GitHub Username: mountis

Scrib

Description

Scrib is a note taking with a simple user interface design that allows users to take notes, sketch notes, and make a to do list.

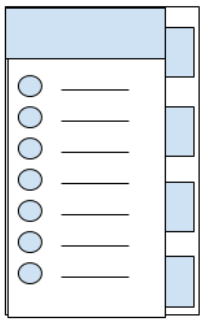
Intended User

This is an App for anyone looking for a notes taking app that has a simple, and easy to use UI with many features.

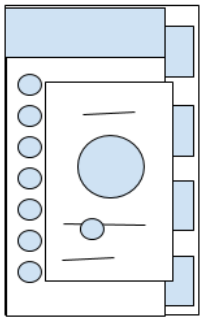
Features

- Material Design interface
- Users can add, modify, archive, trash, share, merge, search notes and delete notes
- Image, audio and generic file attachments capabilities
- Make it easy to keep track of your notes using tags and categories
- Create a To-do list
- App includes a Sketch-note mode
- Users can put notes shortcut on home screen
- Users can export/import notes to backup
- Multiple widgets

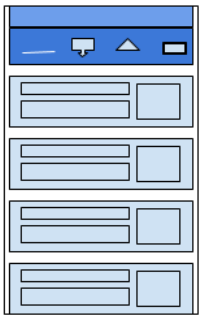
User Interface Mocks



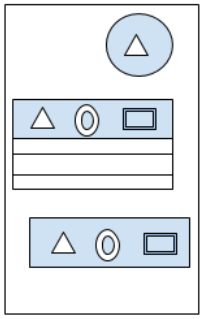
Navigation Drawer Layout



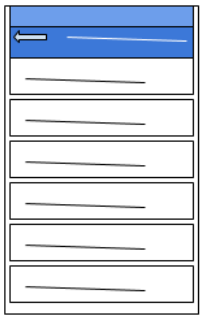
Color Picker Dialogue



Notes List View



Widgets



Settings

Key Considerations

How will your app handle data persistence?

This app utilizes a content provider.

Describe any edge or corner cases in th

User actions of the app

- Select notes to tag, archive or delete

Note searching feature

- Insert text to find it in title or content

Notes sorting preferences

- Manage by ordering with criteria

Notes categorization

- Filter notes tagged, archived or with reminder set using navigation

Color coding for notes

- Choose a color for your tags to highlight notes in lists

Editing features

- Save, share, tag, archive, mask, delete and more...

Attachments types

- Enrich your notes with photos, videos, audio recordings and geo-location

Hyperlinks

- Web url and e-mail auto recognition

Personalized settings

- A lot of available settings to personalize your experience

Easy access to other features

- Resizable widgets, DashClock extension, Google Now integration

Describe any libraries you'll be using and share your reasoning for including them.

- Glide to handle the loading and caching of images.
- RxJava a library for composing asynchronous and event-based programs using observable sequences for the Java VM.
- RxAndroid a library that make writing reactive components in Android applications easy and hassle-free
- Butterknife a Field and method binding for Android views which uses annotation processing to generate boilerplate code for you.
- 'Com.github.flavienlaurent.datetimepicker:library:0.0.2' is a library which contains the beautiful DatePicker that can be seen in the new Google Agenda app.
- 'com.larswerkman:HoloColorPicker:1.4' You can now set the Saturation and Value of a color. Also its possible to set the Opacity for a color. You can also set the last selected color and see the difference with the new selected color.

- 'Com.jakewharton:disklrucache:2.0.2' A cache that uses a bounded amount of space on a filesystem. Each cache entry has a string key and a fixed number of values.
- 'de.keyboardsurfer.android.widget:crouton:1.8.4@aar' Context sensitive notifications for Android
- 'Be.billington.calendar.recurrencepicker:library:1.1.1' Google Calendar Recurrence picker
- 'De.greenrobot:eventbus:2.4.0' Event bus for Android and Java that simplifies communication between Activities, Fragments, Threads, Services, etc. Less code, better quality.
- 'Com.getbase:floatingactionbutton:1.10.1' Yet another library for drawing Material Design promoted actions.
- 'com.nhaarman.listviewanimations:lib-core:3.1.0@aar' An Android library which allows developers to easily add animations to ListView items
- 'com.github.afollestad.material-dialogs:core:0.8.5.6@aar' A beautiful and fluid dialogs API for Kotlin & Android.
- 'Org.mnode.ical4j:ical4j:1.0.6' A Java library for reading and writing iCalendar (*.ics) files
- 'Com.pnikosis:materialish-progress:1.5' A material style progress wheel compatible with 2.3
- 'Com.github.paolorotolo:appintro:1.3.0' AppIntro library
- 'com.tbruyelle.rxpermissions:rxpermissions:0.4.2@aar'
- 'org.ocpsoft.prettytime:prettytime:3.2.7.Final' Social Style Date and Time Formatting for Java
- 'com.github.federicoiosue:SimpleGallery:2.0.0' Android library to include image gallery into your app
- 'com.github.federicoiosue:Springpad-Importer:1.0.1' Springpad notes importer tool
- 'Com.github.federicoiosue:checklistview:3.2.1' Library to convert an EditText into a View capable of acting as checklist
- 'Io.nlopez.smartlocation:library:3.2.4' This library handles all the boilerplate and repetitive code used when playing with Locations in Android projects.
- 'com.github.federicoiosue:analitica:0.0.3:googleAnalyticsRelease@aar' Android library to manage both Google Analytics and Piwik services
- 'Com.google.android.gms:play-services-analytics:9.0.2' Play Services Analytics

Describe how you will implement Google Play Services or other external services.

This app will utilize Google Analytics, and Google Places Api. The app is also Google Now integrated.

Task 1: Project Setup

When it comes to developing Android apps there's a pattern or logic I tend to follow in building the app from start to finish. First and foremost I write out the pseudocode for the app to give me an idea on how the app will flow, how many activities I will need? How many layouts I will use? Which libraries to incorporate in the app? What will my values folder look like? And how to configure my gradle script to build a working app!
At this phase I pull out my blueprint of app development, and my pseudocode, then divide the project in 3 steps to help me stay on track, and have a solid building foundation.

- First I start with the gradle script where I configure the app (signing, repositories, plugins, build variants, dependencies, and so on…)
- Second I sketch out the UI's and design them to look like how i have them in mind with functionalities and everything. I set up my drawable, anim, xml, values, and menu folder to the app's specification.
- Last I start to build the code to make the app functional. I figure what activities i need, fragments to build, adapters to add, models to use, utils, database, async tasks, and any classes that I will need to make the whole thing work.

To make sure my app works perfectly I generate tests for my main functions, and run the app to make sure it is fully functioning before I deploy it to the Google Store.

Task 2: Implement UI for Each Activity and Fragment

1. Create drawable resources for my app by building for multiple screens.
2. Develop my values folders (colors, style, strings, dimensions, integers, analytics, etc…)
3. Create my XML files.
4. Build the layout for each activity, fragment, and component.
5. Create my anim, and animator folders to make the UI more appealing and user friendly.

Task 3: Configure Gradle Script

1. Add plugins
2. Set up the signing configuration, default configurations, packaging options, build types, repositories, and compile options
3. Include the dependencies for the libraries I will be using in the project
4. Configure gradle properties
5. Sync project to make sure it is good standing

Task 4: Code Buildup

1. Write the code for the different activities included in the app
2. Create the supporting fragments, adapters, views, and models
3. Develop a database, async tasks, listeners, and helper classes
4. Add any miscellaneous files relating to the functionality of the app
5. Run the app to make sure the code is working and in sync with the UI
6. Deploy the app on an emulator, or physical phone to test the functionalities

Task 5: Generate APK and deploy to App store

1. Get the signed release APK of the app
2. Follow all the steps to App store deployment
3. Deploy app to app store