM for the Firebase platform  
    implementation platform('com.google.firebase:firebase-bom:29.0.4')  
  
    // Declare the dependency for the Realtime Database library  
    // When using the BoM, you don't specify versions in Firebase library dependencies  
    implementation 'com.google.firebase:firebase-database'  
}
```

By using the [Firebase Android BoM](#), your app will always use compatible versions of the Firebase Android libraries.

(Alternative) Declare Firebase library dependencies *without* using the BoM

Configure Real-time Database Rules:

The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to.

Note: By default, read and write access to your database is restricted so only authenticated users can read or write data. To get started without setting up [Authentication](#), you can [configure your rules for public access](#). This does make your database open to anyone, even people not using your app, so be sure to restrict your database again when you set up authentication.

Write to your database:

Retrieve an instance of your database using `getInstance()` and reference the location you want to write to.

Note: To get a reference to a database other than a **us-central1** default database, you must pass the database URL to `getInstance()` (or Kotlin+KTX `database()`). For a **us-central1** default database, you can call `getInstance()` (or `database`) without arguments.

You can find your Realtime Database URL in the **Realtime Database** section of the [Firebase console](#). It will have the form `https://<databaseName>.firebaseio.com` (for **us-central1** databases) or `https://<databaseName>.<region>.firebase.database.app` (for databases in all other locations).

[JavaKotlin+KTX](#)

```
// Write a message to the database
```

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
```

```
DatabaseReference myRef = database.getReference("message");
```

```
myRef.setValue("Hello, World!");
```

[MainActivity.java](#)

You can save a range of data types to the database this way, including Java objects. When you save an object the responses from any getters will be saved as children of this location.

Read from your database:

To make your app data update in realtime, you should add a [ValueEventListener](#) to the reference you just created.

The `onDataChange()` method in this class is triggered once when the listener is attached and again every time the data changes, including the children.

[JavaKotlin+KTX](#)

```
// Read from the database
```

```
myRef.addValueEventListener(new ValueEventListener() {
```

```
    @Override
```

```
    public void onDataChange(DataSnapshot dataSnapshot) {
```

```
        // This method is called once with the initial value and again
```

```
        // whenever data at this location is updated.
```

```
        String value = dataSnapshot.getValue(String.class);
```

```

        Log.d(TAG, "Value is: " + value);
    }

    @Override
    public void onCancelled(DatabaseError error) {
        // Failed to read value
        Log.w(TAG, "Failed to read value.", error.toException());
    }
});
MainActivity.java

```

Optional: Configure Pro-Guard:

When using Firebase Realtime Database in your app along with ProGuard, you need to consider how your model objects will be serialized and deserialized after obfuscation. If you use `DataSnapshot.getValue(Class)` or `DatabaseReference.setValue(Object)` to read and write data, you will need to add rules to the `proguard-rules.pro` file:

```

# Add this global rule
-keepattributes Signature

# This rule will properly ProGuard all the model classes in
# the package com.yourcompany.models.
# Modify this rule to fit the structure of your app.
-keepclassmembers class com.yourcompany.models.** {
    *;
}

```

To get help for questions or issues related to ProGuard, visit the [Guardsquare Community forums](#) to get assistance from an expert.

Prepare for Launch:

Before launching your app, we recommend walking through our [launch checklist](#) to make sure your app is ready to go!

Be sure to enable [App Check](#) to help ensure that only your apps can access your databases

Expected Outcome:

The main objective of this milestone is to make sure that you have the required development environment in place. On completion of the above requirements, run the application using the virtual device and the end result should be as shown in the picture below.



Task 2:

Setting up Firebase and Authentication:

Firebase is a great service provided by Google for configuring the backend of any application with all the general necessities like database preparation, authentication using various methods, etc. In this milestone, we'll be preparing our database and setting up authentication using email and password. [Note: Use the references provided to implement the following requirements.]

Requirements :

- Setup sign-in method using Email/Password.
- Declare the dependency for the Firebase Authentication Android library in your module (app-level) Gradle file (usually app/build.gradle).
- To use an authentication provider, you need to enable it in the Firebase console. Go to the Sign-in Method page in the Firebase Authentication section to enable Email/Password sign-in and any other identity providers you want for your app.
- Android has Material Components, which helps in building our frontend.
- Create a new activity named Login. Style the activity so that it looks similar to the one shown below.

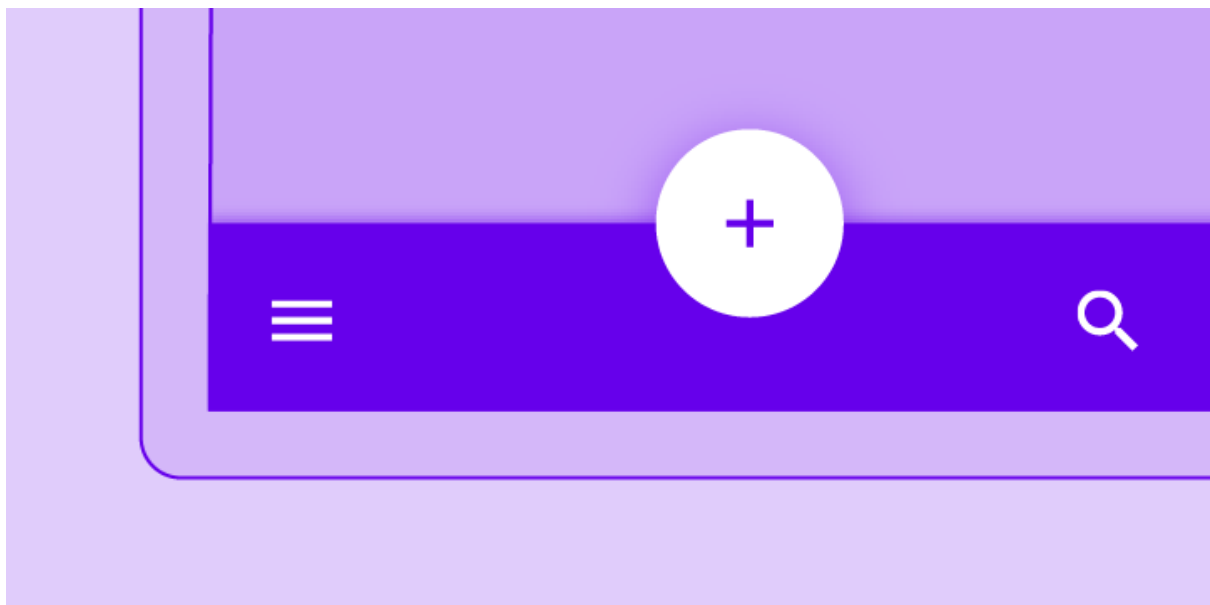


Components:

Material Components are interactive building blocks for creating a user interface.

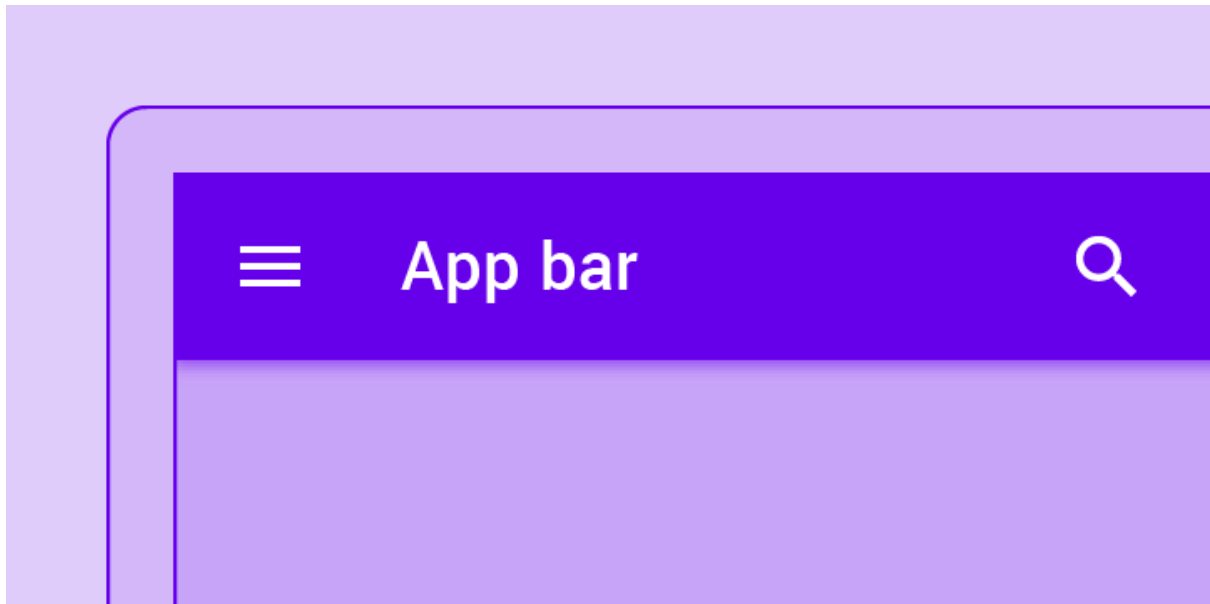
Browse all components or select a specific platform.

All Components Android Web Flutter iOS:



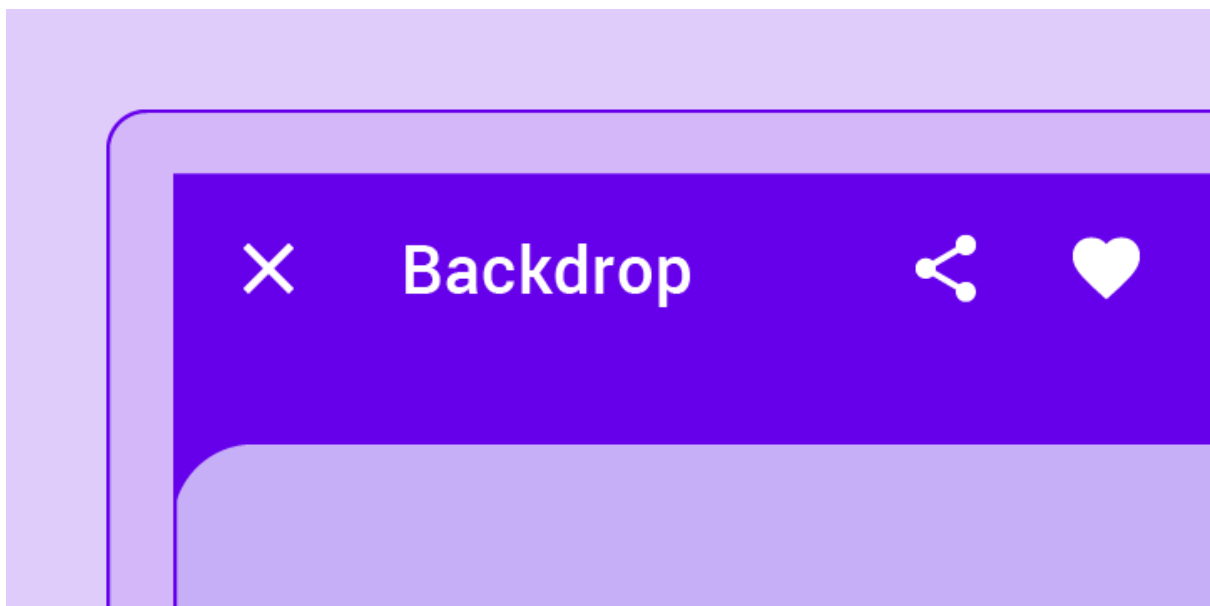
App bars: Bottom

A bottom app bar displays navigation and key actions at the bottom of mobile screens



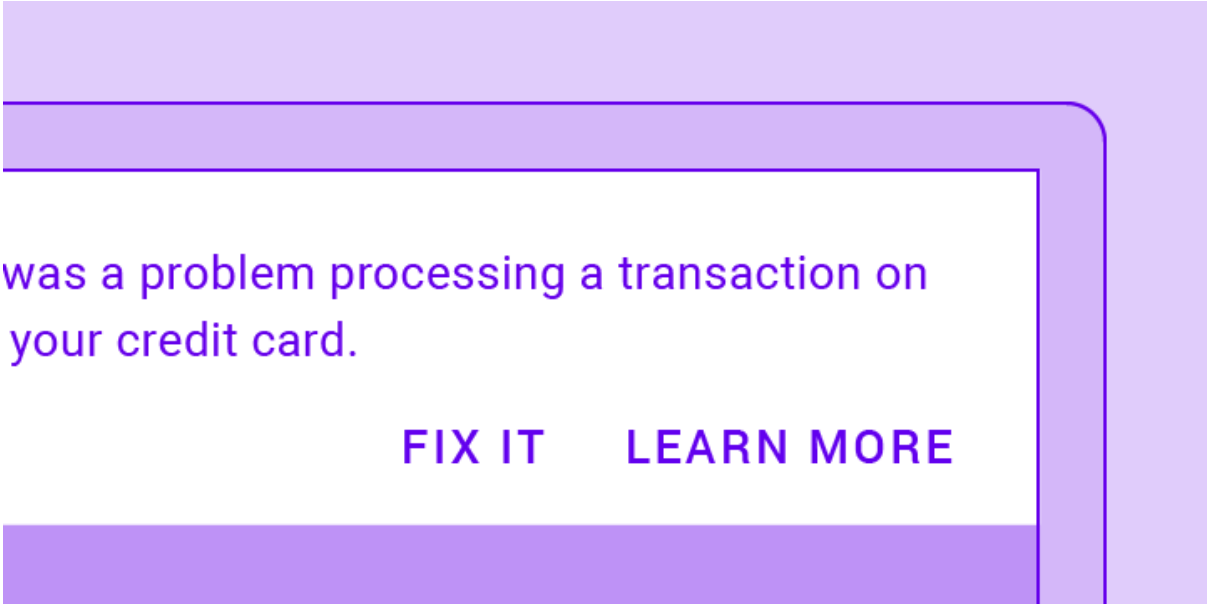
App bars: top

The top app bar displays information and actions relating to the current screen



Backdrop

A backdrop appears behind all other surfaces in an app, displaying contextual and actionable content

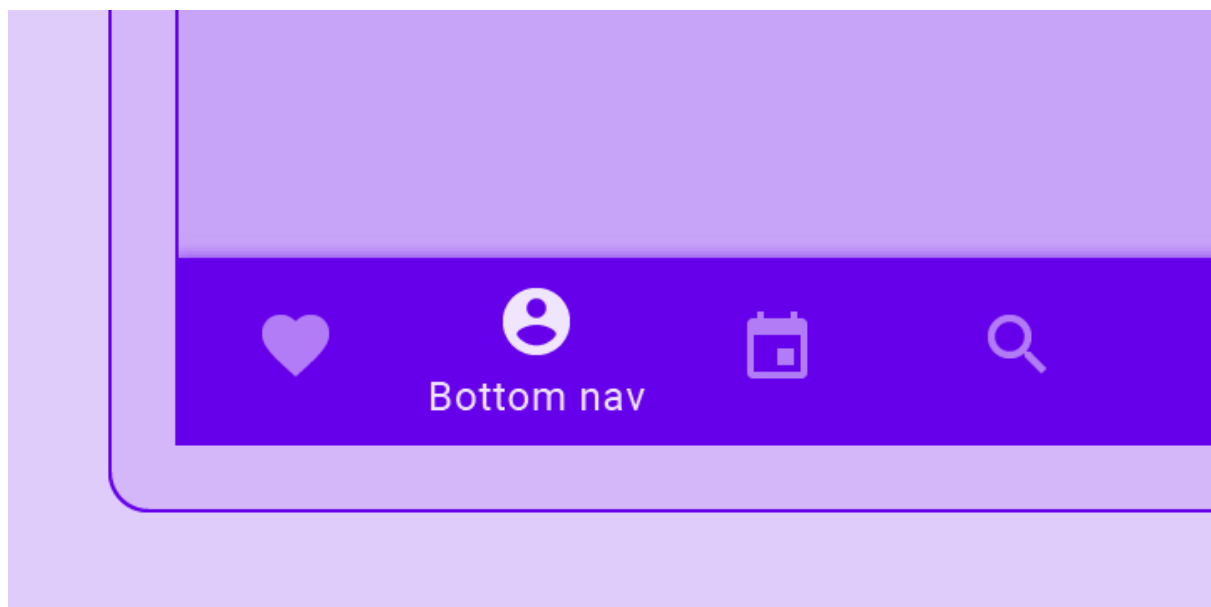
A banner with a light purple background and a white rounded rectangle in the center. The text 'was a problem processing a transaction on your credit card.' is in the white area, and two buttons 'FIX IT' and 'LEARN MORE' are at the bottom of the white area.

was a problem processing a transaction on
your credit card.

FIX IT **LEARN MORE**

Banners:

A banner displays a prominent message and related optional actions



Bottom navigation:

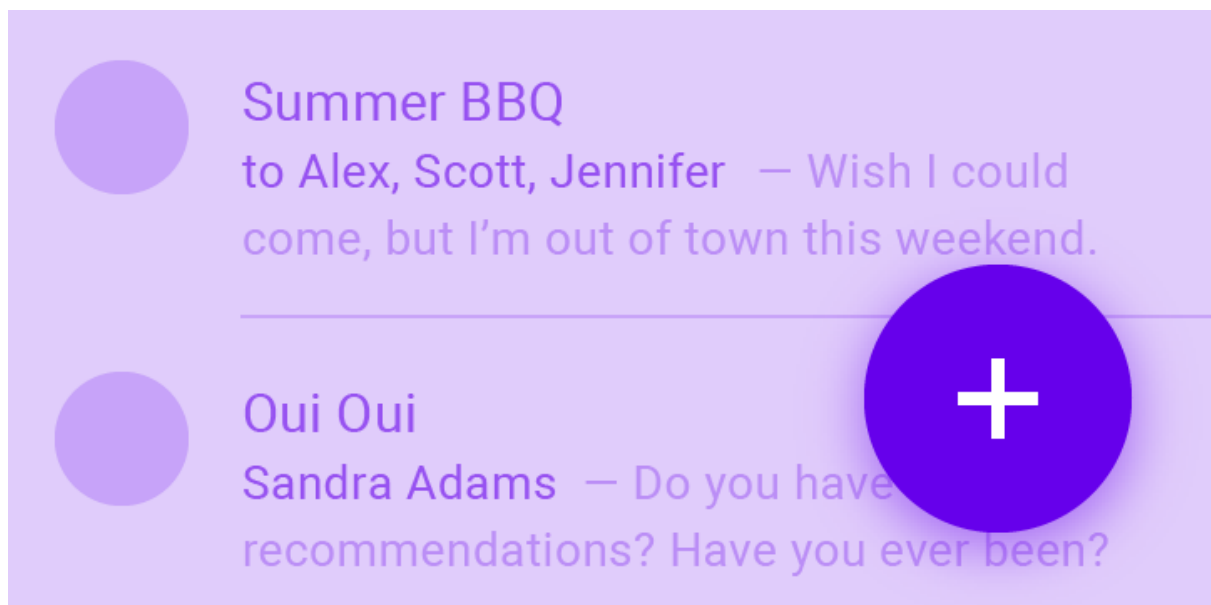
Bottom navigation bars allow movement between primary destinations in an app

The great horned owl is a large owl native to the Americas. It is an extremely adaptable bird with a vast range and is the most widely distributed true owl in the Americas.

BUTTON

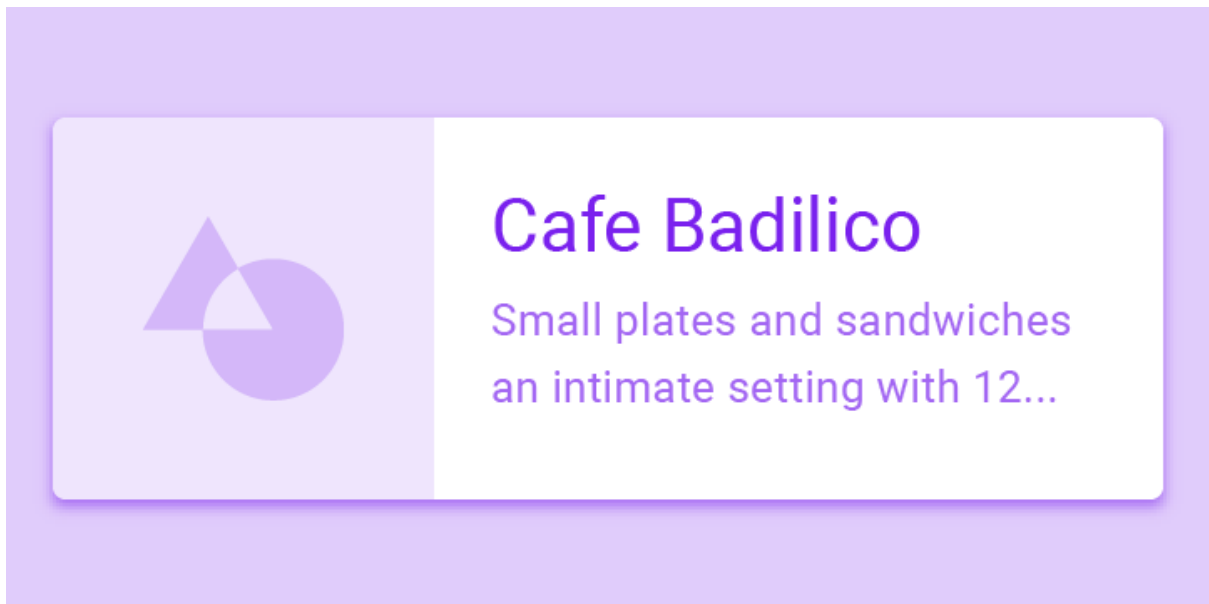
Buttons:

Buttons allow users to take actions, and make choices, with a single tap



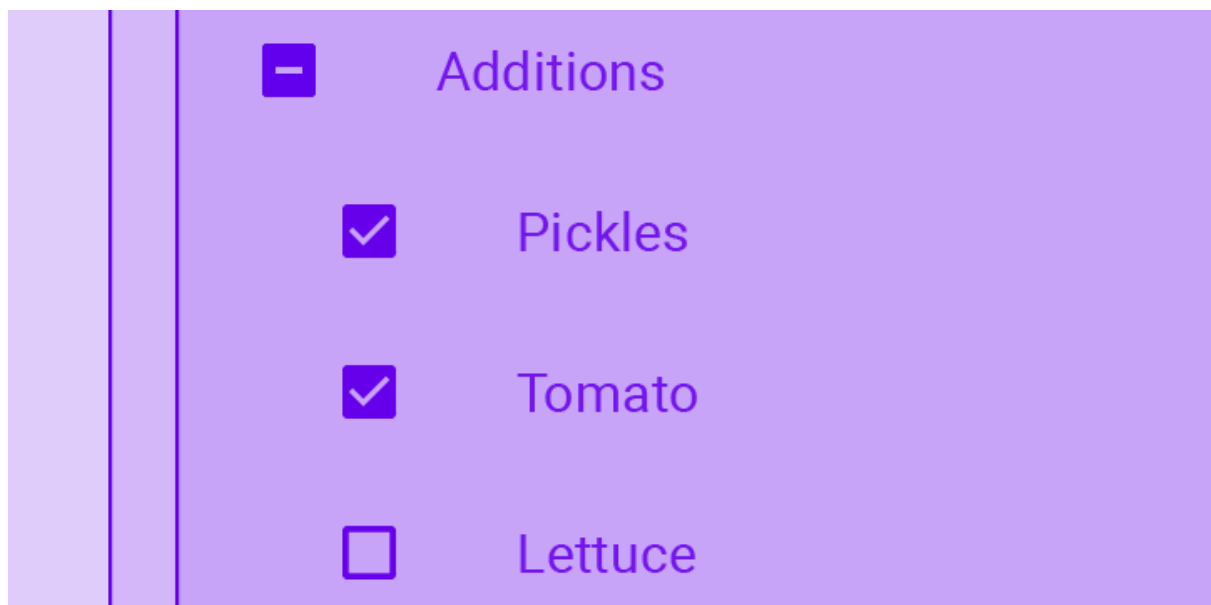
Buttons: floating action button:

A floating action button (FAB) represents the primary action of a screen



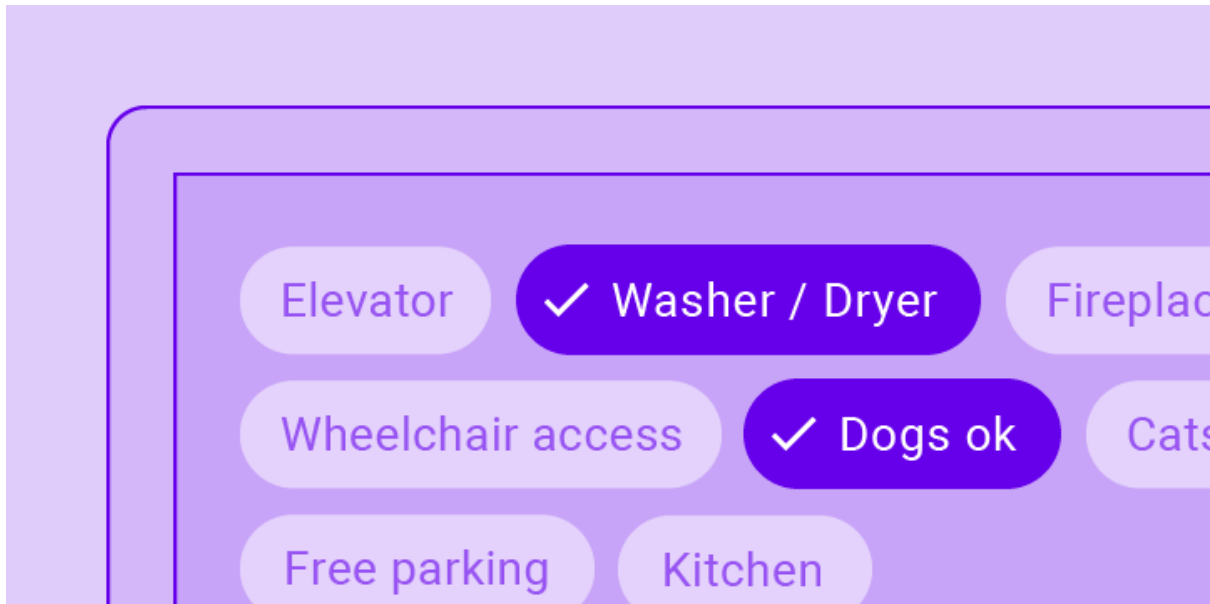
Cards:

Cards contain content and actions about a single subject



Checkboxes:

Checkboxes allow the user to select one or more items from a set or turn an option on or off



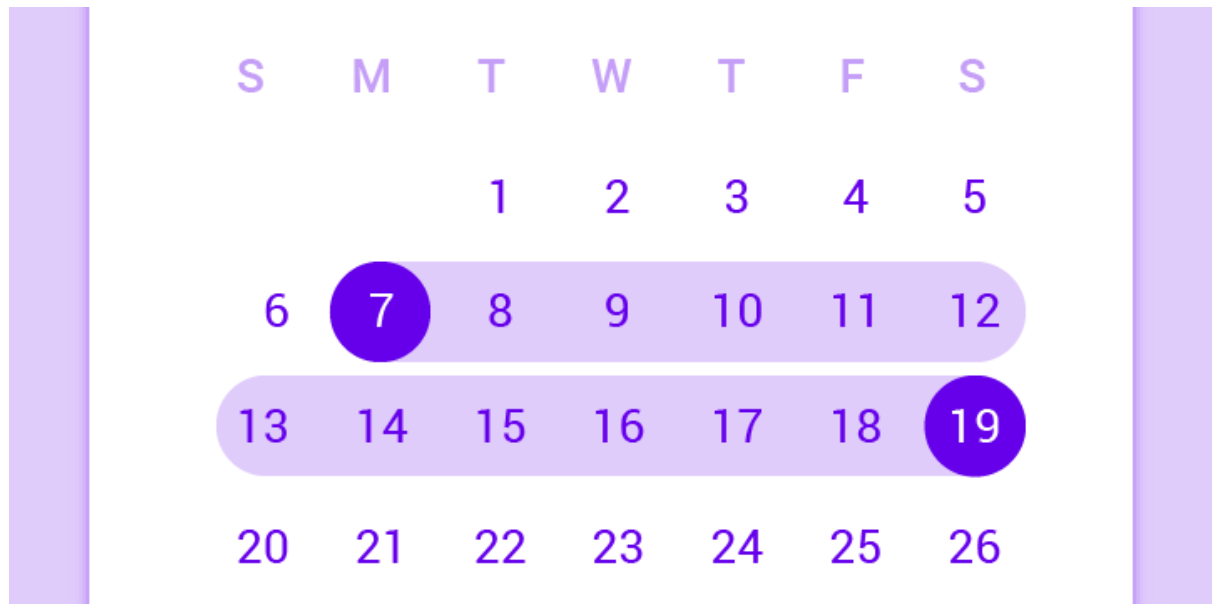
Chips:

Chips are compact elements that represent an input, attribute, or action

<input type="checkbox"/>	Online	Astrid: NE shared main
<input checked="" type="checkbox"/>	Offline	Cosmo: prod shared a
<input checked="" type="checkbox"/>	Online	Phoenix: prod shared l
<input type="checkbox"/>	Online	Sirius: prod shared an

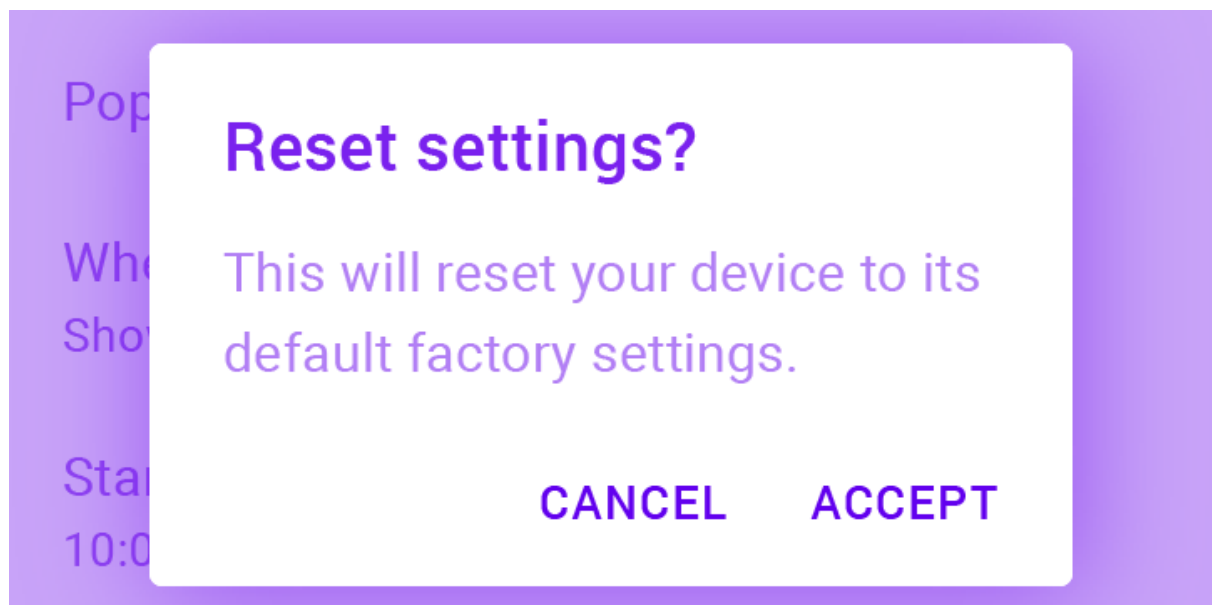
Data tables:

Data tables display sets of data



Date pickers:

Date pickers let users select a date, or a range of dates



Dialogs:

Dialogs inform users about a task and can contain critical information, require decisions, or involve multiple tasks



Dividers:

A divider is a thin line that groups content in lists and layouts

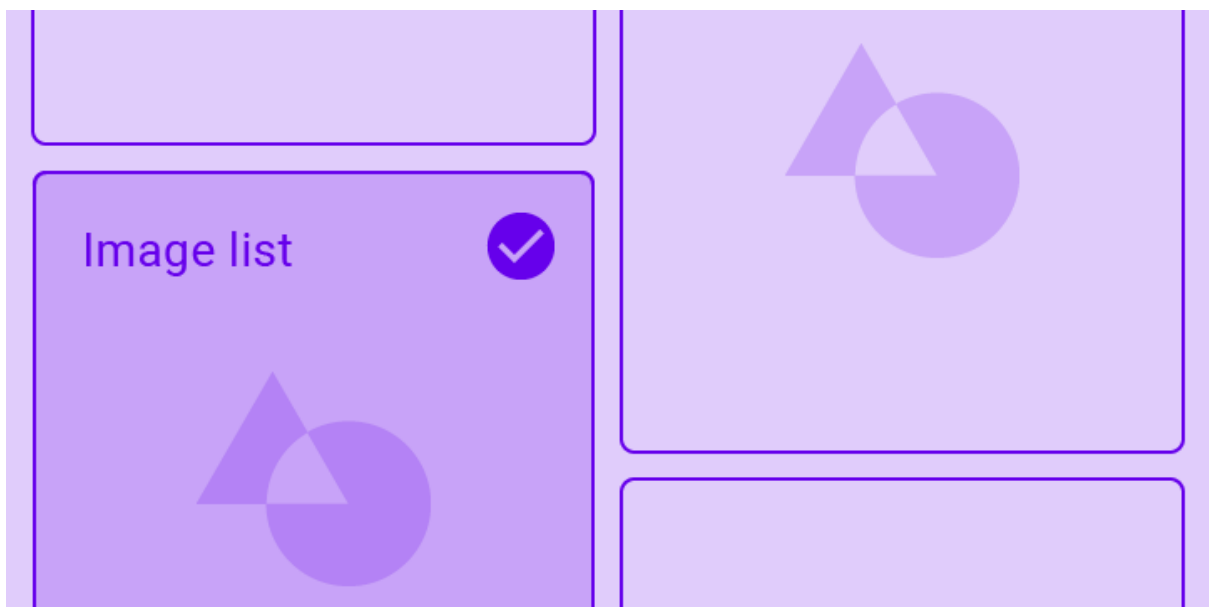
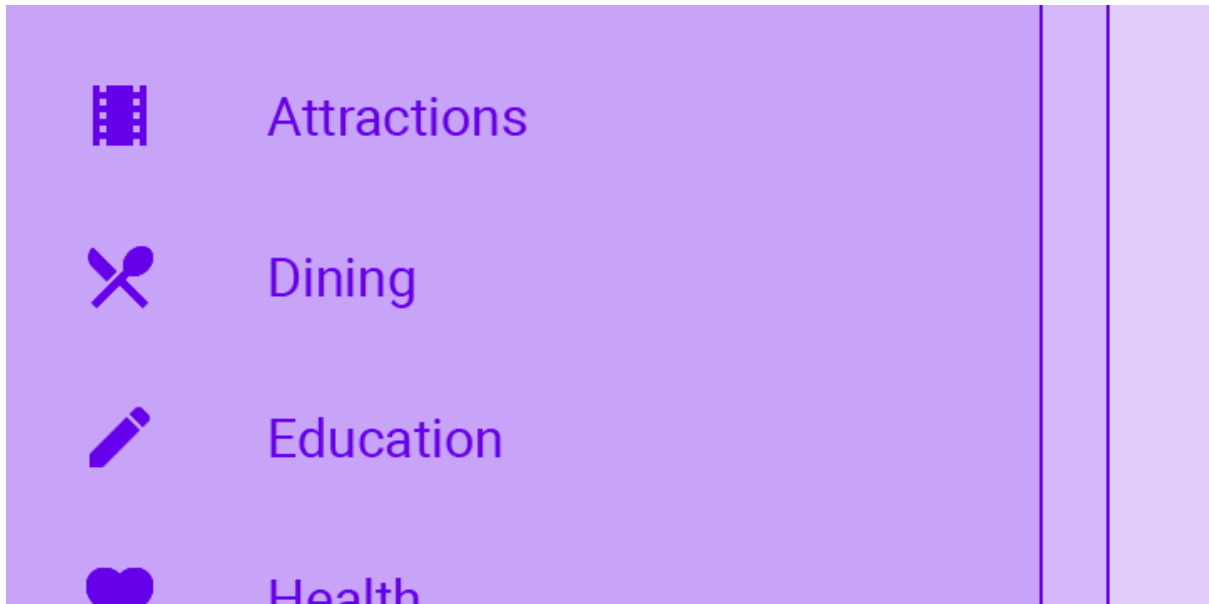


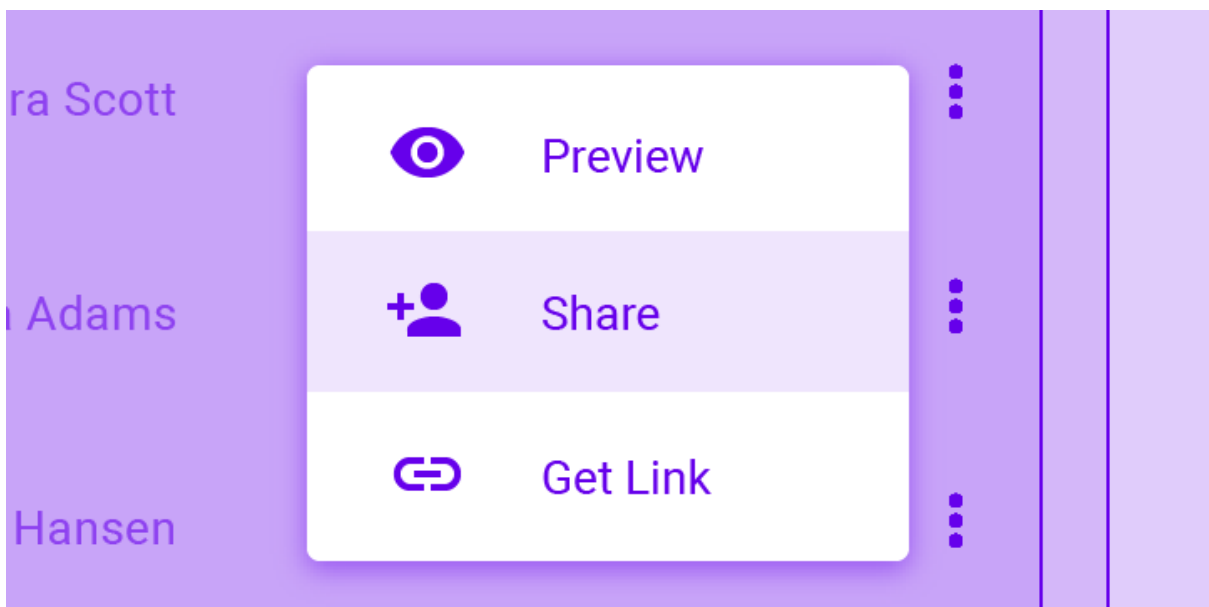
Image lists:

Image lists display a collection of images in an organized grid



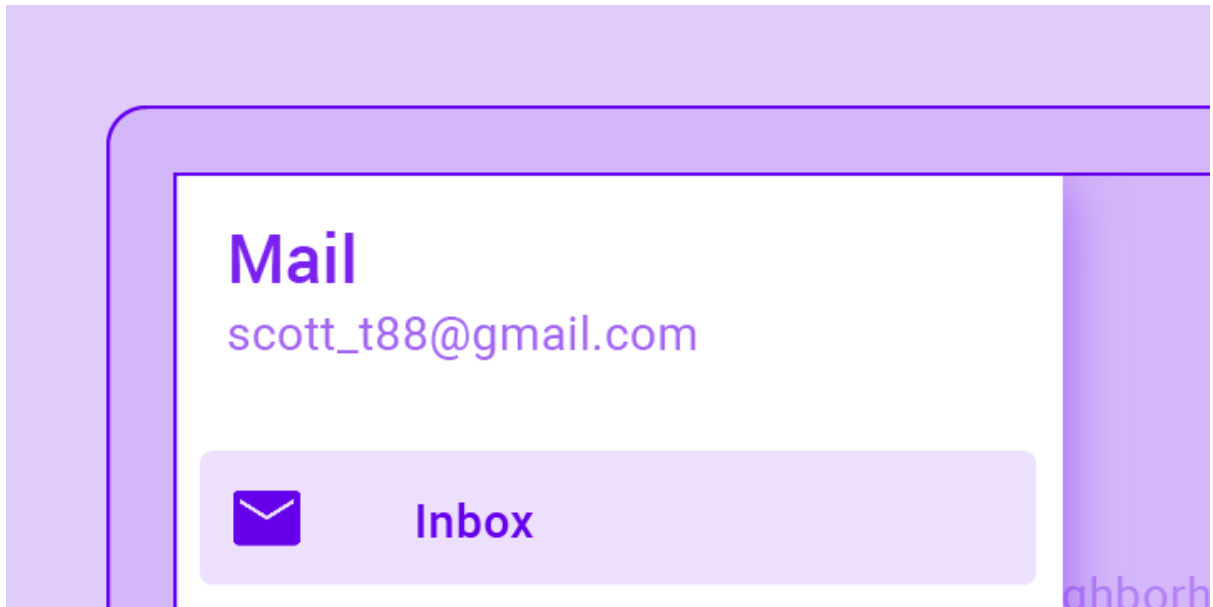
Lists:

Lists are continuous, vertical indexes of text or images



Menus:

Menus display a list of choices on temporary surfaces



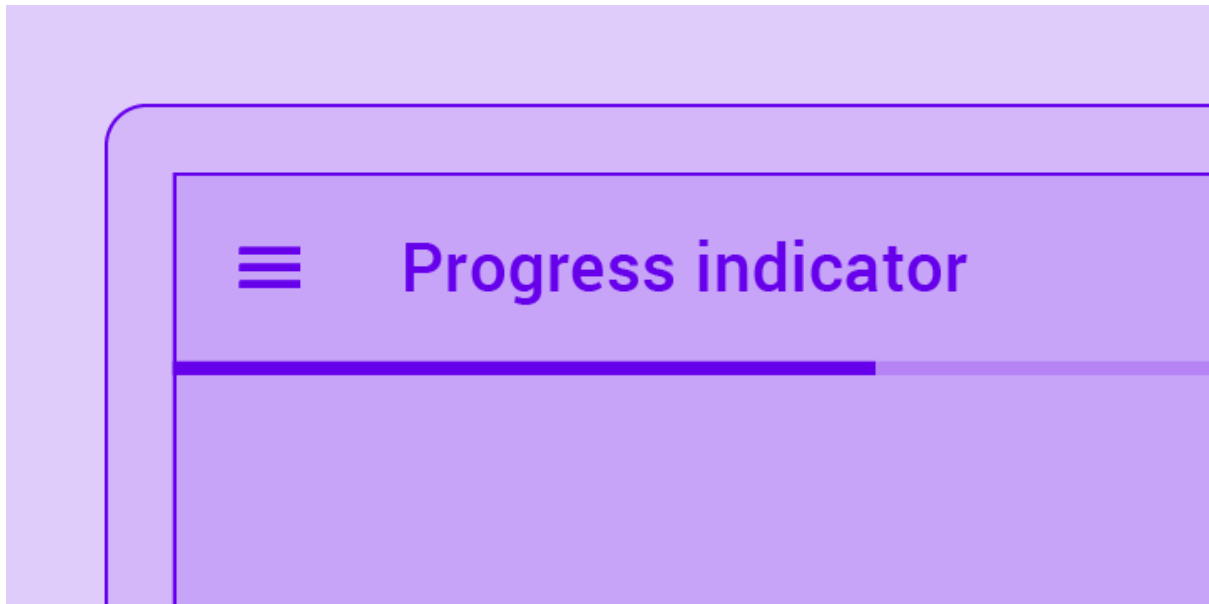
Navigation drawer:

Navigation drawers provide access to destinations in your app



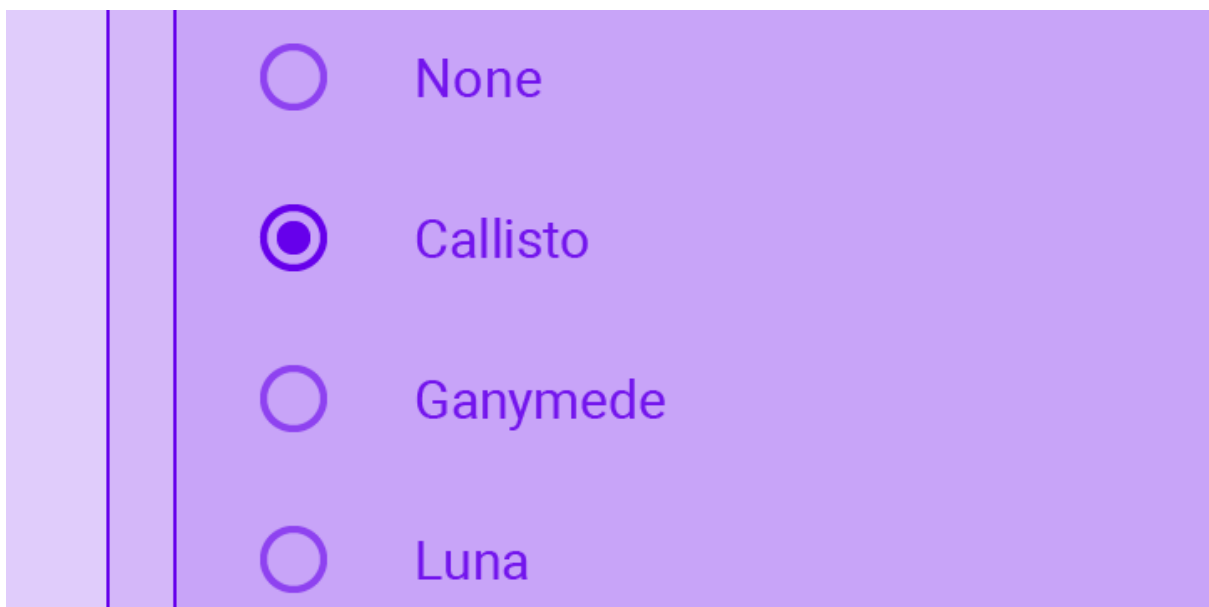
Navigation rail:

The navigation rail provides ergonomic movement between primary destinations in an app



Progress indicators:

Progress indicators express an unspecified wait time or display the length of a process



Radio buttons:

Radio buttons allow the user to select one option from a set



Share



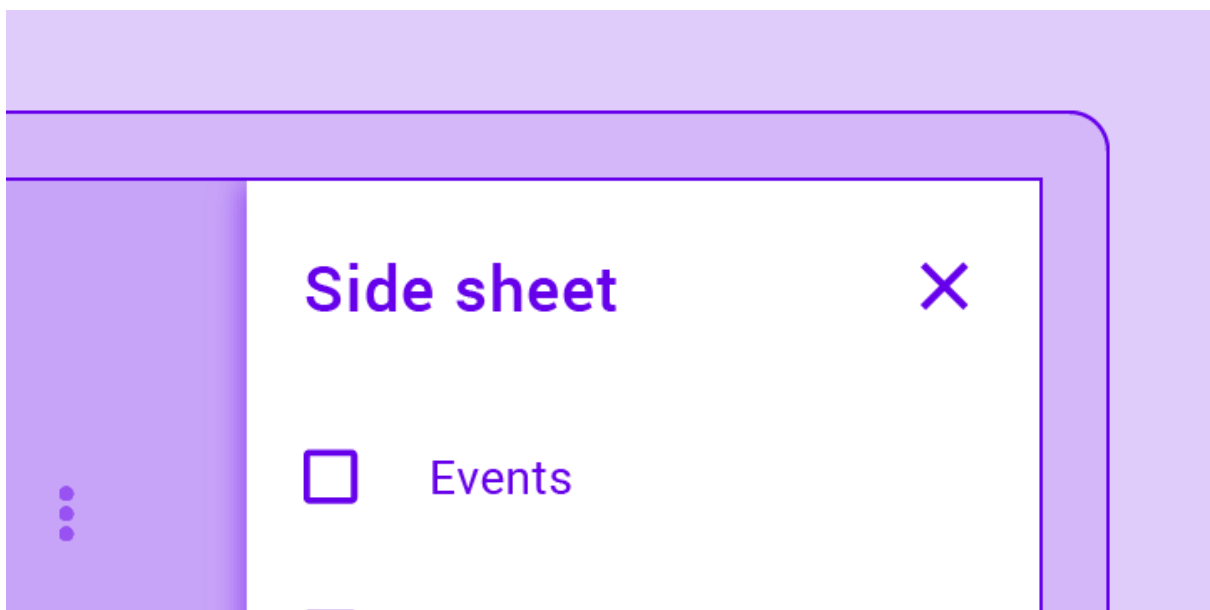
Get link



Edit name

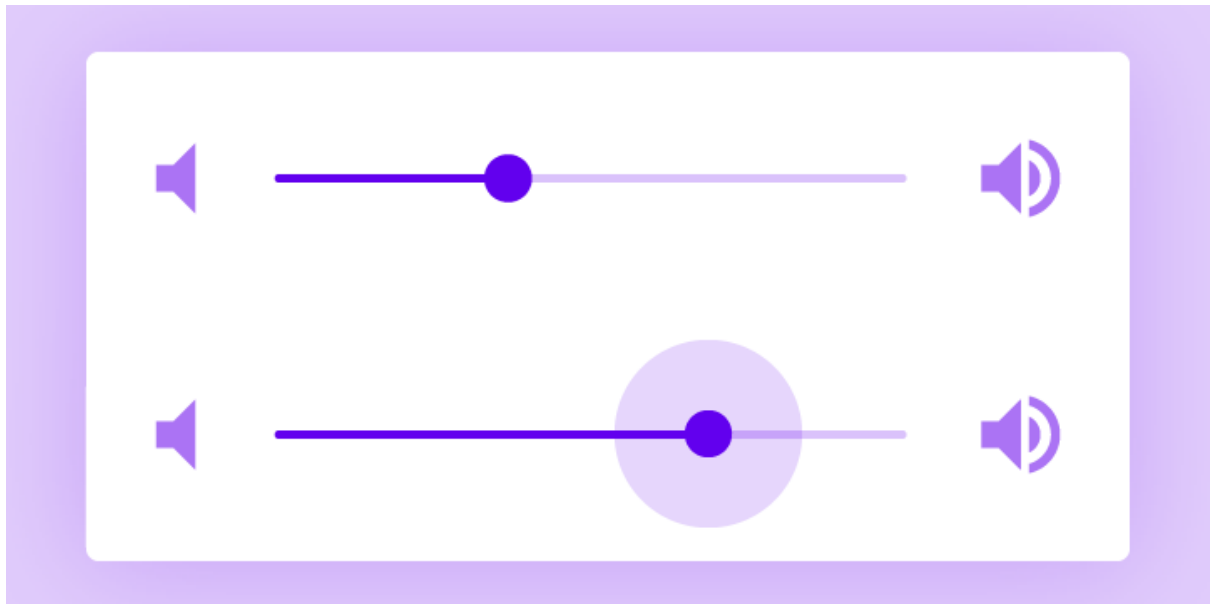
Sheets: bottom

Bottom sheets are surfaces containing supplementary content that are anchored to the bottom of the screen



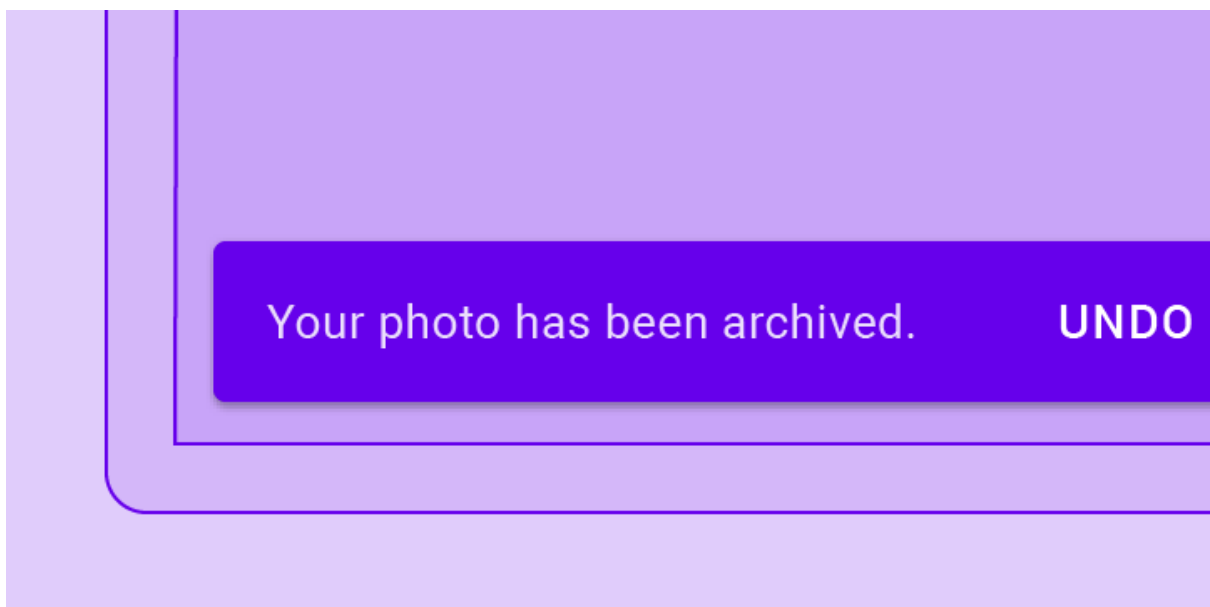
Sheets: side

Side sheets are surfaces containing supplementary content that are anchored to the left or right edge of the screen



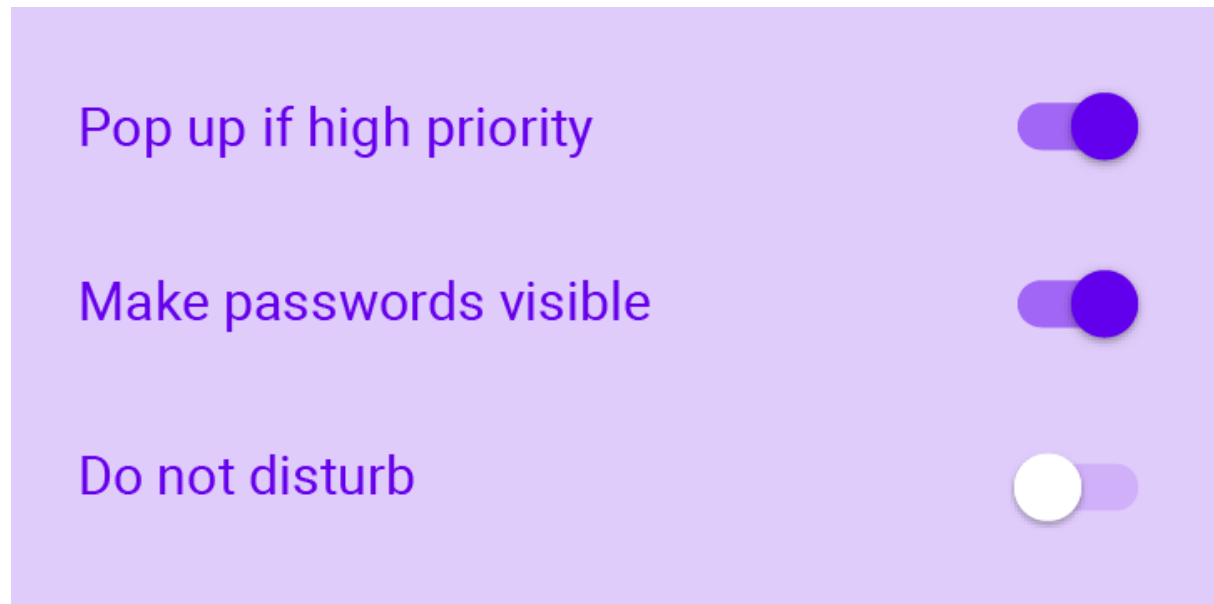
Sliders:

Sliders allow users to make selections from a range of values



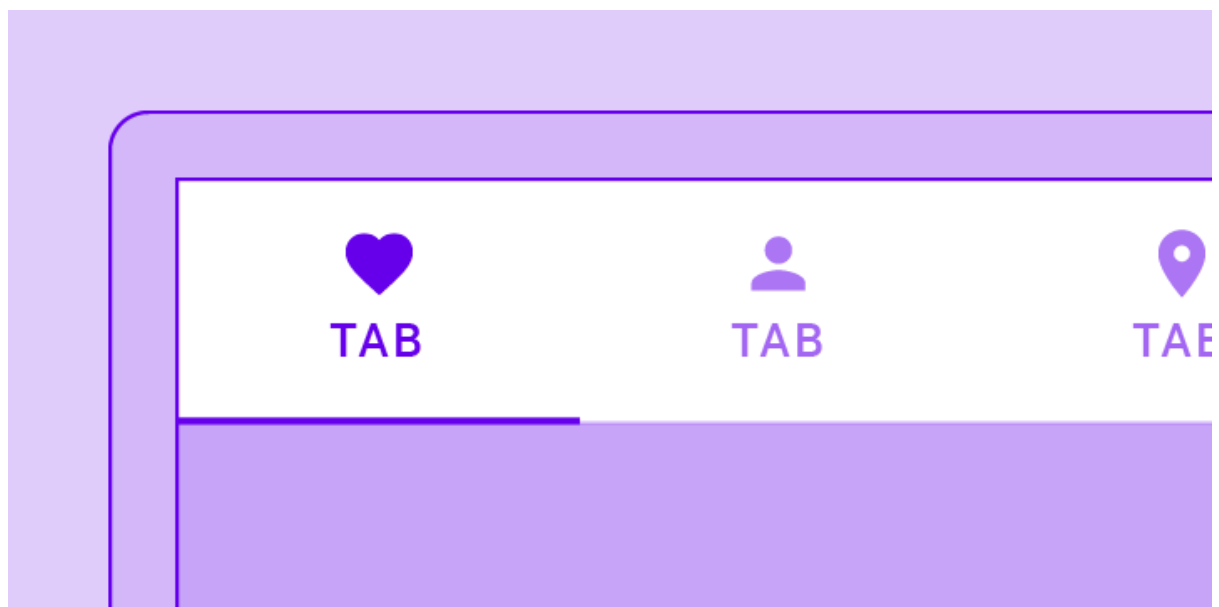
Snack bars:

Snack bars provide brief messages about app processes at the bottom of the screen



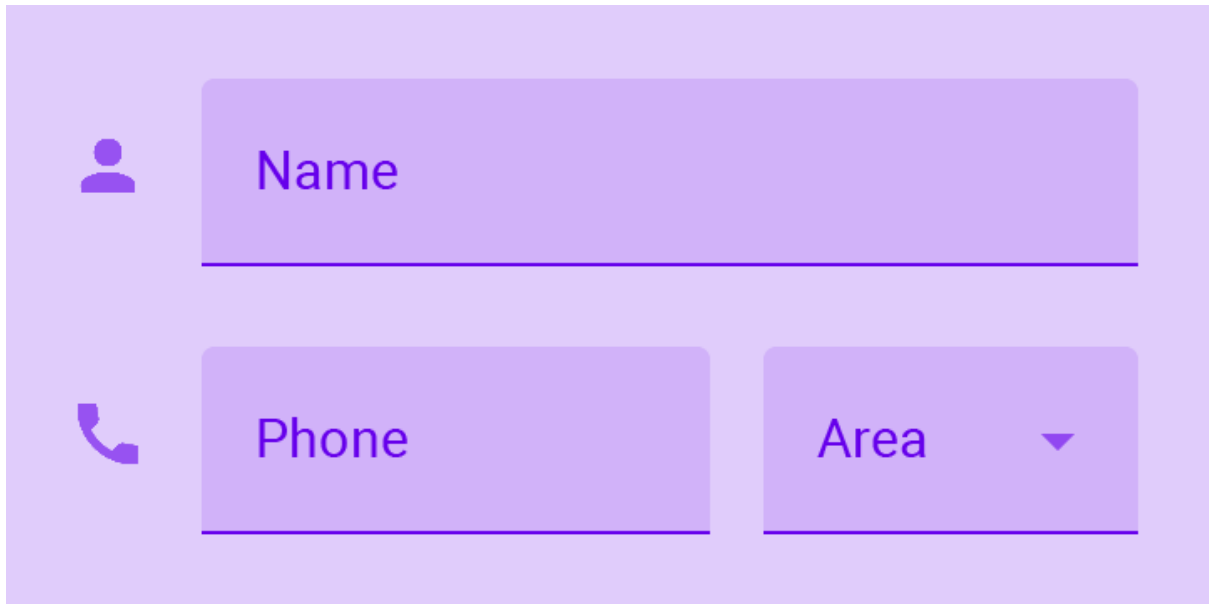
Switches:

Switches toggle the state of a single setting on or off



Tabs:

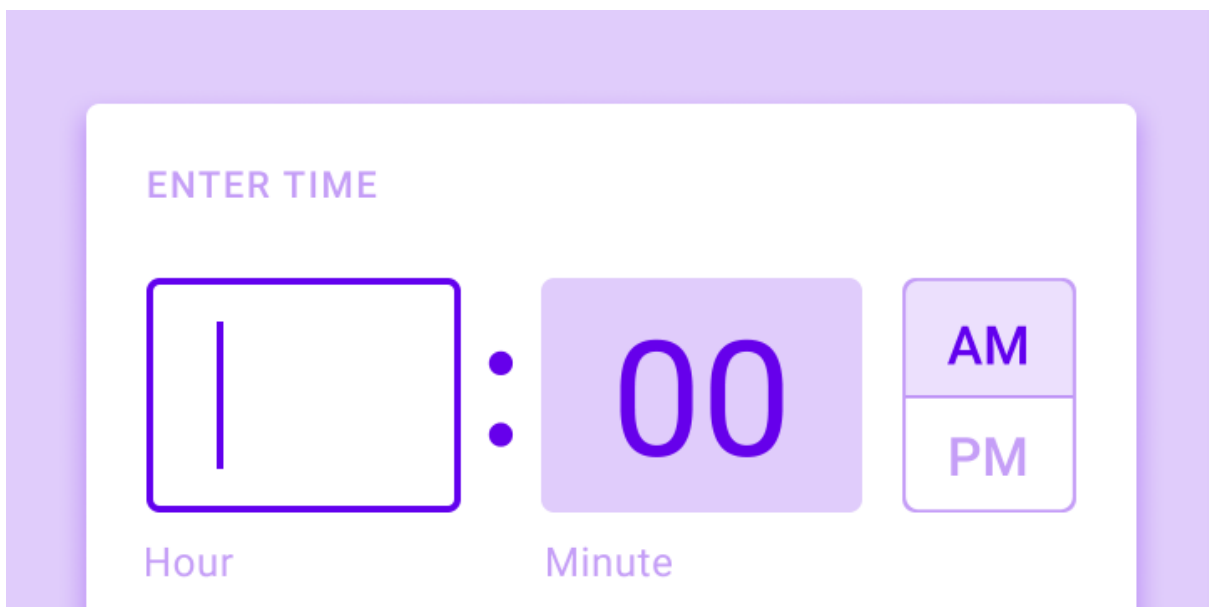
Tabs organize content across different screens, data sets, and other interactions



A form with a light purple background. It contains three input fields. The first field is labeled 'Name' and has a person icon to its left. The second field is labeled 'Phone' and has a telephone icon to its left. The third field is labeled 'Area' and has a dropdown arrow to its right.

Text fields:

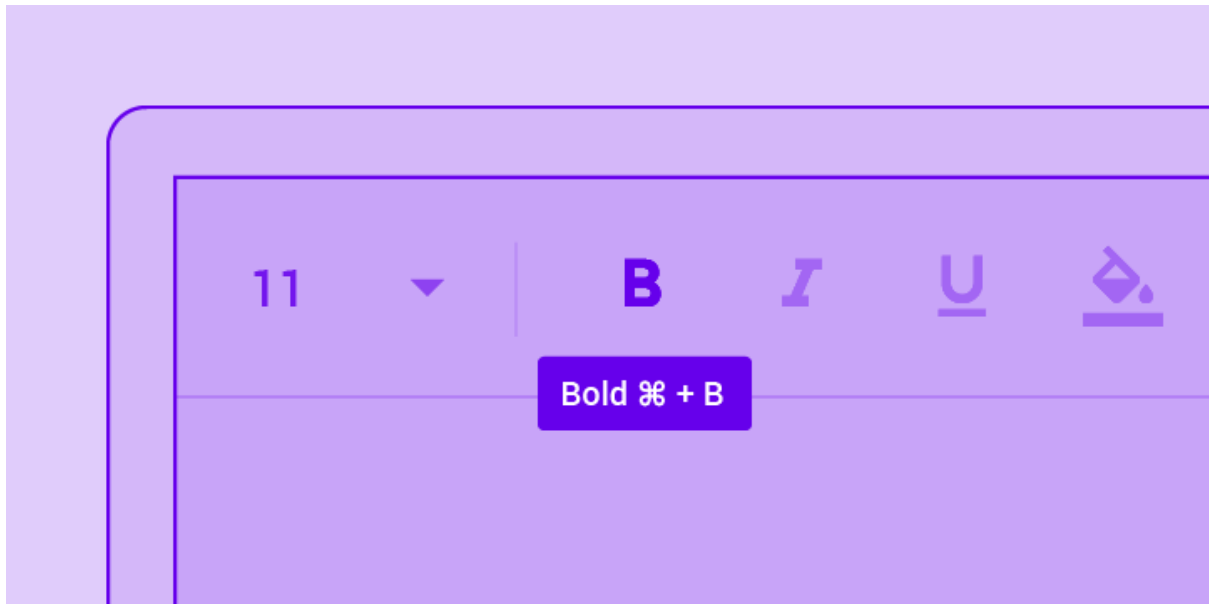
Text fields let users enter and edit text



A time picker form with a light purple background. It contains a white box with the text 'ENTER TIME' inside. Below this box are three input fields. The first field is labeled 'Hour' and contains a single digit '1'. The second field is labeled 'Minute' and contains the text '00'. The third field is a dropdown menu with 'AM' selected and 'PM' as an option.

Time pickers:

Time pickers help users select and set a specific time.



Tooltips:

Tooltips display informative text when users hover over, focus on, or tap an element

Get Started with Firebase Authentication on Android:

Connect your app to Firebase:

If you haven't already, add Firebase to your Android project.

Add Firebase Authentication to your app:

Using the Firebase Android BoM, declare the dependency for the Firebase Authentication Android library in your **module (app-level) Gradle file** (usually app/build.gradle).

JavaKotlin+KTX

```
dependencies {  
    // Import the BoM for the Firebase platform  
    implementation platform('com.google.firebase:firebase-bom:29.0.4')  
  
    // Declare the dependency for the Firebase Authentication library  
    // When using the BoM, you don't specify versions in Firebase library dependencies  
    implementation 'com.google.firebase:firebase-auth'  
}
```

By using the Firebase Android BoM, your app will always use compatible versions of the Firebase Android libraries.

(Alternative) Declare Firebase library dependencies without using the BoM

To use an authentication provider, you need to enable it in the Firebase console. Go to the Sign-in Method page in the Firebase Authentication section to enable Email/Password sign-in and any other identity providers you want for your app.

Prototype and test with Firebase Local Emulator Suite:

Before talking about how your app authenticates users, let's introduce a set of tools you can use to prototype and test Authentication functionality: Firebase Local Emulator Suite. If you're deciding among authentication techniques and providers, trying out different data models with public and private data using Authentication and Firebase Security Rules, or prototyping sign-in UI designs, being able to work locally without deploying live services can be a great idea.

An Authentication emulator is part of the Local Emulator Suite, which enables your app to interact with emulated database content and config, as well as optionally your emulated project resources (functions, other databases, and security rules).

Using the Authentication emulator involves just a few steps:

1. Adding a line of code to your app's test config to connect to the emulator.
2. From the root of your local project directory, running `firebase emulators:start`.
3. Using the Local Emulator Suite UI for interactive prototyping, or the Authentication emulator REST API for non-interactive testing.

A detailed guide is available at [Connect your app to the Authentication emulator](#). For more information, see the [Local Emulator Suite introduction](#).

Now let's continue with [how to authenticate users](#).

Check current auth state:

1. Declare an instance of `FirebaseAuth`.

JavaKotlin+KTX

```
private FirebaseAuth mAuth;
```

EmailPasswordActivity.java

2. In the `onCreate()` method, initialize the `FirebaseAuth` instance.

JavaKotlin+KTX

```
// Initialize Firebase Auth
mAuth = FirebaseAuth.getInstance();
```

EmailPasswordActivity.java

3. When initializing your Activity, check to see if the user is currently signed in.

JavaKotlin+KTX

```
@Override
public void onStart() {
    super.onStart();
    // Check if user is signed in (non-null) and update UI accordingly.
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if(currentUser != null){
        reload();
    }
}
```

```
}  
}
```

EmailPasswordActivity.java

Sign up new users:

Create a new createAccount method that takes in an email address and password, validates them, and then creates a new user with the [createUserWithEmailAndPassword](#) method.

JavaKotlin+KTX

```
mAuth.createUserWithEmailAndPassword(email, password)  
    .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  
        @Override  
        public void onComplete(@NonNull Task<AuthResult> task) {  
            if (task.isSuccessful()) {  
                // Sign in success, update UI with the signed-in user's information  
                Log.d(TAG, "createUserWithEmail:success");  
                FirebaseUser user = mAuth.getCurrentUser();  
                updateUI(user);  
            } else {  
                // If sign in fails, display a message to the user.  
                Log.w(TAG, "createUserWithEmail:failure", task.getException());  
                Toast.makeText(EmailPasswordActivity.this, "Authentication failed.",  
                    Toast.LENGTH_SHORT).show();  
                updateUI(null);  
            }  
        }  
    });
```

EmailPasswordActivity.java

Add a form to register new users with their email and password and call this new method when it is submitted. You can see an example in our quickstart sample.

Sign in existing users:

Create a new signIn method which takes in an email address and password, validates them, and then signs a user in with the [signInWithEmailAndPassword](#) method.

JavaKotlin+KTX

```
mAuth.signInWithEmailAndPassword(email, password)  
    .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  
        @Override  
        public void onComplete(@NonNull Task<AuthResult> task) {  
            if (task.isSuccessful()) {  
                // Sign in success, update UI with the signed-in user's information  
                Log.d(TAG, "signInWithEmail:success");  
                FirebaseUser user = mAuth.getCurrentUser();  
                updateUI(user);  
            } else {
```

```

        // If sign in fails, display a message to the user.
        Log.w(TAG, "signInWithEmail:failure", task.getException());
        Toast.makeText(EmailPasswordActivity.this, "Authentication failed.",
            Toast.LENGTH_SHORT).show();
        updateUI(null);
    }
}
});

```

EmailPasswordActivity.java

Add a form to sign in users with their email and password and call this new method when it is submitted. You can see an example in our quickstart sample.

Access user information:

If a user has signed in successfully you can get their account data at any point with the `getCurrentUser` method.

JavaKotlin+KTX

```

FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
if (user != null) {
    // Name, email address, and profile photo Url
    String name = user.getDisplayName();
    String email = user.getEmail();
    Uri photoUrl = user.getPhotoUrl();

    // Check if user's email is verified
    boolean emailVerified = user.isEmailVerified();

    // The user's ID, unique to the Firebase project. Do NOT use this value to
    // authenticate with your backend server, if you have one. Use
    // FirebaseUser.getIdToken() instead.
    String uid = user.getUid();
}

```

Expected Outcome:

You should be able to develop the Login feature of your application and also integrate Firebase authentication for the same.

Task 3

Creating home activity:

Now we will create an activity where there will be two options **note** and **password**.

- Style the activity so that it looks similar to the one shown below.



[Note: The above image might give you some inspiration to work on your own design.]

Creating note activity:

Now we will create one more activity through which all the notes will be managed and besides add, delete and update functionality should work.

Requirements:

- We need a couple of icons like the navigation icon, note icon, and password icon.
- Create a new component called App-Bar in note activity.
- Add the necessary code in the xml and java files so that all the work is done accurately as mentioned earlier.

Read and Write Data on Android:

This document covers the basics of reading and writing Firebase data.

Firebase data is written to a FirebaseDatabase reference and retrieved by attaching an asynchronous listener to the reference. The listener is triggered once for the initial state of the data and again anytime the data changes.

Note: By default, read and write access to your database is restricted so only authenticated users can read or write data. To get started without setting up Authentication, you can configure your rules for public access. This does make your database open to anyone, even people not using your app, so be sure to restrict your database again when you set up authentication.

Get a Database Reference:

To read or write data from the database, you need an instance of DatabaseReference:

JavaKotlin+KTX

```
private DatabaseReference mDatabase;  
// ...  
mDatabase = FirebaseDatabase.getInstance().getReference();
```

ReadAndWriteSnippets.java

Write data:

Basic write operations

For basic write operations, you can use `setValue()` to save data to a specified reference, replacing any existing data at that path. You can use this method to:

- Pass types that correspond to the available JSON types as follows:
 - String
 - Long
 - Double
 - Boolean
 - Map<String, Object>
 - List<Object>
- Pass a custom Java object, if the class that defines it has a default constructor that takes no arguments and has public getters for the properties to be assigned.

If you use a Java object, the contents of your object are automatically mapped to child locations in a nested fashion. Using a Java object also typically makes your code more readable and easier to maintain. For example, if you have an app with a basic user profile, your `User` object might look as follows:

JavaKotlin+KTX

```
@IgnoreExtraProperties  
public class User {  
  
    public String username;  
    public String email;  
  
    public User() {  
        // Default constructor required for calls to DataSnapshot.getValue(User.class)  
    }  
  
    public User(String username, String email) {  
        this.username = username;  
        this.email = email;  
    }  
  
}
```

User.java

You can add a user with `setValue()` as follows:

JavaKotlin+KTX

```
public void writeNewUser(String userId, String name, String email) {  
    User user = new User(name, email);  
  
    mDatabase.child("users").child(userId).setValue(user);  
}
```

ReadAndWriteSnippets.java

Using `setValue()` in this way overwrites data at the specified location, including any child nodes. However, you can still update a child without rewriting the entire object. If you want to allow users to update their profiles you could update the username as follows:

JavaKotlin+KTX

```
mDatabase.child("users").child(userId).child("username").setValue(name);
```

Read data:

Read data with persistent listeners

To read data at a path and listen for changes, use the `addValueEventListener()` method to add a `ValueEventListener` to a `DatabaseReference`.

Listener	Event callback	Typical usage
<code>ValueEventListener</code>	<code>onDataChange()</code>	Read and listen for changes to the entire contents of a path.

You can use the `onDataChange()` method to read a static snapshot of the contents at a given path, as they existed at the time of the event. This method is triggered once when the listener is attached and again every time the data, including children, changes. The event callback is passed a snapshot containing all data at that location, including child data. If there is no data, the snapshot will return `false` when you call `exists()` and `null` when you call `getValue()` on it.

Important: The **`onDataChange()`** method is called every time data is changed at the specified database reference, including changes to children. To limit the size of your snapshots, attach only at the highest level needed for watching changes. For example, attaching a listener to the root of your database is not recommended.

The following example demonstrates a social blogging application retrieving the details of a post from the database:

JavaKotlin+KTX

```
ValueEventListener postListener = new ValueEventListener() {  
    @Override  
    public void onDataChange(DataSnapshot dataSnapshot) {  
        // Get Post object and use the values to update the UI  
        Post post = dataSnapshot.getValue(Post.class);  
        // ..  
    }  
}
```

```

@Override
public void onCancelled(DatabaseError databaseError) {
    // Getting Post failed, log a message
    Log.w(TAG, "loadPost:onCancelled", databaseError.toException());
}
};
mPostReference.addValueEventListener(postListener);

```

ReadAndWriteSnippets.java

The listener receives a DataSnapshot that contains the data at the specified location in the database at the time of the event. Calling `getValue()` on a snapshot returns the Java object representation of the data. If no data exists at the location, calling `getValue()` returns null.

In this example, `ValueEventListener` also defines the `onCancelled()` method that is called if the read is canceled. For example, a read can be canceled if the client doesn't have permission to read from a Firebase database location. This method is passed a `DatabaseError` object indicating why the failure occurred.

Read data once

Read once using `get()`

The SDK is designed to manage interactions with database servers whether your app is online or offline.

Generally, you should use the `ValueEventListener` techniques described above to read data to get notified of updates to the data from the backend. The listener techniques reduce your usage and billing, and are optimized to give your users the best experience as they go online and offline.

If you need the data only once, you can use `get()` to get a snapshot of the data from the database. If for any reason `get()` is unable to return the server value, the client will probe the local storage cache and return an error if the value is still not found.

Unnecessary use of `get()` can increase use of bandwidth and lead to loss of performance, which can be prevented by using a realtime listener as shown above.

JavaKotlin+KTX

```

mDatabase.child("users").child(userId).get().addOnCompleteListener(new
OnCompleteListener<DataSnapshot>() {
    @Override
    public void onComplete(@NonNull Task<DataSnapshot> task) {
        if (!task.isSuccessful()) {
            Log.e("firebase", "Error getting data", task.getException());
        }
        else {
            Log.d("firebase", String.valueOf(task.getResult().getValue()));
        }
    }
});

```


Read once using a listener

In some cases you may want the value from the local cache to be returned immediately, instead of checking for an updated value on the server. In those cases you can use `addListenerForSingleValueEvent` to get the data from the local disk cache immediately.

This is useful for data that only needs to be loaded once and isn't expected to change frequently or require active listening. For instance, the blogging app in the previous examples uses this method to load a user's profile when they begin authoring a new post.

Updating or deleting data:

Update specific fields

To simultaneously write to specific children of a node without overwriting other child nodes, use the `updateChildren()` method.

When calling `updateChildren()`, you can update lower-level child values by specifying a path for the key. If data is stored in multiple locations to scale better, you can update all instances of that data using data fan-out. For example, a social blogging app might have a `Post` class like this:

JavaKotlin+KTX

`@IgnoreExtraProperties`

`public class Post {`

```
    public String uid;
    public String author;
    public String title;
    public String body;
    public int starCount = 0;
    public Map<String, Boolean> stars = new HashMap<>();
```

```
    public Post() {
        // Default constructor required for calls to DataSnapshot.getValue(Post.class)
    }
```

```
    public Post(String uid, String author, String title, String body) {
        this.uid = uid;
        this.author = author;
        this.title = title;
        this.body = body;
    }
```

`@Exclude`

```
    public Map<String, Object> toMap() {
        HashMap<String, Object> result = new HashMap<>();
        result.put("uid", uid);
        result.put("author", author);
        result.put("title", title);
        result.put("body", body);
        result.put("starCount", starCount);
    }
```

```

        result.put("stars", stars);

        return result;
    }
}

```

Post.java

To create a post and simultaneously update it to the recent activity feed and the posting user's activity feed, the blogging application uses code like this:

JavaKotlin+KTX

```

private void writeNewPost(String userId, String username, String title, String body) {
    // Create new post at /user-posts/$userid/$postid and at
    // /posts/$postid simultaneously
    String key = mDatabase.child("posts").push().getKey();
    Post post = new Post(userId, username, title, body);
    Map<String, Object> postValues = post.toMap();

    Map<String, Object> childUpdates = new HashMap<>();
    childUpdates.put("/posts/" + key, postValues);
    childUpdates.put("/user-posts/" + userId + "/" + key, postValues);

    mDatabase.updateChildren(childUpdates);
}

```

ReadAndWriteSnippets.java

This example uses `push()` to create a post in the node containing posts for all users at `/posts/$postid` and simultaneously retrieve the key with `getKey()`. The key can then be used to create a second entry in the user's posts at `/user-posts/$userid/$postid`.

Using these paths, you can perform simultaneous updates to multiple locations in the JSON tree with a single call to `updateChildren()`, such as how this example creates the new post in both locations. Simultaneous updates made this way are atomic: either all updates succeed or all updates fail.

Add a Completion Callback

If you want to know when your data has been committed, you can add a completion listener. Both `setValue()` and `updateChildren()` take an optional completion listener that is called when the write has been successfully committed to the database. If the call was unsuccessful, the listener is passed an error object indicating why the failure occurred.

JavaKotlin+KTX

```

mDatabase.child("users").child(userId).setValue(user)
    .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Write was successful!
            // ...
        }
    })

```

```

.addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        // Write failed
        // ...
    }
});

```

ReadAndWriteSnippets.java

Delete data

The simplest way to delete data is to call `removeValue()` on a reference to the location of that data.

You can also delete by specifying null as the value for another write operation such as `setValue()` or `updateChildren()`. You can use this technique with `updateChildren()` to delete multiple children in a single API call.

Detach listeners:

Callbacks are removed by calling the `removeEventListener()` method on your Firebase database reference.

If a listener has been added multiple times to a data location, it is called multiple times for each event, and you must detach it the same number of times to remove it completely.

Calling `removeEventListener()` on a parent listener does not automatically remove listeners registered on its child nodes; `removeEventListener()` must also be called on any child listeners to remove the callback.

Save data as transactions:

When working with data that could be corrupted by concurrent modifications, such as incremental counters, you can use a transaction operation. You give this operation two arguments: an update function and an optional completion callback. The update function takes the current state of the data as an argument and returns the new desired state you would like to write. If another client writes to the location before your new value is successfully written, your update function is called again with the new current value, and the write is retried.

For instance, in the example social blogging app, you could allow users to star and unstar posts and keep track of how many stars a post has received as follows:

JavaKotlin+KTX

```

private void onStarClicked(DatabaseReference postRef) {
    postRef.runTransaction(new Transaction.Handler() {
        @Override
        public Transaction.Result doTransaction(MutableData mutableData) {
            Post p = mutableData.getValue(Post.class);
            if (p == null) {
                return Transaction.success(mutableData);
            }
        }
    });
}

```

```

        if (p.stars.containsKey(getUid())) {
            // Unstar the post and remove self from stars
            p.starCount = p.starCount - 1;
            p.stars.remove(getUid());
        } else {
            // Star the post and add self to stars
            p.starCount = p.starCount + 1;
            p.stars.put(getUid(), true);
        }

        // Set value and report transaction success
        mutableData.setValue(p);
        return Transaction.success(mutableData);
    }

    @Override
    public void onComplete(DatabaseError databaseError, boolean committed,
        DataSnapshot currentData) {
        // Transaction completed
        Log.d(TAG, "postTransaction:onComplete:" + databaseError);
    }
});
}

```

ReadAndWriteSnippets.java

Using a transaction prevents star counts from being incorrect if multiple users star the same post at the same time or the client had stale data. If the transaction is rejected, the server returns the current value to the client, which runs the transaction again with the updated value. This repeats until the transaction is accepted or too many attempts have been made.

Note: Because **doTransaction()** is called multiple times, it must be able to handle **null** data. Even if there is existing data in your remote database, it may not be locally cached when the transaction function is run, resulting in **null** for the initial value.

Atomic server-side increments

In the above use case we're writing two values to the database: the ID of the user who stars/unstars the post, and the incremented star count. If we already know that user is starring the post, we can use an atomic increment operation instead of a transaction.

JavaKotlin+KTX

```

private void onStarClicked(String uid, String key) {
    Map<String, Object> updates = new HashMap<>();
    updates.put("posts/"+key+"/stars/"+uid, true);
    updates.put("posts/"+key+"/starCount", ServerValue.increment(1));
    updates.put("user-posts/"+uid+"/"+key+"/stars/"+uid, true);
    updates.put("user-posts/"+uid+"/"+key+"/starCount", ServerValue.increment(1));
    mDatabase.updateChildren(updates);
}

```

ReadAndWriteSnippets.java

This code does not use a transaction operation, so it does not automatically get re-run if there is a conflicting update. However, since the increment operation happens directly on the database server, there is no chance of a conflict.

If you want to detect and reject application-specific conflicts, such as a user starring a post that they already starred before, you should write custom security rules for that use case.

Work with data offline:

If a client loses its network connection, your app will continue functioning correctly.

Every client connected to a Firebase database maintains its own internal version of any data on which listeners are being used or which is flagged to be kept in sync with the server. When data is read or written, this local version of the data is used first. The Firebase client then synchronizes that data with the remote database servers and with other clients on a "best-effort" basis.

As a result, all writes to the database trigger local events immediately, before any interaction with the server. This means your app remains responsive regardless of network latency or connectivity.

Once connectivity is reestablished, your app receives the appropriate set of events so that the client syncs with the current server state, without having to write any custom code.

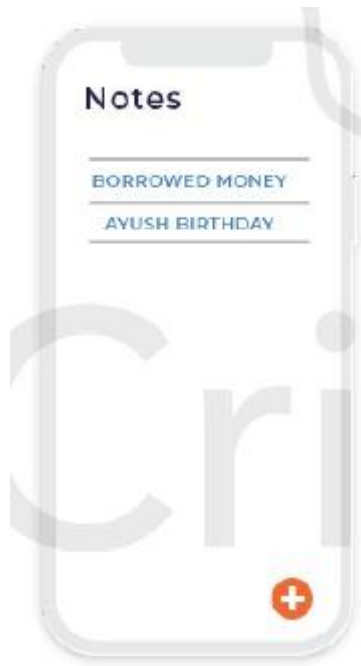
We'll talk more about offline behavior in [Learn more about online and offline capabilities](#).

Tip:

You can use Listview or Recycleview to show notes as mentioned earlier. Using Listview or Recycleview, you can only show the title of the notes and after clicking on those titles, the rest of the functionality can be done.

Expected Outcome:

On completion of this milestone, the note manager of your application should look similar to the one shown below.



Task 4

Creating password activity:

Now we will create one more activity through which all the passwords will be managed and besides add and delete functionality should work feasibly.

Requirements:

- We need a couple of icons like the password icon and the add icon.
 - Create a new component called App-Bar in password activity.
 - Add the necessary code in the xml and java files so that all the work is done accurately as mentioned earlier.
 - Create a new function for generating password
-
- You should create a new function that will measure the strength of any password.

Tip:

You can use Listview or Recycleview to show passwords like notes as mentioned earlier. Using Listview or Recycleview you can only show the field name of the passwords and after clicking on those field names, the rest of the functionality can be done.

Expected Outcome:

On completion of this milestone, the password manager of your application should look similar to the one shown below



Key References:

- [Android Studio](#)
- [Firebase](#)

Author's: *Pramod Joshi*
Mukul Kandpal
Hem Upadhyay

B.Tech.(CSE) 2nd year
Graphic Era Hill University, Bhimtal.
