

Kata Pengantar

Segala puji bagi Allah Subhanahu Wa Ta'ala, Atas segala rahmat dan karunia-Nya. Atas segala nikmat yang telah diberikan yaitu nikmat sehat dan nikmat iman, sehingga kita bisa menambah wawasan dan ilmu yang bermanfaat.

Saya ucapan juga terima kasih kepada pihak-pihak yang membantu saya dalam menulis buku ini, yaitu para ustadz dan teman-teman sekalian yang membantu dan mendukung saya dalam menyelesaikan buku ini.

Sebagai manusia, saya sadar tidak luput dari kesalahan dan kekeliruan, baik dari segi penulisan atau pun bahasa dalam penyampaian. Oleh sebab itu, saya mohon agar pembaca memberikan kritik dan saran terhadap buku ini, agar saya dapat berkembang lebih baik daripada sebelumnya.

Demikian buku ini dibuat, dengan harapan agar dapat menambah wawasan serta dapat meningkatkan pengembangan keahlian pembaca dalam bidang ini. Terima kasih.

Bekasi, 31 Oktober 2023

Muhammad Akna Mafaid I A.

Daftar Isi

Kata Pengantar	i
Daftar Isi	ii
Daftar Gambar	v
Daftar Code Program	viii
.....	1
Bagian 1 : Android	1
1.1 Sekilas Tentang Android	2
1.2 Google Play Store	3
1.3 Android Studio	4
Bagian 2 : UI dan UX	6
2.1 Apa itu UI dan UX ?	7
2.2 Standar UI / UX	8
2.3 Tema Aplikasi	12
Bagian 3 : Design	14
3.1 Framework	15
3.2 Desain	16
3.3 Color Palette	19

Bagian 4 : Membuat Aplikasi (Auth)	14
4.1 Memulai Project.....	15
4.2 Menyiapkan Database	24
4.3 Data	30
4.4 Auth.....	39
Bagian 5 : Membuat Aplikasi (Post).....	66
5.1 Menyiapkan <i>repository</i> dan local <i>database</i>	67
5.2 Worker	77
5.3 CRUD dengan Worker.....	82
5.4 Get <i>data</i> dengan realtime	95
5.5 Implementasi worker pada <i>repository</i>.....	98
5.6 Membuat Sort & Order by	105
Bagian 6 : Membuat Aplikasi (Screen)	116
6.1 HomeScreen	117
6.2 EditorScreen	137
6.3 LoginScreen.....	149
6.4 RegisterScreen	155
6.5 Theme	164

6.6 SettingsScreen	181
6.7 EditorScreen (2)	189
6.8 SearchScreen	192
6.9 OnBoarding.....	198
6.10 DetailScreen.....	206
Bagian 7 : Menyelesaikan Aplikasi	210
 7.1 Application	211
 7.2 App.....	213
 7.3 MainActivity	217
 7.4 AndroidManifest	218
 7.5 SplashScreen & Icon	220
 7.5 Preview	228
Tentang Penulis.....	233
Referensi	234
Teori	234
Code Program.....	234

Daftar Gambar

Gambar 1 – Logo Android	2
Gambar 2 – Logo Google Play.....	3
Gambar 3 – Logo Android Studio.....	4
Gambar 4 – Logo Kotlin	5
Gambar 5 - Android vs Ios UI	7
Gambar 6 https://m3.material.io/	9
Gambar 7 Framework yang didukung.....	10
Gambar 8 Implementasi komponen.....	11
Gambar 9 Cara penggunaan yang baik pada komponen	11
Gambar 10 Tema Terang vs Tema Gelap.....	12
Gambar 11 Color palette tema terang dan gelap	13
Gambar 12 Framework aplikasi saat ini	15
Gambar 13 Desain jadi vs kerangka.....	16
Gambar 14 Figma Logo	17
Gambar 15 Preview Figma.....	17
Gambar 16 Desain yang sudah jadi	18
Gambar 17 Contoh color pallete	19
Gambar 18 Theme Builder	20
Gambar 19 Primary.....	20
Gambar 20 Dialog Primary	21
Gambar 21 Tombol Export.....	22
Gambar 22 Export Menu	22
Gambar 23 Download Android Studio.....	16
Gambar 24 Fitur Android Studio.....	16
Gambar 25 Install Android Studio.....	17

Gambar 26 Supabase	25
Gambar 27 Dahsboard Project	25
Gambar 28 Project Supabase.....	26
Gambar 29 Beranda Tabel.....	27
Gambar 30 Editor Table	27
Gambar 31 Table Name	28
Gambar 32 Table Column.....	29
Gambar 33 Table User	29
Gambar 34 https://material-foundation.github.io/material-theme-builder	165
Gambar 35 Untuk membuka button eksport.....	165
Gambar 36 Export	166
Gambar 37 Pilih Jetpack Compose	166
Gambar 38 VS Code di microsoft store	167
Gambar 39 Open with vsCode.....	167
Gambar 40 https://github.com/canopas/rich-editor-compose	175
Gambar 41 Folder parser	175
Gambar 42 JsonEditorParser	176
Gambar 43 RichTextSpanAdapter.....	177
Gambar 44 Membuat draft baru.....	221
Gambar 45 Ukuran logo.....	221
Gambar 46 Pastikan gambar di group	222
Gambar 47 Menambahkan vector image	224
Gambar 48 Logo ukuran 512 x 512.....	226
Gambar 49 Package mipmap	226

Gambar 50 Menambah image asset	227
Gambar 51 Home Screen.....	228
Gambar 52 EditorScreen.....	229
Gambar 53 HomeScreen (2).....	229
Gambar 54 Profile Screen (wkwkwk).....	230
Gambar 55 SearchScreen.....	230
Gambar 56 LoginScreen	231
Gambar 57 RegisterScreen.....	232

Daftar Code Program

Code Program 1 Welcome Screen.....	18
Code Program 2 Template menu	19
Code Program 3 Ganti name dan package	20
Code Program 4 Main Activity.....	21
Code Program 5 File menu.....	22
Code Program 6 Menu settings	23
Code Program 7 New UI enable	24
Code Program 8 Gradle	30
Code Program 9 Menambah Package.....	31
Code Program 10 Menamai package	31
Code Program 11 Package data	32
Code Program 12 Gradle Module	33
Code Program 13 Menambah depedency	35
Code Program 14 Package source di dalam	35
Code Program 15 Package remote di dalam source.....	35
Code Program 16 Package serializable di dalam remote	35
Code Program 17 PostDto	36
Code Program 18 User	37
Code Program 19 Table post supabase.....	37
Code Program 20 Table user supabase.....	38
Code Program 21 Foreign key post ke user.....	39
Code Program 22 Paclage di.....	40
Code Program 23 Membuat object app module	40
Code Program 24 Provide supabase client	41
Code Program 25 Mendapatkan supabase url dan key.....	42

Code Program 26 Local properties.....	43
Code Program 27 Variabel supabase url dan key.....	44
Code Program 28 Build config	44
Code Program 29 Mendeklarasikan variabel build config	45
Code Program 30 Menambahkan "" pada value.....	45
Code Program 31 Icon "palu" gradle	45
Code Program 32 Memanggil variabel build config.....	46
Code Program 33 Import build config	46
Code Program 34 Package repository.....	47
Code Program 35 Class interface user repository pada domain.....	47
Code Program 36 Class UserRepoImpl pada data	47
Code Program 37 Register & Login.....	48
Code Program 38 Class UserRepoImpl yang masih kosong	49
Code Program 39 Extend class ke interface	50
Code Program 40 Context menu.....	50
Code Program 41 Dialog implement members.....	51
Code Program 42 Override fun dari members fun interface	52
Code Program 43 Provide user repo	52
Code Program 44 Memanggil client pada parameter.....	53
Code Program 45 Logika register	53
Code Program 46 "username" adalah key dan val username adalah value	55
Code Program 47 Object local user.....	55
Code Program 48 Variabel di dalam Local User	56
Code Program 49 Logika Login	57

Code Program 50 Function untuk menambahkan data dari auth	58
Code Program 51 Trigger untuk menjalankan function yang dibuat.....	58
Code Program 52 Package usecase di dalam package domain.....	59
Code Program 53 Isi package use case	59
Code Program 54 Class AuthUseCase.....	60
Code Program 55 Class interface GlobalUseCase	61
Code Program 56 Class UseCaseInteractor	61
Code Program 57 Provide UseCase di AppModule	62
Code Program 58 Package presentation	62
Code Program 59 Package login & register	63
Code Program 60 Screen & View Model.....	63
Code Program 61 LoginViewModel	64
Code Program 62 Class Interface Diary repository.....	68
Code Program 63 Class DiaryRepolmpl	68
Code Program 64 Package local dan isinya.....	69
Code Program 65 PostEntity	70
Code Program 66 DiaryDao	71
Code Program 67 DiaryDatabase	72
Code Program 68 Provide database di AppModule	73
Code Program 69 Class Post	73
Code Program 70 Class Post	74
Code Program 71 fun toPostEntity() pada Class PostDto	75
Code Program 72 Fun toPost() pada Class PostEntity	76

Code Program 73 DiaryRepoImpl	77
Code Program 74 Provide Diary Repo di AppModule	77
Code Program 75 Membuat package utils	78
Code Program 76 Membuat File Constant	78
Code Program 77 Key untuk data worker	79
Code Program 78 Package worker	80
Code Program 79 Class yang ada di dalam package worker	80
Code Program 80 NotificationWorker	81
Code Program 81 InsertDataWorker	84
Code Program 82 UpdateDataWorker.....	87
Code Program 83 DeleteDataWorker	88
Code Program 84 DeleteAllDraftWorker	90
Code Program 85 DeleteAllDataWorker	90
Code Program 86 DeleteDataWorker	91
Code Program 87 InsertDraftWorker	91
Code Program 88 UpdateDraftWorker.....	92
Code Program 89 UpdateUserWorker	94
Code Program 90 Private val channel	96
Code Program 91 Fun fetch() untuk mendapatkan data secara realtime dari supabase	97
Code Program 92 Mendapatkan draft, tidak realtime.....	97
Code Program 93 Mengambil data favorite.....	98
Code Program 94 Fun unsubscribe() untuk berhenti mengikuti pembaruan.....	98
Code Program 95 Deklarasi Work manager dan constraint..	99

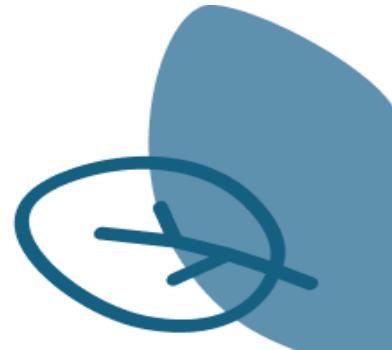
Code Program 96 override suspend fun insert post.....	99
Code Program 97 File helper.....	99
Code Program 98 Ekstensi toDataWorker	100
Code Program 99 Override fun publish, merubah draft menjadi posts	100
Code Program 100 Override suspend fun update.....	101
Code Program 101 Override suspend fun delete post	101
Code Program 102 deleteFavorite	102
Code Program 103 insertDraft.....	102
Code Program 104 Override suspend fun delete all post ...	102
Code Program 105 deleteAllDraft	103
Code Program 106 deleteFavoriteAll.....	103
Code Program 107 Diary repository	104
Code Program 108 Object Network.....	105
Code Program 109 Membuat sealed class baru.....	106
Code Program 110 Sealed Class Diary Order	106
Code Program 111 Sealed Class OrderBy	106
Code Program 112 Diary use case.....	109
Code Program 113 Package-package pada presentation ..	110
Code Program 114 HomeViewModel.....	112
Code Program 115 CreateViewModel	113
Code Program 116 DetailViewModel	114
Code Program 117 SearchViewModel	114
Code Program 118 SettingsViewModel	115
Code Program 119 Package component	117
Code Program 120 File-file pada setiap package	118

Code Program 121 CustomRadioButton.....	119
Code Program 122 Menggunakan preview	120
Code Program 123 PostItem	122
Code Program 124 OrderSection.....	124
Code Program 125 SaveDialog	125
Code Program 126 Draft.kt.....	129
Code Program 127 Favorite.kt.....	132
Code Program 128 Post.kt	136
Code Program 129 HomeScreen	136
Code Program 130 File EditorControls	143
Code Program 131 EmojiPickerBottomSheet.....	144
Code Program 132 Tambahkan pada parameter.....	144
Code Program 133 Menerima data yang dikirim dari screen lain	144
Code Program 134 EditorScreen	148
Code Program 135 Fun String.isValidEmail	149
Code Program 136 Ekstensi untuk memeriksa password ..	149
Code Program 137 LoginScreen	155
Code Program 138 TextFieldEmail	158
Code Program 139 Memeriksa error pada TextField password	159
Code Program 140 Mengatur visibility untuk password.....	159
Code Program 141 Var baru untuk menampilkan error pada textfield name.....	160
Code Program 142 Mengatur text yang ditampilkan ketika error.....	160

Code Program 143 Memeriksa apakah semua textfield tidak kosong ?	161
Code Program 144 Text pada button.....	161
Code Program 145 Momen ketika success.....	162
Code Program 146 Mengatur error pada DisplayResult	162
Code Program 147 Textbutton untuk berpindah ke LoginScreen	163
Code Program 148 Error pada HomeScreen	164
Code Program 149 GlobalPreferences.....	164
Code Program 150 Susunan color Light	168
Code Program 151 Color.kt	168
Code Program 152 Contoh, primaryLight diubah menjadi primaryLightMountain.....	169
Code Program 153 lightScheme	169
Code Program 154 Contoh, lightScheme diubah menjadi MountainLightScheme	170
Code Program 155 Enum Class AppTheme	171
Code Program 156 Package drawable	171
Code Program 157 Menambahkan gambar ke drawable ...	172
Code Program 158 Mengubah nama file menggunakan lowercase dan underscore	172
Code Program 159 Contoh enum	173
Code Program 160 GlobalPreferences.....	173
Code Program 161 GlobalState	174
Code Program 162 State untuk rich editor	174
Code Program 163 Package parser	177

Code Program 164 File di dalam package parser.....	178
Code Program 165 Parameter pada tema	178
Code Program 166 Val colorScheme.....	180
Code Program 167 KoreDiaryTheme.kt.....	180
Code Program 168 SettingsScreen.....	189
Code Program 169 EditorNavArgs	190
Code Program 170 Mengirim data lewat FAB	191
Code Program 171 Mengirim data melalui item post	192
Code Program 172 SearchBar.....	194
Code Program 173 Button Clear	195
Code Program 174 SearchScreen	197
Code Program 175 OnBoarding.kt	206
Code Program 176 DetailScreen	209
Code Program 177 Convert bitmap to uri, DetailViewModel	209
Code Program 178 Class Application	211
Code Program 179 Class Application (2)	212
Code Program 180 Class Application (3)	212
Code Program 181 File App	213
Code Program 182 var isOnBoarding pada GlobalPreferences & Global State	213
Code Program 183 BottomBar.kt pada package component pada package presentation	214
Code Program 184 KoreApp.kt.....	216
Code Program 185 MainActivity.....	217
Code Program 186 Uses Permission	218

Code Program 187 Menambahkan name pada application	218
Code Program 188 Menambahkan provider	219
Code Program 189 filepaths.xml.....	220
Code Program 190 Menambahkan tema	224
Code Program 191 Mengganti tema.....	225
Code Program 192 InstallSplashScreen	225
Code Program 193 AndroidManifest.....	227



Bagian 1 :

Android

1.1 Sekilas Tentang Android

Android



Gambar 1 – Logo Android

Sekarang ini, kita tentu mengenal system operasi bernama Android yang dikeluarkan oleh Google. Android dapat kita jumpai di smartphone dan tablet. Pada peranti yang menggunakan Android, Android adalah alat multimedia. Sistemnya dapat digunakan sebagai telefon, pengirim pesan, pemutar musik, dan pemutar video.

Android dipakai oleh banyak produsen smartphone, karena sistem operasi ini dapat dikustomisasi untuk peranti yang mereka ciptakan, yang mana ini menarik di mata Perusahaan teknologi yang membutuhkan barang langsung jadi dengan biaya yang rendah.

1.2 Google Play Store



Gambar 2 – Logo Google Play

Android bersifat open source sehingga menarik komunitas *developer* untuk mengembangkannya. Aplikasi yang dikembangkan oleh *developer* dapat didistribusikan melalui *web*, *file Apk*, dan *store* (Google Play Store). Developer dapat mendistribusikan melalui

Google Play yang merupakan cara termudah ke banyak pengguna.

Google Play Store adalah *store* resmi untuk Android yang dibuat dan dikelola oleh Google. Di sana terdapat banyak aplikasi yang bisa dicari dan diunduh. Google Play tidak hanya menyediakan aplikasi. Terdapat banyak konten lainnya, seperti buku, musik, film, dan program televisi.

1.3 Android Studio



Gambar 3 – Logo Android Studio

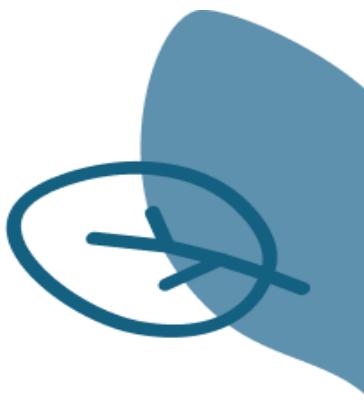
Dalam mengembangkan sebuah program, para *developer* tentu menggunakan IDE (*Integrated Development Enviroment*). Untuk pengembangan Android, Google merilis IDE berbasis IntelliJ IDEA bernama Android Studio.

Android Studio memiliki SDK (*Software Development Kit*), fungsinya untuk membantu dalam mengembangkan aplikasi Android. SDK memiliki tools seperti *software libraries*, *emulator*, *sample code*, *debugger*, dokumentasi, dan tutorial.



Gambar 4 – Logo Kotlin

Pada awalnya, pengembangan aplikasi Android lebih sering menggunakan bahasa pemrograman Java. Android Studio juga mendukung penggunaan bahasa pemrograman seperti C++ dan Go. Google juga merilis bahasa pemrograman turunan Java bernama Kotlin, dan pada IO 2017, Kotlin ditetapkan sebagai tambahan bahasa pemrograman aplikasi Android.



Bagian 2 :

UI dan UX

2.1 Apa itu UI dan UX ?

Apa kalian mengenal UI dan UX ?. *User Interface (UI)* adalah desain elemen visual dan interaktif, seperti web dan aplikasi. Secara singkatnya adalah tampilan yang akan dilihat oleh pengguna. Yang biasanya termasuk ke dalam UI itu seperti tata letak layar, tombol, ikon, dan tipografi. Sebagai Developer (pengembang), kita harus membuat UI yang mudah digunakan, mudah dipahami, dan nyaman secara visual bagi pengguna.



Gambar 5 - Android vs Ios UI

User Experience (UX) adalah pengalaman yang dirasakan pengguna Ketika menggunakan sebuah aplikasi. UX biasanya terkait dengan kemudahan pengguna, kinerja, dan kegunaan.

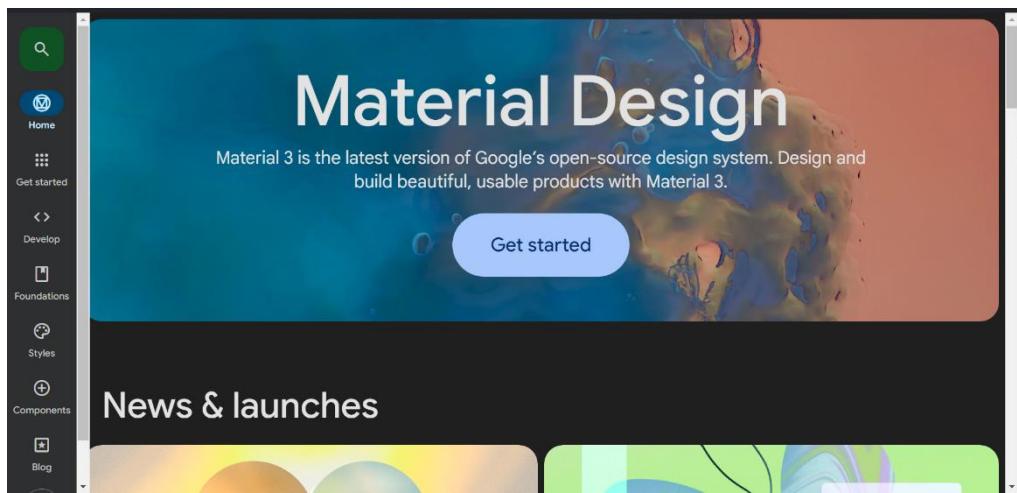
UI dan UX sangat berkaitan. UI yang baik berpengaruh terhadap UX yang baik, tetapi UX yang baik dapat dicapai dengan UI yang tidak terlalu sempurna. Misalnya sebuah aplikasi yang memiliki desain UI yang sederhana dan kurang menarik, tetapi memiliki navigasi dan fitur yang baik sehingga memberikan UX yang baik terhadap pengguna.

2.2 Standar UI / UX

Setelah mengetahui UI / UX, mungkin kita bertanya, “Lalu, bagaimana cara membuat UI / UX yang baik ?”.

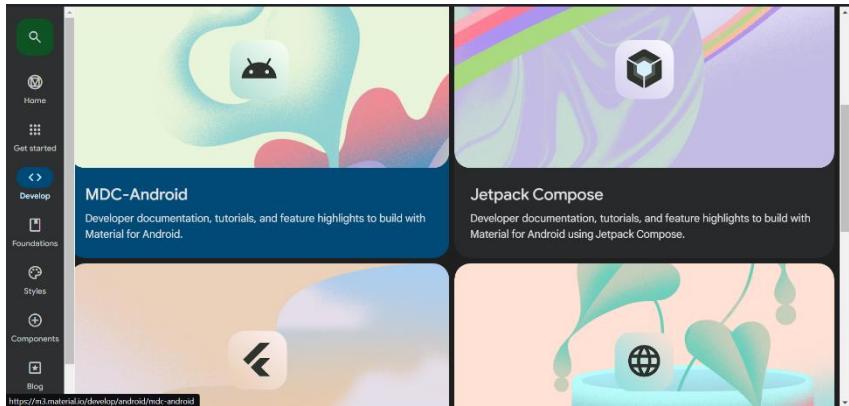
Google telah menyediakan sebuah “material” yang bisa kita gunakan sebagai contoh UI / UX yang akan kita buat. Google juga menyertakan standar ukuran yang baik untuk sebuah App Bar misalnya.

Kita bisa mengunjungi situs Material Design untuk melihat standar ukuran yang ditetapkan oleh Google.



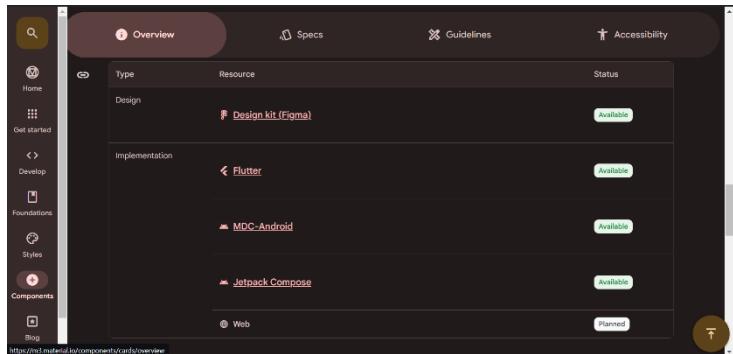
Gambar 6 <https://m3.material.io/>

Material Design menyediakan dokumentasi untuk membantu pengembangan pada Android dan juga Web. Dokumentasi ini bisa di implementasikan pada beberapa framework yang didukung.



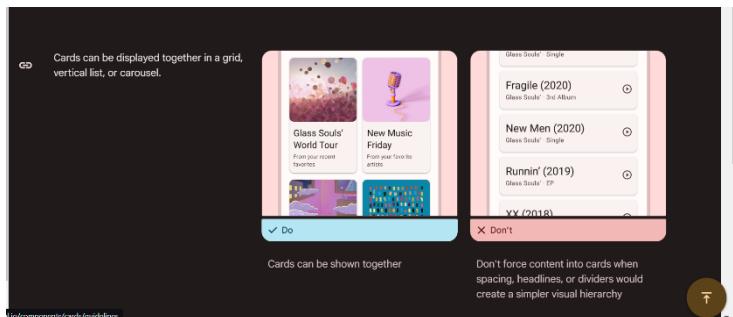
Gambar 7 Framework yang didukung

Kita dapat mengetahui bagaimana komponen-komponen yang terdapat pada Android misalnya, bisa kita implementasikan pada kode pemrograman.



Gambar 8 Implementasi komponen

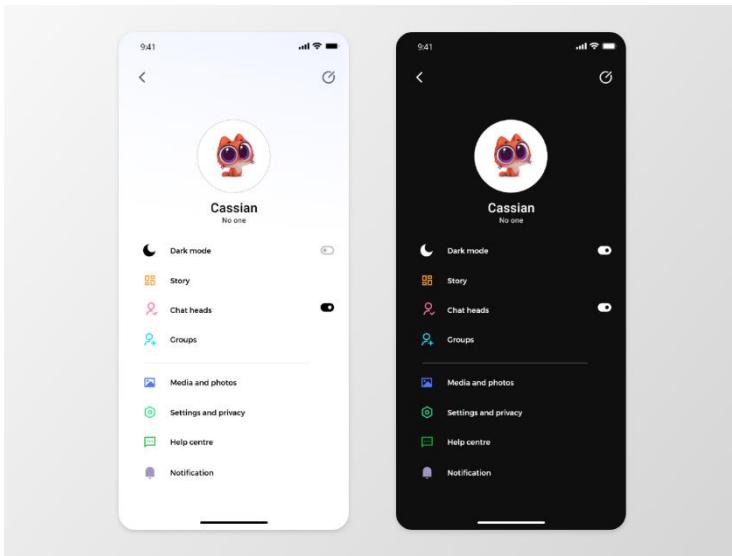
Material Design juga memberi kita penjelasan mengenai *fungsi* dari setiap komponen dan bagaimana penggunaannya.



Gambar 9 Cara penggunaan yang baik pada komponen

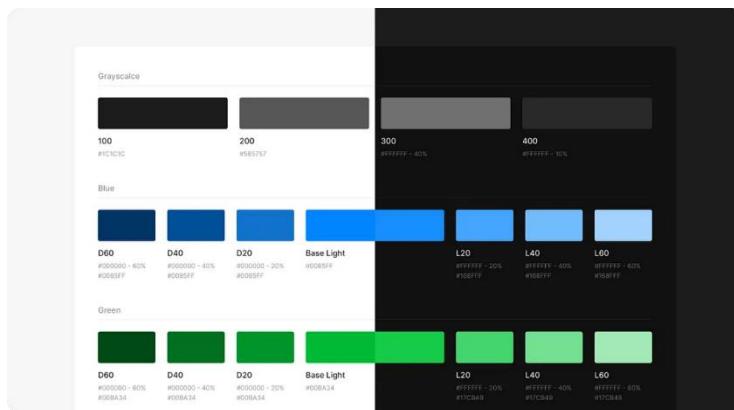
2.3 Tema Aplikasi

Tema merupakan unsur yang penting pada UI / UX. Pemilihan warna tema yang tepat sangat memengaruhi kenyamanan pengguna.



Gambar 10 Tema Terang vs Tema Gelap

Warna yang digunakan harus memiliki color palette atau kita anggap seperti susunan warna yang sesuai. Pada tema yang terang kita menggunakan warna-warna yang gelap, sebaliknya pada tema yang gelap kita menggunakan warna-warna yang lebih terang.



Gambar 11 Color palette tema terang dan gelap



Bagian 3 :

Design

3.1 Framework

Setelah kita mengetahui apa itu UI dan UX, sekarang kita akan memulai mendesain tampilan untuk pengguna. Ada beberapa tahapan ketika kita mendesain tampilan untuk pengguna. Pertama kita perlu membuat framework atau kita bisa menyebutnya kerangka desain.

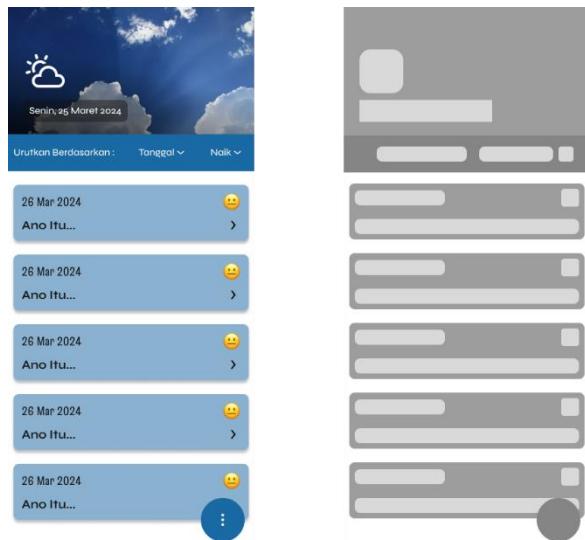


Gambar 12 Framework aplikasi saat ini

Seperti yang bisa kita lihat di atas, *framework* itu hanya kerangkanya saja (💀). Tidak terdapat desain yang utuh dan pewarnaan. Dari kerangka desain inilah baru kemudian kita membuat desain utuhnya.

3.2 Desain

Setelah kita membuat kerangka desain, sekarang kita akan membuat desain versi jadinya.



Gambar 13 Desain jadi vs kerangka

Kalau kalian dari tadi bertanya-tanya saya desain kerangka dan desain jadinya menggunakan apa ?. Saya sarankan menggunakan software figma.



Gambar 14 Figma Logo

Kalian dapat menelusuri pada browser dan menginstallnya. Software ini menyediakan versi gratis nya kok.



Gambar 15 Preview Figma

Figma bisa dioperasikan pada semua platform – Windows, Mac, dan Linux. Bukan Nintendo, Playstation, dan Xbox! -

Setelah menyelesaikan desain, mari kita beranjak ke tahap selanjutnya.

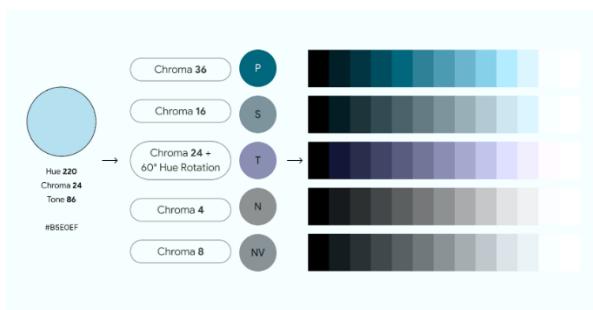


Gambar 16 Desain yang sudah jadi

3.3 Color Palette

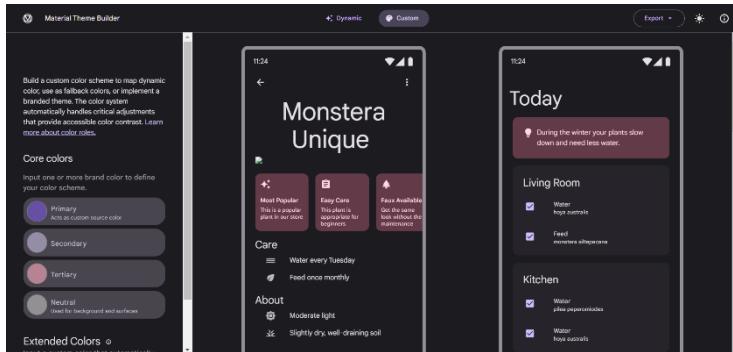
Sebagaimana yang telah kita pelajari pada bab sebelumnya, penggunaan warna sangat memengaruhi pada UI dan UX. Hal itu juga berdampak pada pengguna.

Jadi, pastikan kalian telah memiliki palet warna kalian sendiri.



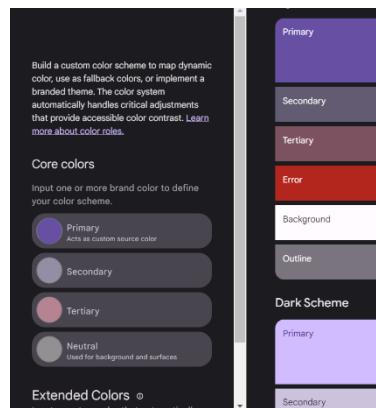
Gambar 17 Contoh color palette

Setelah kalian memiliki *color palette* atau *color schema* kalian sendiri, sekarang kita pergi ke situs material design 3. Ketik pada tab search browser kalian "<https://m3.material.io/theme-builder#/custom>".



Gambar 18 Theme Builder

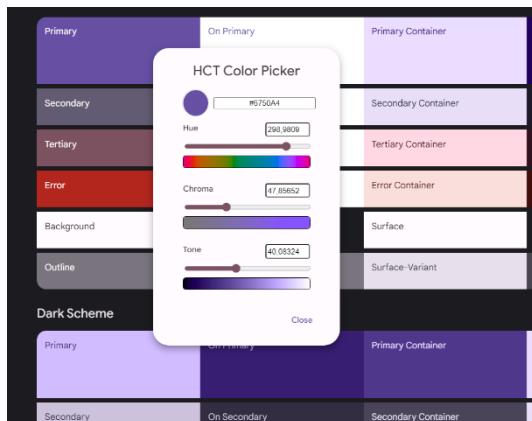
Klik pada “Primary” pada side view sebelah kiri.



Gambar 19 Primary

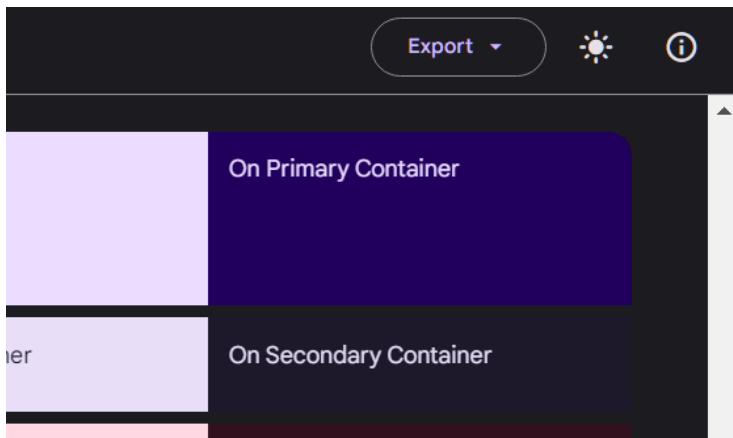
Akan muncul dialog berisi warna, *Hue*, *Chroma*, *Tone*. Di sini format warna yang digunakan adalah kode hex yang diawali '#' dan biasanya diikuti angka dan huruf.

Jadi untuk membuat seluruh palet warna yang akan dibutuhkan pada sebuah aplikasi, kita hanya perlu kode heksa dari warna yang kita punya dan mengisi kolom warna pada dialog primary tersebut.



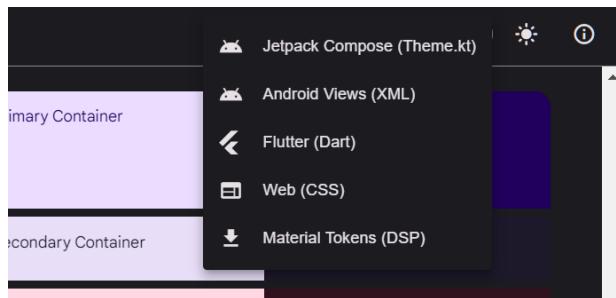
Gambar 20 Dialog Primary

Setelah palet warnanya dibuat secara otomatis, klik "export" pada kanan atas.



Gambar 21 Tombol Export

Setelah itu, akan muncul berbagai macam opsi material seperti Css (Web), Material Token, XML (Android), Flutter (Mobile), dan Compose (Android). Pilih yang Compose karena kita akan membuat aplikasi menggunakan Compose.



Gambar 22 Export Menu

Sesudah kita memilih compose, file warna-warna yang akan digunakan pada element android sudah siap digunakan.

Mari kita mulai membuat aplikasi, persiapkan diri kalian. Kita akan melangkah ke bab berikutnya.



Bagian 4 :

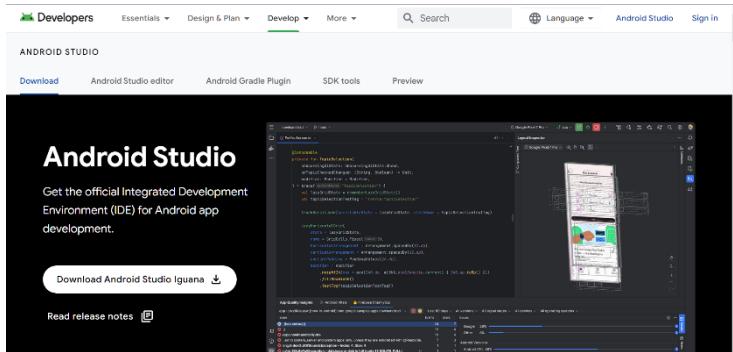
Membuat Aplikasi

(Auth)

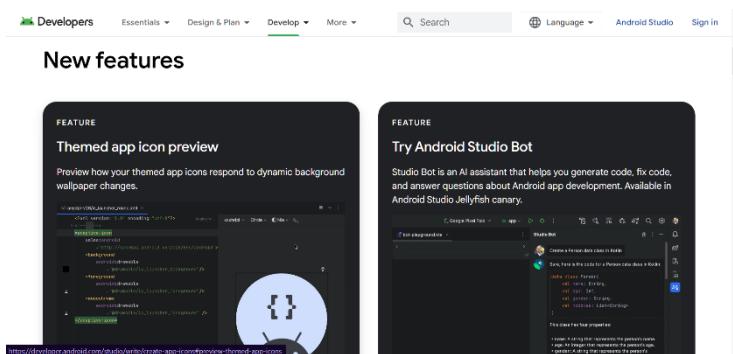
4.1 Memulai Project

Pada aplikasi itu ada yang namanya Main Thread dan Background Thread. Main Thread adalah lapisan dimana UI di-*rendering* dan Background Thread adalah lapisan dimana Logika pemrograman bekerja. Secara ringkasnya kalian anggap saja seperti ini.

Tapi sebelum kita memulai, kalian sudah memiliki Android Studio ?. Apa ? Belum ?!. Kalian bisa mengunjungi situs resmi dari google yaitu “developer.android.com/studio”. Di sana kalian juga bisa melihat apa saja sih fitur atau tools yang android studio punya atau miliki.



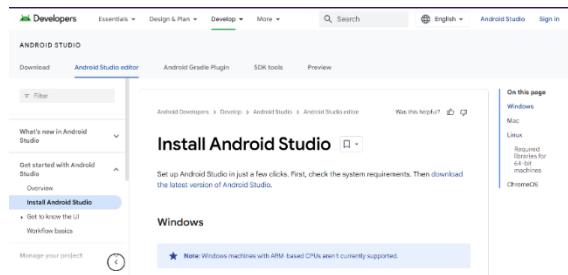
Gambar 23 Download Android Studio



Gambar 24 Fitur Android Studio

Lalu, cara install nya ?. Sepertinya tidak perlu saya terangkan di sini, karena pada website tersebut terdapat tutorial atau cara untuk menginstallnya. Jika tidak

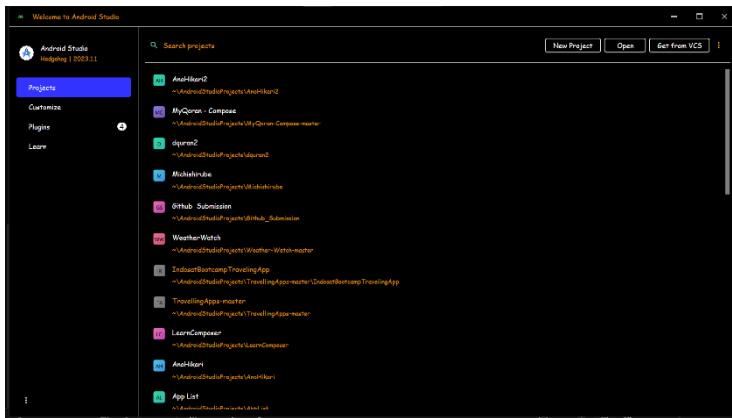
menemukannya kalian bisa kunjungi “developer.android.com/studio/install”.



Gambar 25 Install Android Studio

Setelah itu kita bisa melanjutkan untuk memulai aplikasi.

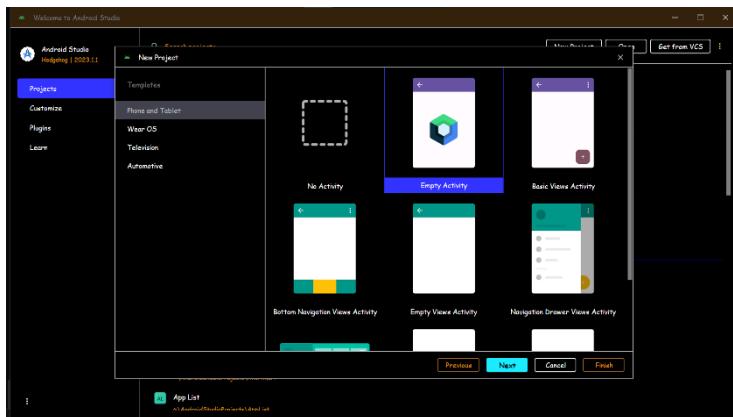
Setelah selesai menginstall dan muncul *splash screen*, kalian akan tiba pada layar selamat datang.



Code Program 1 Welcome Screen

Mungkin pada layar kalian, belum ada banyak project seperti diatas dan hanya ada layar kosong. Klik tombol “New Project”.

Akan muncul dialog berisi pilihan *template* yang dapat kita pilih. Klik pada *template* yang terdapat logo Jetpack Compose dan Bernama “Empty Activity”. Lalu klik “Next”.

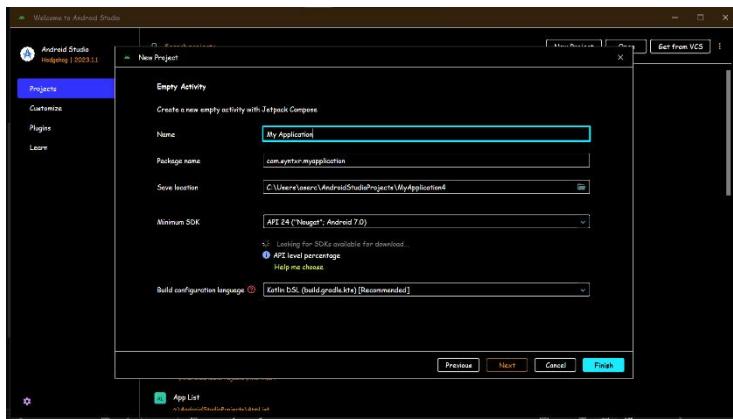


Code Program 2 Template menu

Setelah itu kita akan dihadapkan dengan dialog yang berisi *Name*, *Package Name*, *save location*, dan *minimum sdk*. Kita harus mengubah *Package Name*. Di layar kalian pasti masih menggunakan “com.google.myapplication”. Silahkan kalian ganti “google” dengan nama *developer* atau nama buatan kalian.

Kenapa “google” harus diganti ?. Ketika kita nanti merilis aplikasi, akan ada yang namanya persiapan rilis. Hal ini akan menyebabkan *error* karena google telah membatasi penamaan package “google”.

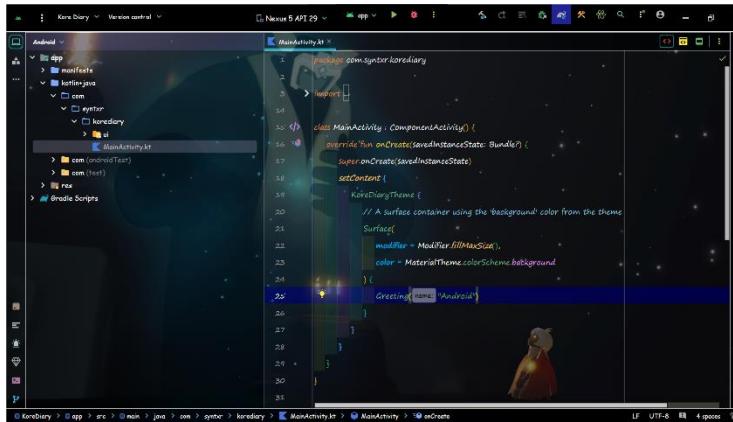
Kalian juga mengganti kolom “Name” dengan nama aplikasi kalian.



Code Program 3 Ganti name dan package

Klik “Finish” setelah selesai mengganti nama.

Kita akan dihadapkan dengan file Main Activity. Kalian melihat “Codingan” kalian merah ?. Tenang, Android Studio sedang men-sync codingan kalian dengan gradle.

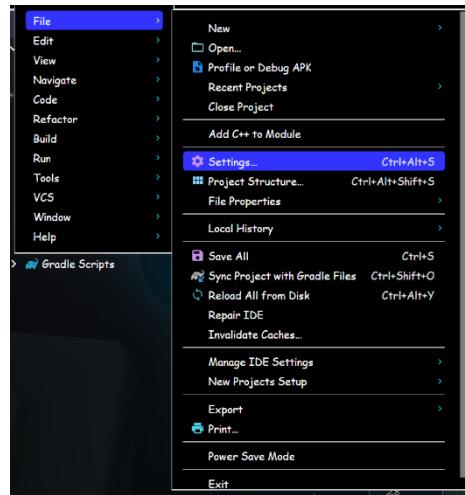


The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the file tree showing the project structure under 'Android'. The main editor window displays the code for 'MainActivity.kt'.

```
1 package com.zynter.koredairy
2
3 import android.os.Bundle
4 import androidx.appcompat.app.AppCompatActivity
5 import androidx.compose.foundation.layout.fillMaxSize
6 import androidx.compose.material.MaterialTheme
7 import androidx.compose.runtime.Composable
8 import androidx.compose.ui.Modifier
9 import androidx.compose.ui.graphics.Color
10 import androidx.compose.ui.unit.dp
11 import androidx.compose.ui.window.Window
12 import androidx.compose.ui.window.application
13 import androidx.compose.ui.window.rememberWindowState
14
15 class MainActivity : AppCompatActivity() {
16     override fun onCreate(savedInstanceState: Bundle?) {
17         super.onCreate(savedInstanceState)
18         setContent {
19             KoredairyTheme {
20                 // A surface container using the 'background' color from the theme
21                 Surface(
22                     modifier = Modifier.fillMaxSize(),
23                     color = MaterialTheme.colorsScheme.surface
24                 ) {
25                     Greeting(name = "Android")
26                 }
27             }
28         }
29     }
30 }
31
32 @Composable
33 fun Greeting(name: String) {
34     Text(text = "Hello $name")
35 }
```

Code Program 4 Main Activity

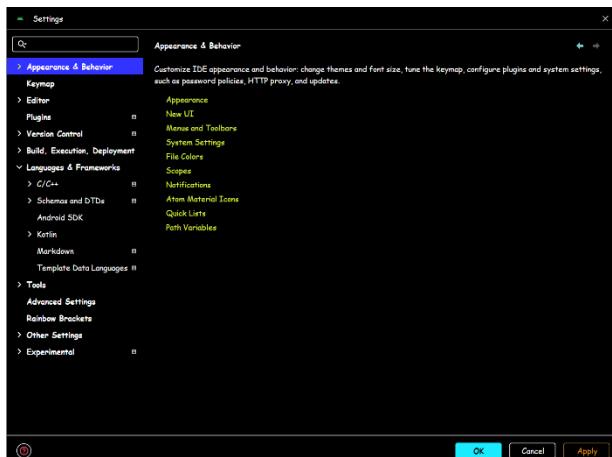
Jika codingan kalian tetap merah, klik saja logo bergambar gajah atau gunakan shortcut “Ctrl + Shift + O”.



Code Program 5 File menu

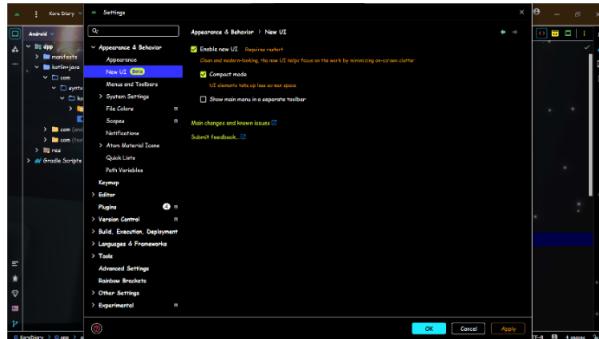
Tampilan kalian berbeda dengan punya saya ?. Kalau kalian menggunakan Android Studio versi Giraffe ke atas, kalian bisa menggunakan GUI (*Graphic User Interface*) yang baru.

Klik pada File > Settings. – ini maksudnya bukan kurang dari ya! – Akan muncul dialog settings yang berisi banyak pengaturan.



Code Program 6 Menu settings

Seperti yang kalian lihat di atas, klik pada Appereance & Behaviour > New UI. Setelah itu klik “Enable New UI”. Ini membutuhkan *restart* (membuka ulang software) untuk menyimpan perubahan. Klik Apply dan Ok.



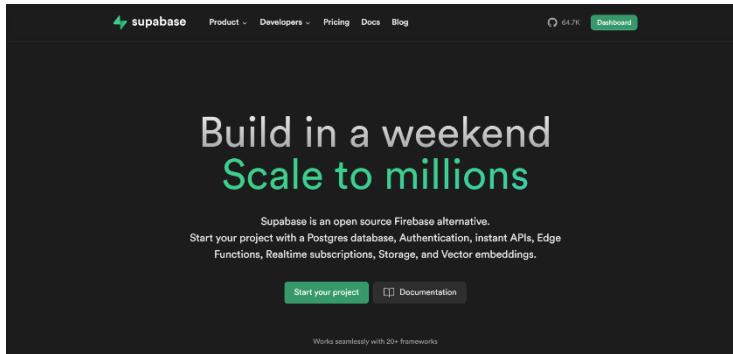
Code Program 7 New UI enable

Kembali lagi ke project. Kita akan membuat *data* dan alurnya terlebih dahulu baru kemudian kita membuat UI atau *screen*-nya. Nah, disini kita akan mencoba untuk membuat sebuah aplikasi diary yang memiliki autentikasi pengguna dan bisa menyimpan *data* secara online.

Lalu, dengan apa kita dapat membuat sebuah penyimpanan online untuk pengguna ?. Kita akan menggunakan *database* untuk membuatnya.

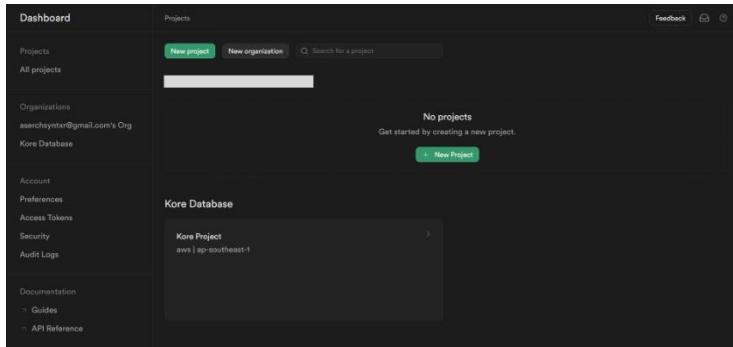
4.2 Menyiapkan Database

Agar mempermudah, kita akan menggunakan *database* dari supabase. Supabase menyediakan API agar *data* bisa diolah, baik itu di Input, Delete, ataupun Read.



Gambar 26 Supabase

Klik “Dashboard” jika kalian telah memiliki akun. Jika kalian belum memiliki akun ya bikin dulu (Pakai Nanya). Nantinya kita akan tiba pada tampilan *Dashboard Project*.



Gambar 27 Dahsboard Project

Kalau kalian belum memiliki project, kalian bisa klik pada “New Project” yang ada di tengah. Kalau sudah kalian akan tiba pada halaman project yang kalian buat.

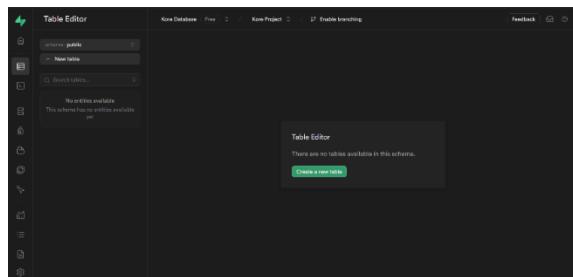
Pada bilah sisi sebelah kiri terdapat banyak daftar.

The screenshot shows the Kore Project interface. On the left, there's a sidebar with various icons for managing projects and databases. The main area is titled "Kore Project" and displays a message: "Welcome to your new project. Your project has been deployed on its own instance, with its own API all set up and ready to use." Below this, there's a section titled "Get started by building out your database" with a brief introduction to the database editor. At the bottom of the main area, there are three buttons: "Table Editor" (which is selected), "SQL editor", and "About Database". To the right of the main area, a table editor window is open, showing a table with columns "id", "name", and "iso2". The data in the table is as follows:

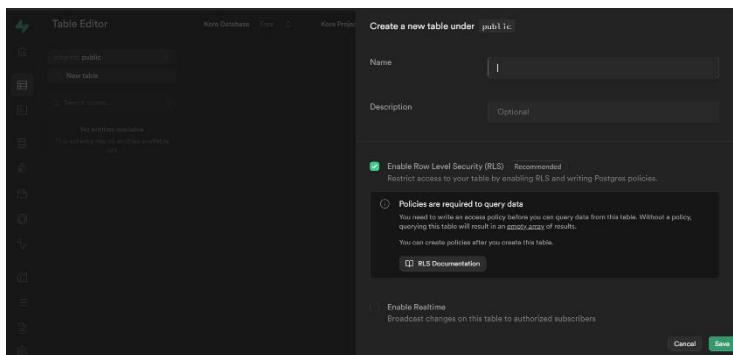
	id	name	iso2
1	Angola	AO	
2	Timor-Leste	TL	
3	Serbia	RS	
4	Bahamas	BS	
5	Chile	CL	
6	Denmark	DK	
7	Timor-Leste	TL	
8	Angola	AO	

Gambar 28 Project Supabase

Kita tekan pada *Table Editor*. Kemudian kita tekan pada tombol “Create a new table”. Tampilan untuk membuat tabel baru akan muncul.



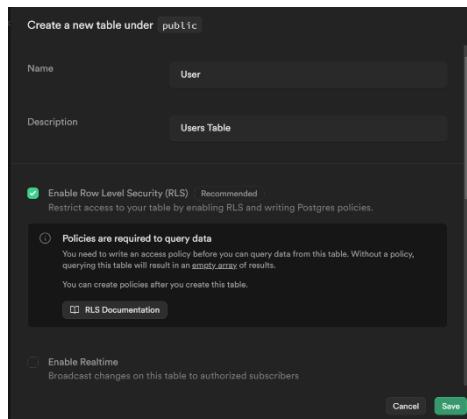
Gambar 29 Beranda Tabel



Gambar 30 Editor Table

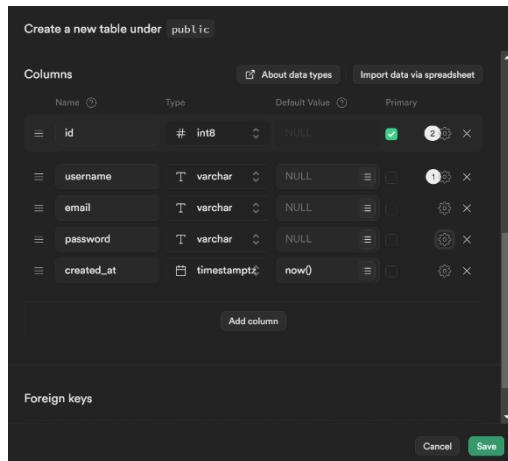
Karena kita membuat sebuah aplikasi yang membutuhkan pengguna, maka kita akan membuat tabel untuk pengguna terlebih dahulu.

Pada kolom nama, kita akan memberi tabel ini dengan nama “User”. Kita akan mendapati dua kolom yang sudah tersedia yaitu “id” dan “created_at”, dua kolom ini kita biarkan. Kita akan membuat kolom “*username*” yang memiliki tipe *data* “varchar”, begitu juga dengan kolom “*email*” dan “*password*”. Setelah itu kita tekan “save”.

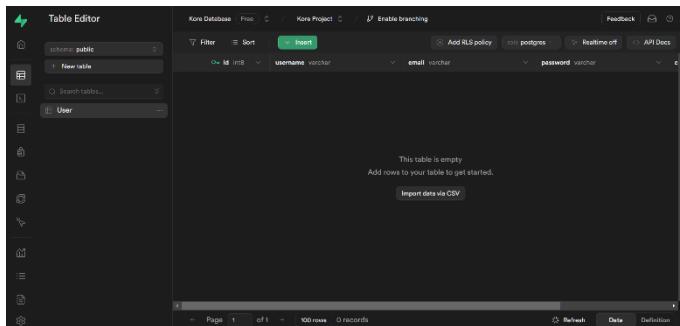


Gambar 31 Table Name

Setelah tabel yang kita buat berhasil disimpan, maka pada table editor akan muncul nama tabelnya.



Gambar 32 Table Column



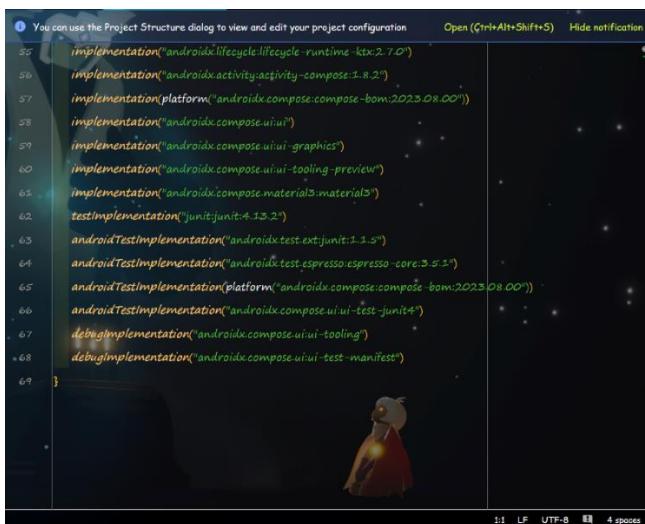
Gambar 33 Table User

4.3 Data

Sekarang kita akan kembali ke android studio. Setelah kita selesai membuat tabel, kita akan membuat client pada kotlin agar *database* tadi bisa digunakan.

Supabase menyediakan client untuk Kotlin. Menariknya project ini bukan official milik supabase, melainkan hasil dari komunitas supabase.

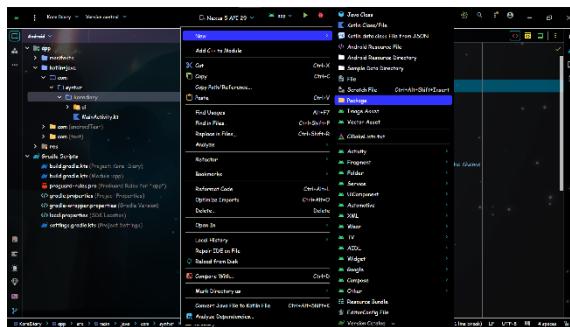
Lalu, bagaimana cara menggunakannya ?. Pertama, kita membuka file build.gradle.kts (yang “Module : App”). Lalu kita tambahkan .



```
① You can use the Project Structure dialog to view and edit your project configuration Open (Ctrl+Alt+Shift+S) Hide notification
55 implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
56 implementation("androidx.activity:activity-compose:1.8.2")
57 implementation(platform("androidx.compose:compose-bom:2023.08.00"))
58 implementation("androidx.compose.ui:ui")
59 implementation("androidx.compose.ui:ui-graphics")
60 implementation("androidx.compose.ui:ui-tooling-preview")
61 implementation("androidx.compose.material:material")
62 testImplementation("junit:junit:4.13.2")
63 androidTestImplementation("androidx.test.ext:junit:1.1.5")
64 androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
65 androidTestImplementation(platform("androidx.compose:compose-bom:2023.08.00"))
66 androidTestImplementation("androidx.compose.ui:ui-test-junit4")
67 debugImplementation("androidx.compose.ui:ui-tooling")
68 debugImplementation("androidx.compose.ui:ui-test-manifest")
69 }
```

Code Program 8 Gradle

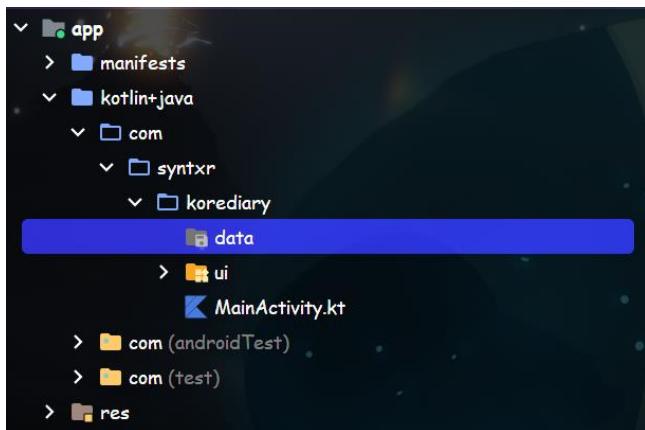
Setelah itu kita perlu membuat package "data" agar file-file program yang berisi *data* tidak tercampur dengan UI dan logic. Hal ini akan memudahkan kita nantinya.



Code Program 9 Menambah Package

```
setContent {  
    . . .  
    KoreDiaryTheme {  
        New Package  
        com.syntx.korediary.data  
        modifier = Modifier.fillMaxSize(),  
        color = MaterialTheme.colorScheme.ba  
    }  
}
```

Code Program 10 Menamai package

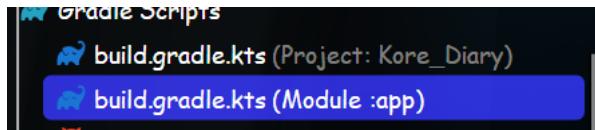


Code Program 11 Package data

Kita mau bikin apa sih ? Kita akan membuat sebuah class atau lebih tepatnya *data* class. Data class ini merupakan sebuah class yang berisi variabel. Baik itu val yang konstan atau tidak berubah nilainya, atau var yang bisa berubah nilainya.

Kita akan menggunakan *dependency* atau *library* dari kotlin, berupa *serialization* dimana kita bisa menerima *data* tanpa harus menggunakan JSON.

Maka dari itu kita harus menambahkannya terlebih dahulu di gradle module. Kita juga akan menambahkan *library* lain yang akan kita butuhkan nanti, sehingga kita tidak perlu berulang-ulang kembali ke file gradle.



Code Program 12 Gradle Module

```
// gson, icon, splash screen, fonts & api wrapper
implementation("com.google.code.gson:gson:2.10.1")
implementation("androidx.compose.material:material-icons-extended-android:1.6.6")
implementation("androidx.compose.material:material-icons-core-android:1.6.6")
implementation("androidx.core:core-splashscreen:1.0.1")
implementation("androidx.compose.ui:ui-text-google-fonts:1.0.6")
implementation("com.github.rmaprojects:apiresponsewrapper:1.7")

// worker
implementation("androidx.work:work-runtime-ktx:2.9.0")
implementation("androidx.hilt:hilt-work:1.2.0")

// coil
implementation("io.coil-kt:coil-compose:2.6.0")

// rich txt
implementation("com.canopas.editor:rich-editor-compose:0.1.0")
```

```
// emoji selector  
implementation("com.github.mendelordanza:compose-emoji-picker:0.0.2")  
  
// multi fab  
implementation("com.github.bumptech:MultiFab:1.0.8")  
  
// supabase  
implementation(platform("io.github.jan-tennert.supabase:bom:2.3.1"))  
implementation("io.github.jan-tennert.supabase:gotrue-kt")  
implementation("io.github.jan-tennert.supabase:postrest-kt")  
implementation("io.github.jan-tennert.supabase:realtime-kt")  
  
// kotlin ext and coroutine support for room  
implementation("androidx.room:room-runtime:2.6.1")  
implementation("androidx.room:room-ktx:2.6.1")  
ksp("androidx.room:room-compiler:2.6.1")
```

```
// ktor  
implementation("io.ktor:ktor-client-okhttp:2.3.10")  
  
// kotpref  
implementation("com.chibatching:kotpref:kotpref:2.13.2")  
implementation("com.chibatching:kotpref-initializer:2.13.2")  
implementation("com.chibatching:kotpref-enum-support:2.13.2")  
  
// dagger hilt  
implementation("com.google.dagger:hilt-android:2.51.1")  
implementation("androidx.hilt:hilt-navigation-compose:1.2.0")  
ksp("com.google.dagger:hilt-android-compiler:2.51.1")  
ksp("com.google.dagger:hilt-compiler:2.51.1")  
ksp("androidx.hilt:hilt-compiler:1.2.0")  
  
// Destination  
implementation("io.github.raamcosta.compose-destinations:animations-core:1.9.54")  
ksp("io.github.raamcosta.compose-destinations:ksp:1.9.54")
```

```
// notification  
implementation("io.kurniawan:notify:1.4.0")
```

Code Program 13 Menambah depedency

Setelah kita menambah library kemudian klik “Sync Now” agar gradle kita di-build ulang. Pastikan kalian memiliki koneksi internet ya!.

Setelah itu kita tambahkan file pada package *data*. Kita akan membuat *data* class yang dilengkapi serialization. Sebelum itu, kita akan membuat package agar mudah dibaca nantinya.



Code Program 14 Package source di dalam



Code Program 15 Package remote di dalam source



Code Program 16 Package serializable di dalam remote

Kita tambahkan *data class* PostDto pada package serializable.

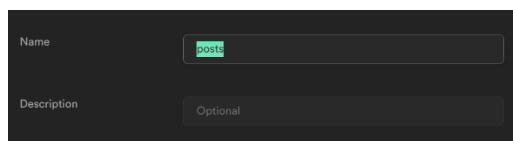
```
@Serializable  
data class PostDto(  
    @SerializedName("id") val id: Int? = null,  
    @SerializedName("uuid") val uuid: String,  
    @SerializedName("user_id") val userId: String,  
    @SerializedName("title") val title: String,  
    @SerializedName("value") val value: String,  
    @SerializedName("mood") val mood: String,  
    @SerializedName("created_at") val createdAt: String  
)
```

Code Program 17 PostDto

Kita juga tambahkan *data class* serializable untuk User. Setelah itu kita akan kembali ke supabase untuk membuat table Post, kita juga akan memperbarui table User.

```
@Serializable  
data class User(  
    @SerializedName("id") val id : Int,  
    @SerializedName("uuid") val uuid : String,  
    @SerializedName("username") val username : String,  
    @SerializedName("created_at") val createdAt : String  
)
```

Code Program 18 User



id	# int8	NULL	✓	2	x
created_at	timestamptz	now()	☰	⚙️	x
user_id	uuid	NULL	☰	⚙️	x
value	text	NULL	☰	⚙️	x
mood	varchar	NULL	☰	⚙️	x
title	text	NULL	☰	⚙️	x
uuid	text	NULL	☰	⚙️	x
published	bool	false	☰	⚙️	x

Code Program 19 Table post supabase

Untuk table *user* kita akan menghapus *email* dan *password*. Kita akan menggunakan auth sehingga yang kita butuhkan pada table adalah UUID dan Username.

The screenshot shows the Supabase Studio interface for managing a table named 'user'. The table structure is defined with the following columns:

- created_at**: timestamp, with a constraint `now()`.
- username**: text, with a constraint `"::text"`.
- uuid**: uuid, with a constraint `NULL`.

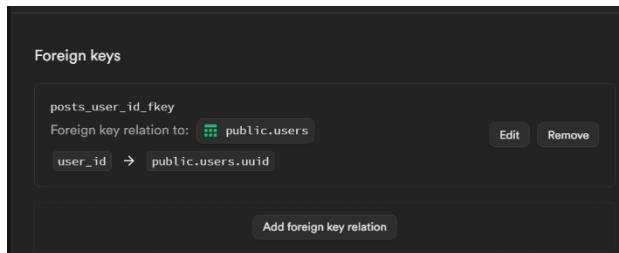
Below the table, the 'Foreign keys' section displays a single entry:

- public_users_uuid_fkey**: A foreign key relation to the 'auth.users' table.
- Details: `Foreign key relation to: auth.users`, `uuid → auth.users.id`.
- Action buttons: `Edit` and `Remove`.

A button labeled 'Add foreign key relation' is also visible.

Code Program 20 Table user supabase

Tambahkan relasi dari table *post* ke table *user*. Table *post* akan butuh *user id* agar *data* yang didapatkan oleh *user* tidak tercampur dengan *user* lainnya.



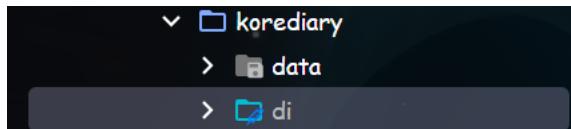
Code Program 21 Foreign key post ke user

4.4 Auth

Apa yang akan pertama kali dilakukan pengguna ketika pertama kali menjalankan aplikasi ?. Benar, pengguna akan *login* terlebih dahulu. Lalu ketika pengguna ternyata belum memiliki akun, maka pengguna akan mendaftar atau melakukan *register* terlebih dahulu.

Login atau *register* merupakan proses autentikasi. Supabase menyediakan autentikasi dengan berbagai macam *provider*. Mulai dari Email, Google, Twitter, Facebook dan bahkan *login* menggunakan akun Spotify 😱. Tapi, kita hanya akan menggunakan *login* dan *register* dengan *email* saja.

Kita akan mulai membuat logika untuk *login* dan *register*. Sebelum itu, kita akan membuat *dependency injection*. Buat apa tuh ? Ini akan memudahkan kita karena kita menggunakan hilt-dagger.



Code Program 22 Paclage di

Setelah membuat package baru bernama “di”, kita akan membuat Object bernama “App Module” di dalam package “di”.



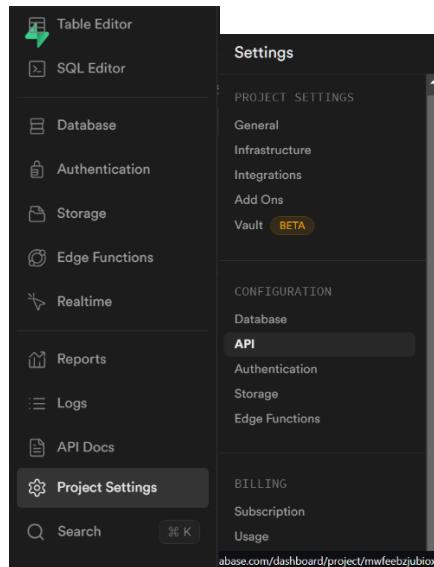
Code Program 23 Membuat object app module

Setelah itu kita akan membuat fun provideSupabaseClient. Kenapa kita membuat ini?. Fun ini diperlukan agar kita bisa menggunakan fitur-fitur supabase di kotlin seperti Auth, Storage, Postgrest, dan Realtime.

```
object AppModule {  
  
    @Provides  
    @Singleton  
  
    fun provideSupaClient(): SupabaseClient = createSupabaseClient(  
        supabaseUrl = BuildConfig.SUPABASE_URL,  
        supabaseKey = BuildConfig.SUPABASE_API_KEY  
  
    )  
        this: SupabaseClientBuilder  
        install(Auth) // agar bisa menggunakan auth dari supabase  
        install(Postrest) // agar bisa menggunakan table dari supabase  
        install(Realtime) // agar bisa menggunakan realtime dari supabase  
  
    }  
}
```

Code Program 24 Provide supabase client

Saya di sini menyembunyikan supabase url dan key nya. Bagaimana cara kita mendapatkan supabase url dan key?. Kita bisa mendapatkannya di *project settings* pada laman supabase.

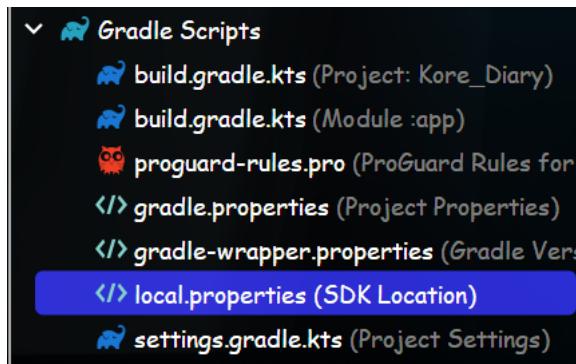


A screenshot of the Supabase API Settings page. The top header says "API Settings". The first section is "Project URL" with a "URL" input field containing a redacted URL, a "Copy" button, and a descriptive text: "A RESTful endpoint for querying and managing your database". The second section is "Project API keys" with a note: "Your API is secured behind an API gateway which requires an API Key for every request. You can use the keys below in the Supabase client libraries." It includes a "Client Docs" button and two key options: "anon" and "public". The "public" key is shown with a redacted key value and a "Copy" button, with a note: "This key is safe to use in a browser if you have enabled Row Level Security for your tables and configured policies."

Code Program 25 Mendapatkan supabase url dan key

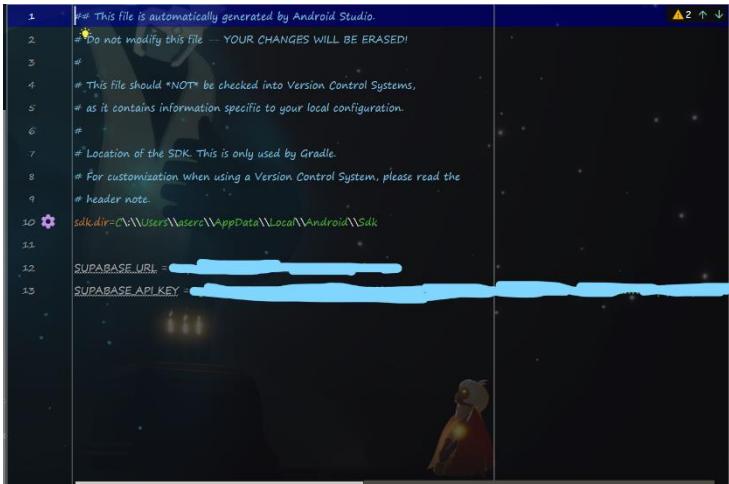
Sebagaimana yang kalian lihat. Di sana tertera supabase url dan keynya. Kenapa saya sembunyikan? Karena url dan key itu satu-satunya cara untuk mengakses project yang kita buat di supabase. Kalian bisa meng-copy url dan key tersebut dan mem-pastenya ke *fun provideSupabaseClient()* tadi.

Atau, jika kalian juga ingin menyembunyikannya seperti saya. Kalian pergi ke file Local.Properties.



Code Program 26 Local properties

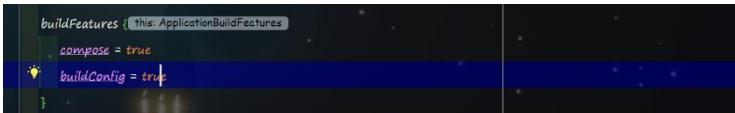
Lalu kalian buat variabel di sana. Seperti gambar di bawah ini.



```
1 // This file is automatically generated by Android Studio.
2 // Do not modify this file -- YOUR CHANGES WILL BE ERASED!
3 //
4 // This file should *NOT* be checked into Version Control Systems,
5 // as it contains information specific to your local configuration.
6 //
7 // Location of the SDK. This is only used by Gradle.
8 // For customization when using a Version Control System, please read the
9 // header note.
10 sdk.dir=C:\\Users\\aserc\\AppData\\Local\\Android\\Sdk
11
12 SUPABASE_URL = [REDACTED]
13 SUPABASE_API_KEY = [REDACTED]
```

Code Program 27 Variabel supabase url dan key

Setelah itu kalian pergi ke Gradle module. Lalu kita aktifkan build config agar kita bisa menggunakan variabel ini nantinya.



```
buildFeatures {
    compose = true
    buildConfig = true
}
```

Code Program 28 Build config

Selanjutnya kita deklarasikan agar variabel yang telah kita buat bisa digunakan.

```
val properties : Properties = readProperties(project.rootProject.file("local.properties"))

buildConfigField(type: "String", name: "SUPABASE_URL", value: "\\\"${properties["SUPABASE_URL"]}\\\"")
buildConfigField(type: "String", name: "SUPABASE_API_KEY", value: "\\\"${properties["SUPABASE_API_KEY"]}\\\"")
```

Code Program 29 Mendeklarasikan variabel build config

Setelah itu kalian klik “sync now” atau kalian klik tombol palu di toolbar atas. Class Build Gradle akan otomatis dibuat. Jika kalian mendapati *error* setelah itu, biasanya class yang mengalami *error* akan ditampilkan. Pada class `BuildConfig.java` kalian tambahkan petik dua pada tiap ujung *value* dari variabel yang telah kalian buat. Sehingga menjadi seperti ini “`SUPABASE_URL`” & “`SUPABASE_KEY`”.



```
public static final String SUPABASE_API_KEY = "";
// Field from default config.
@Usage
public static final String SUPABASE_URL = "";
}
```

Code Program 30 Menambahkan “” pada value

Setelah itu kalian klik icon palu pada toolbar atas. Gradle akan kembali membuat class `BuildConfig.java`.



Code Program 31 Icon “palu” gradle

Setelah itu kalian panggil variabel yang telah dibuat pada *fun* provideSupabaseClient().

```
supabaseUrl = BuildConfig.SUPABASE_URL, // supabase url di sini  
supabaseKey = BuildConfig.SUPABASE_API_KEY // supabase key disini
```

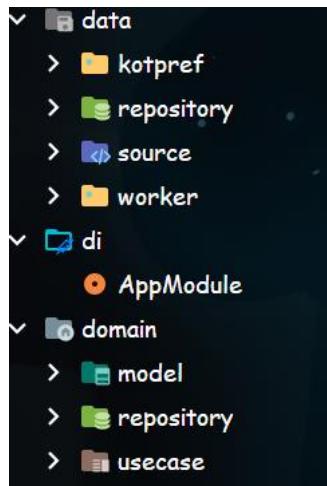
Code Program 32 Memanggil variabel build config

Pastikan kalian meng-*import* BuildConfig yang benar!. Import nya yang berisi com.namadeveloper.namaaplikasi dan baru BuildConfig.

```
import com.syntaxr.korediary.BuildConfig
```

Code Program 33 Import build config

Setelah itu kita akan membuat *repository*. Kita akan membuat package baru bernama “domain” dengan package “repository” di dalamnya. Kita juga membuat package “repository” di dalam package *data*.



Code Program 34 Package repository

Kalian abaikan saja dulu package yang lain, kita akan membuatnya bertahap. Pada domain, *repository* kalian isi dengan Class Interface UserRepository. Sedangkan pada *data*, *repository* kalian isi dengan Class UserRepoImpl.

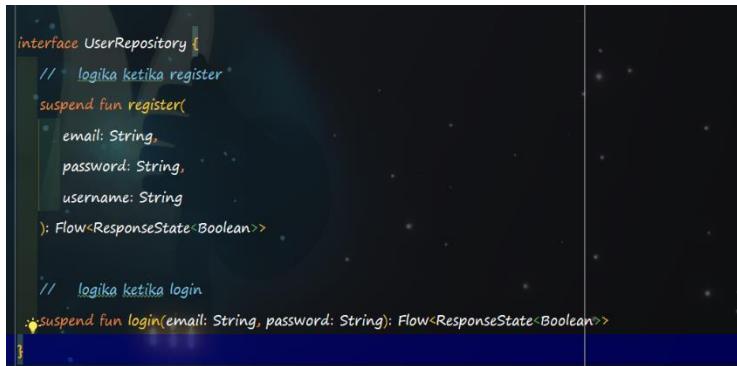
UserRepository

Code Program 35 Class interface user repository pada domain

UserRepoImpl

Code Program 36 Class UserRepoImpl pada data

Setelah dibuat, kita akan mengerjakan Interfacenya terlebih dahulu.



```
interface UserRepository {
    // logika ketika register
    suspend fun register(
        email: String,
        password: String,
        username: String
    ): Flow<ResponseState<Boolean>>

    // logika ketika login
    suspend fun login(email: String, password: String): Flow<ResponseState<Boolean>>
}
```

Code Program 37 Register & Login

Apa itu *suspend*?.. Kalau kita menggunakan *suspend fun*, maka *fun* atau logika yang kita jalankan akan dilakukan di latar belakang dan tidak mengganggu *user interface*. Karena jika kita menggunakannya di *user interface*, misalnya nih, ketika pengguna sedang nge-lag dan menghubungi ke *databasenya* lama. Aplikasi akan crash karena *user interface* terlalu lama diam atau *freeze*.

Kalau kalian perhatikan lagi. Fun yang kita buat itu dikembalikan nilainya atau *return* menjadi *Flow<ResponseState<Boolean>>*. Apa itu *flow*, *responseState* dan *boolean*?

Flow adalah aliran *data* yang dinamis atau bisa berubah. Sebelum *flow* ada, dulu biasanya *developer*

menggunakan `liveData`. Jadi, `data` yang diterima nantinya akan menyesuaikan dengan perubahan yang ada, tanpa perlu dijalankan ulang.

`ResponseState` di sini adalah `state` yang akan diterima ketika sedang menjalankan logika. Apakah dia `success`, `error`, `loading` atau `idle` (tidak sedang melakukan apapun). Biasanya, Class `ResponseState` ini kita buat sendiri. Tapi, di sini kita menggunakan dependency `ApiWrapper` yang dibuat oleh `rmaproject`. Kalian bisa melihatnya di `github` semisal kalian penasaran.

Lalu, kenapa `ResponseState` disini ada `booleannya`. Class `ResponseState` di sini membutuhkan `data` apapun atau `Any (T)` agar bisa dikembalikan jikalau logika `success` atau `error` (menyesuaikan kasus).

Setelah itu, kita akan membuat logika ketika sedang `register` atau `login`. Kita pergi ke Class `UserRepoImpl`.



Code Program 38 Class `UserRepoImpl` yang masih kosong

Class Implementation atau Impl ini kita extends dulu ke Interface yang telah kita buat.



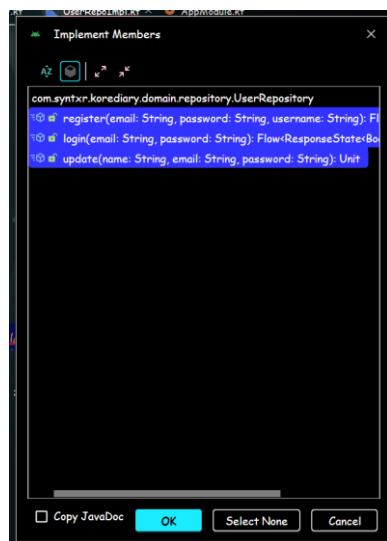
Code Program 39 Extend class ke interface

Setelah itu, kalian akan mendapati error. Bagaimana cara mengatasinya ? kalian klik ke UserRepositoryImpl, lalu tekan Alt + Enter. Maka, akan muncul context menu.



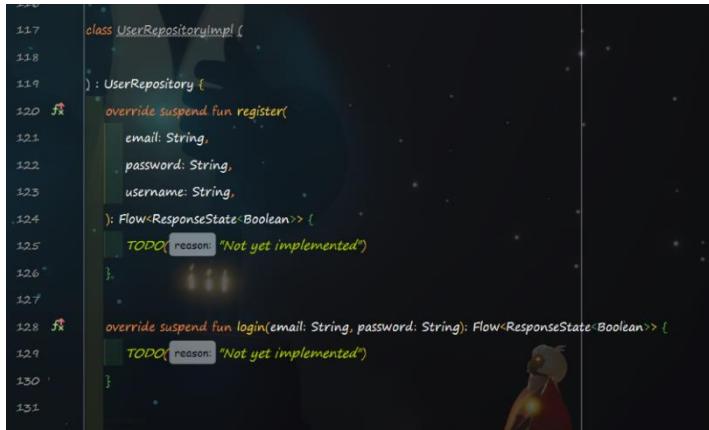
Code Program 40 Context menu

Lalu kalian klik pada “implement members”. Setelah itu akan muncul dialogue.



Code Program 4.1 Dialog implement members

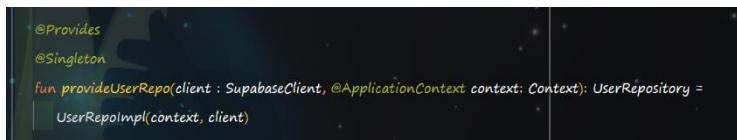
Kita klik ok. Setelah itu akan muncul override *fun* dari *fun* yang telah kita buat sebelumnya di *interface*.



```
117 class UserRepositoryImpl {
118
119 ) : UserRepository {
120     override suspend fun register(
121         email: String,
122         password: String,
123         username: String,
124     ): Flow<ResponseState<Boolean>> {
125         TODO(reason: "Not yet implemented")
126     }
127
128     override suspend fun login(email: String, password: String): Flow<ResponseState<Boolean>> {
129         TODO(reason: "Not yet implemented")
130     }
131 }
```

Code Program 42 Override fun dari members fun interface

Supaya kita bisa menggunakan supabase, kita akan membutuhkan supabase Client yang telah kita buat di AppModule. Kita kembali ke AppModule terlebih dahulu. Kita akan membuat *fun* provideUserRepo agar client tadi bisa di-inject pada UserRepolImpl.



```
@Provides
@Singleton
fun provideUserRepo(client: SupabaseClient, @ApplicationContext context: Context): UserRepository =
    UserRepolImpl(context, client)
```

Code Program 43 Provide user repo

@Provides adalah *annotation* yang digunakan agar *function* yang kita buat bisa di-inject. Lalu @Singleton itu digunakan agar kita bisa menggunakan class atau *function* yang telah di provide tanpa perlu kita mendeklarasikannya

dengan variabel. Jadi, hanya ada satu cara (Single) untuk bisa memanggil class atau functionnya.

Setelah itu kita panggil client yang telah kita inject ke UserRepolml di parameternya. Kita tambahkan @Inject constructor() pada class UserRepolml

```
class UserRepolml @Inject constructor(  
    private val context: Context,  
    private val client : SupabaseClient  
) : UserRepository {
```

Code Program 44 Memanggil client pada parameter

```
override suspend fun register(  
    email: String,  
    password: String,  
    username: String,  
) : Flow<ResponseState<Boolean>> // = adalah return, karena butuh flow maka returnnya flow  
flow { this: FlowCollector<ResponseState<Boolean>>  
    emit(ResponseState.Loading) // status loading  
    try { // pakai try & catch biar tidak crash  
        client.auth.signInWithEmail(Email(thisEmail)) // memanggil auth dari supabase  
        thisEmail = email  
        thisPassword = password  
        data = buildJsonObject(thisJsonObjectBuilder) // menggunakan raw user meta data  
        put("username", username)  
    }  
    emit(ResponseState.Success(data = true)) // status success  
} catch (e: Exception) { // catch error  
    emit(ResponseState.Error(e.message.toString())) // status error  
}
```

Code Program 45 Logika register

Apakah kalian tau apa itu *emit*? *Emit* digunakan untuk me-return *data* yang dibungkus dengan *flow*. Di sini kita melakukan *emit(ResponseState.Loading)* karena sebelum logika berjalan, kita akan menampilkan *loading* terlebih dahulu pada User.

Lalu, kenapa kita pakai try dan catch?. Try digunakan untuk menguji apakah logika bisa berjalan, dan jika bisa kita akan me-return *success*. Catch digunakan untuk menangkap *error*. Pada beberapa kasus, jika kita tidak menggunakan - catch. Error yang terjadi akan membuat aplikasi crash. Sedangkan jika kita menggunakannya, *error* yang didapat bisa kita tampilkan kepada pengguna. Ketika mendapat *error* kita me-returnnya sebagai *error* dengan pesan *error* di dalamnya.

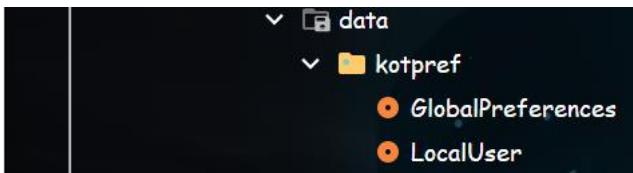
Karena kita hanya menyediakan *register* & *login* dengan *email*, maka di sini kita menggunakan *auth.signInWith>Email*). Kemudian untuk menghindari salah panggil, kita menggunakan *this.email* & *this.password*. Fungsi *this* ini menjelaskan bahwa *email* dan *password* ini adalah variabel dari *auth.signIn* dan bukan dari parameter *fun register()*. Kita akan menyimpan *username* pengguna dengan *raw_user_metadata* yang berbasis JSON. Untuk

menggunakannya kita perlu membuat key dan menyediakan *value*. Karena JSON tidak memiliki val dan hanya menyediakan key dan *value*.

```
data = buildJsonObject { this: JsonObjectBuilder // menggunakan raw user meta data
    put("username", username)
```

Code Program 46 "username" adalah key dan val username adalah value

Setelah membuat logika untuk *register* kita akan membuat logika untuk *login*. Tapi sebelum itu, kita akan membuat Object Local User untuk menyimpan *data user* secara luring atau lokal.



Code Program 47 Object local user

```
import com.chibatching.kotpref.KotprefModel

object LocalUser : KotprefModel() {
    var uuid : String by stringPref(default: "")
    var username : String by stringPref(default: "")
    var email : String by stringPref(default: "")
    var password : String by stringPref(default: "")
}
```

Code Program 48 Variabel di dalam Local User

Umumnya, developer menggunakan shared preferences untuk menyimpan *data* lokal yang tidak terhapus ketika aplikasi ditutup. Di sini kita menggunakan *library* kotpref agar memudahkan kita sehingga kita tidak perlu mendeklarasikan *key* dan *value* untuk menyimpan *data*. Semisal kalian penasaran, kalian bisa mencarinya di internet.

```
override suspend fun login(email: String, password: String): Flow<ResponseState<Boolean>> =  
    Flow { this: FlowCollector<ResponseState<Boolean>> ->  
        emit(ResponseState.Loading) // loading  
        try {  
            client.auth.signInWith(Email) { this: Email.Config } // login  
            this.email = email  
            this.password = password  
        }  
        val user : UserInfo? = client.auth.currentUserOrNull() // mendapatkan data user yang login  
        val publicUser : User = client.from(table: "users") // memeriksa apakah user ada di table users supabase  
            .select(this: PostgresQueryBuilder)  
            .filter(this: PostgresFilterBuilder)  
                User.uuid eq user?.id  
        }.decodeSingle<User>() // decode atau convert data ke class User
```

```
LocalUser.apply { this: LocalUser } // menyimpan data ke lokal
    uuid = publicUser.uuid
    username = publicUser.username
    this.email = email
    this.password = password
}
emit(ResponseState.Success( data: true)) // success
} catch (e: Exception) {
    emit(ResponseState.Error(e.message.toString())) // error
}
}
```

Code Program 49 Logika Login

Setelah *user login*, kita perlu memeriksa di table supabase apakah *user* tersebut ada. Jika ada maka kita ambil *datanya* dan kita *decode single* menjadi *user* *serializable*. Lalu, *data* tersebut disimpan ke local *user*.

Apakah dengan begitu proses auth nya sudah selesai ? . Belum, kita akan membuat supaya ketika pengguna *register* dengan auth, secara otomatis *username* dan *uuid* auth mereka di-*input* di table *users*. Bagaimana caranya? Kita akan menggunakan *function* dan *trigger* di supabase. Kita kembali ke supabase dulu.

Kita sekarang akan menggunakan sql editor untuk membuat supabase dan triggernya.

```
create function public.handle_new_user() --nama function
returns trigger as $$ 
begin
    -- Tulis query SQL disini
    insert into public.users (uuid, username) -- sesuaikan dengan table users kalian
    values (new.id, new.raw_user_meta_data -> 'username');

    -- new itu didapatkan dari auth, karena kita akan mentrigger function ini saat memasukkan user baru, jadinya yang 'new'
    -- berarti yang baru ditambahkan
    -- raw_user_meta_data itu bentuknya JSON, meta_data didapat dari data yang dikirimkan saat melakukan signUp

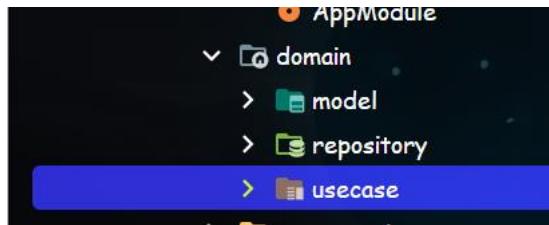
    return new;
end;
$$ language plpgsql security definer;
```

Code Program 50 Function untuk menambahkan data dari auth

```
1  create trigger on_auth_user_created --nama trigger nya
2  | after insert on auth.users --after insert berarti setelah insert function tersebut di trigger
3  |   for each row execute procedure public.handle_new_user(); --sesuaikan dengan nama function nya
```

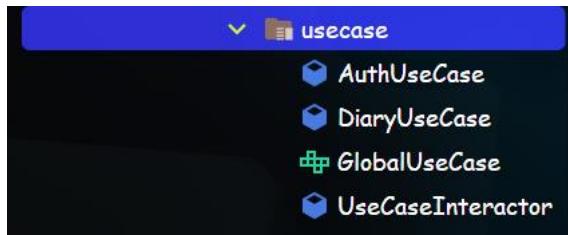
Code Program 51 Trigger untuk menjalankan function yang dibuat

Setelah kita menambahkan kedua hal tersebut, maka secara otomatis *data pengguna* akan ter-*input* ke table. Tapi, logika kita tidak selesai sampai di sini. Selanjutnya, kita akan membuat use case agar bisa dipanggil ke view model.



Code Program 52 Package usecase di dalam package domain

Di dalam package usecase, kita buat Class Interface Global use case, dan Use case interactor.



Code Program 53 Isi package use case

Kita abaikan dulu class “DiaryUseCase”. Kira-kira seperti itulah isinya. Kita akan fokus ke AuthUseCase terlebih dahulu.

```
class AuthUseCase @Inject constructor(
    private val repository: UserRepository,
) {
    suspend fun register(email: String, password: String, username: String): Flow<ResponseState<Boolean>> =
        repository.register(email, password, username)

    suspend fun login(email: String, password: String): Flow<ResponseState<Boolean>> =
        repository.login(email, password)
}
```

Code Program 54 Class AuthUseCase

Use case itu untuk apa sih?. Use case itu untuk menjembatani antara view model dan *data*. Makanya use case ini terletak di domain, karena hal-hal yang ada di domain ini merupakan jembatan antara package *data* dan package presentation nantinya. Di dalam usecase kita juga bisa menyatukan *data* yang diperoleh secara remote ataupun local.

Lalu kenapa AuthUseCase diberi `@Inject?`. Karena AuthUseCase membutuhkan UserRepository yang didapat dari `@Singleton fun provideUserRepo` di dalam App Module.

Setelah itu, kita pergi ke class *interface* globalusecase. Di sana kita akan mendaftarkan auth use case agar dapat dipanggil secara global di aplikasi.

```
3 fx interface GlobalUseCase {  
4 fx     val authUseCase : AuthUseCase  
5 }  
6
```

Code Program 55 Class interface GlobalUseCase

Setelah itu kita buka class use case interactor.

```
1 package com.syntxr.korediary.domain.usecase  
2  
3 > import ...  
4  
5  
6  
7 class UseCaseInteractor(  
8     private val userRepo: UserRepository,  
9     private val diaryRepo: DiaryRepository, // abaikan dulu  
10    private val context: Context  
11 ) : GlobalUseCase {  
12  
13 fx     override val authUseCase : AuthUseCase  
14         get() = AuthUseCase(userRepo)  
15  
16     // abaikan dulu  
17 fx     override val diaryUseCase: DiaryUseCase  
18         get() = DiaryUseCase(diaryRepo, context)  
19  
20 }
```

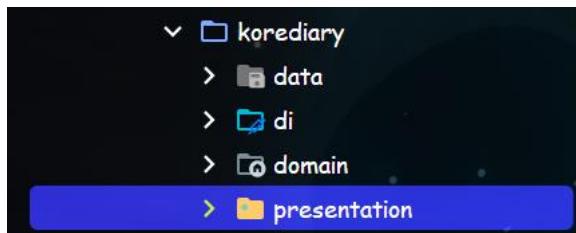
Code Program 56 Class UseCaseInteractor

Kemudian Global Use Case tersebut kita provide di AppModule, supaya bisa di-inject ke dalam view model.

```
@Provides  
@Singleton  
fun provideUseCase(  
    user: UserRepository,  
    diary: DiaryRepository,  
    @ApplicationContext context: Context  
) : GlobalUseCase = UseCaseInteractor(user, diary, context)
```

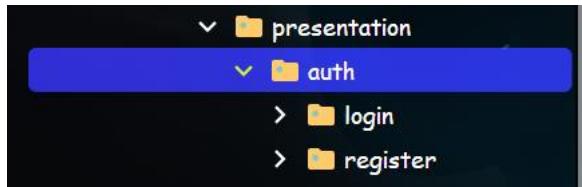
Code Program 57 Provide UseCase di AppModule

Setelah itu kita membuat package baru bernama “presentation”. Pada package inilah kita akan membuat User Interface dan view model.



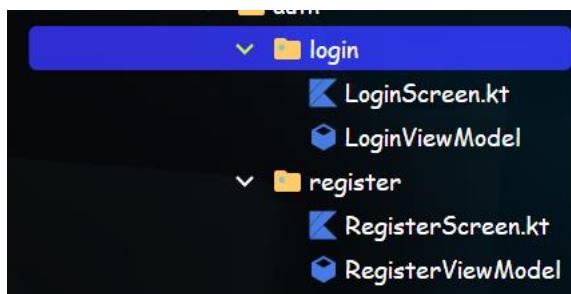
Code Program 58 Package presentation

Di dalam package presentation, kita buat package auth. Di dalamnya lagi kita buat package untuk *login* dan *register*.



Code Program 59 Package login & register

Di dalam masing-masing package, kita buat satu file kotlin untuk *screen* dan satu class untuk *view model*.



Code Program 60 Screen & View Model

Kita akan mulai dari *login* terlebih dahulu, lalu kita buat class *LoginViewModel* menjadi seperti ini. Jangan lupa buat juga untuk *register*.

```
@HiltViewModel
class LoginViewModel @Inject constructor(
    globalUseCase: GlobalUseCase
) : ViewModel() {

    private val _email: MutableStateFlow<String> = MutableStateFlow("value") // membuat val yang berisi initial value untuk email
    val email: Flow<String> = _email // val email yang menggunakan flow agar perubahan data nya terjadi langsung bisa dipanggil ke screen

    private val _password: MutableStateFlow<String> = MutableStateFlow("value") // sama kayak email tapi untuk password
    val password: Flow<String> = _password

    fun onEmailChange(email: String) { // function untuk menerima perubahan data yang di input dari pengguna
        _email.value = email // memasukkan data yang diterima ke val email
    }

    fun onPasswordChange(password: String) { // sama kayak email
        _password.value = password
    }
}
```

```
private val _loginState: MutableStateFlow<ResponseState<Boolean>> = MutableStateFlow<ResponseState<Boolean>>(ResponseState.Idle)
// membuat val yang berisi ResponseState<Boolean> dengan flow, initial valuenya ResponseState.Idle
val loginState: StateFlow<ResponseState<Boolean>> = _loginState.asStateFlow()
    .stateIn(
        viewModelScope,
        SharingStarted.WhileSubscribed(5000),
        ResponseState.Idle
    )
// val loginState dibuat state flow agar bisa dipanggil valuenya di screen

private val authUC: AuthUseCase = globalUseCase.authUseCase // memanggil AuthUseCase

// fun yang memanggil logika login dari repository melalui gsecase yang telah dibuat
fun login() {
    viewModelScope.launch { thisCoroutineScope
        loginState.emitAll(authUC.login(_email.value, _password.value))
    }
}
```

Code Program 61 LoginViewModel

Karena kita menggunakan hilt-dagger, pastikan view model kalian dilengkapi dengan `@HiltViewModel` dan mempunyai `@Inject constructor ()`. Dengan begini kita telah

menyelesaikan logika autentikasi. Selanjutnya kita akan membahas tentang table *post*.



Bagian 5 : Membuat Aplikasi (Post)

5.1 Menyiapkan *repository* dan local *database*

Apa yang bisa dilakukan oleh pengguna dengan table *post*? Mereka bisa melakukan *insert* dengan membuat diary yang baru. Mereka juga bisa melakukan *update* dengan mengubah isi dari diary. Mereka bisa menghapus diary yang mereka buat dan mereka akan melihat *list* diary yang mereka buat.

Hal-hal yang disebutkan di atas merupakan dasar dalam mengolah *database*. CREATE, READ, UPDATE, dan DELETE. Kalau kalian sebelumnya pernah belajar SQL atau pun PostgreSQL hal ini lah yang akan kalian pelajari pertama kali.

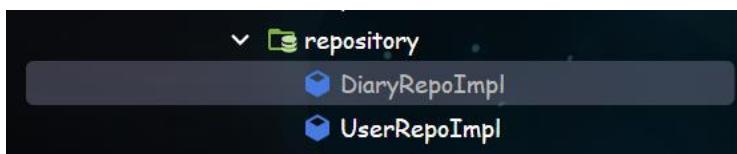
Karena kita menggunakan supabase yang dasarnya dari *postgreSql*, maka kita akan menggunakan Postgre yang telah disediakan supabase.

Seperti yang kita lakukan pada Auth, kita akan membuat *interface* dan class untuk *repository* terlebih

dahulu. Saya memberinya nama DiaryRepository. Tapi, *it's up to you* lah.



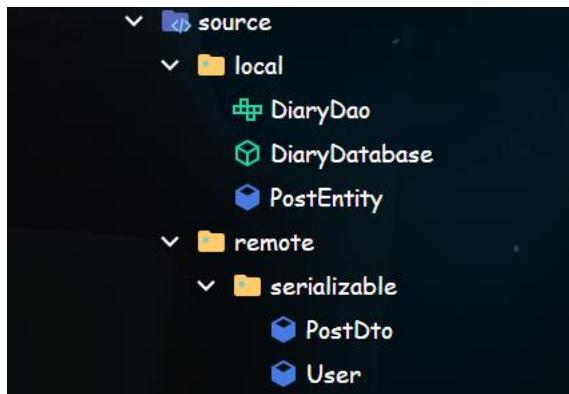
Code Program 62 Class Interface Diary repository



Code Program 63 Class DiaryRepoImpl

Sebelum kita mulai membuat logika, kita akan membuat *database* local, entity dan class untuk Post. Kenapa begitu ? Kita akan membuat agar ketika pengguna memiliki internet dan ingin menyimpannya sebagai favorite, *data* tersebut akan disimpan secara offline. Jadi, pengguna tetap

bisa melihat *data* mereka meskipun tidak sedang tersambung ke internet.



Code Program 64 Package local dan isinya

Pertama, kita akan membuat PostEntity terlebih dahulu. Kurang lebih seperti ini.

```
@Entity(tableName: "post")
data class PostEntity (
    @PrimaryKey
    val uuid : String,
    val title : String,
    val value : String,
    val mood : String,
    val userId : String,
    val createdAt : String,
)
```

Code Program 65 PostEntity

Apa itu entity? Entity adalah class yang berfungsi sebagai table, Karena di sini kita menggunakan Room database yang berdasar pada SQL. Kemudian kita membuat Dao nya.

```
interface DiaryDao {  
  
    @aorync  
    @Insert  
    suspend fun insert(postEntity: PostEntity)  
  
    @aorync  
    @Query("SELECT * FROM post")  
    fun getStoredData(): Flow<List<PostEntity>>  
  
    @aorync  
    @Query("DELETE FROM post")  
    suspend fun clear()  
  
    @aorync  
    @Query("DELETE FROM post WHERE uuid = :uuid")  
    suspend fun delete(uuid: String)  
}
```

Code Program 66 DiaryDao

Dao atau Data Access Object digunakan agar kita dapat mengolah *database* dan *table* yang kita buat dengan CRUD (Create, Read, Update dan Delete). Sebagaimana yang kalian liat di atas, ada *insert*, *upsert* (*update* dan *insert*), *delete*, dan *select* (*read*).

Setelah itu kita buat *databasenya*, seperti di bawah ini.

```
@Database(entities = [PostEntity::class], version = 1, exportSchema = true)
abstract class DiaryDatabase : RoomDatabase() {
    abstract val dao: DiaryDao

    companion object {
        const val DB_NAME = "diary.db"
    }
}
```

Code Program 67 DiaryDatabase

Kemudian kita provide *databasenya* di AppModule, supaya bisa di-*inject* di DiaryRepolmpl.

```
@Provides  
@Singleton  
fun provideDatabase(  
    @ApplicationContext context: Context,  
) : DiaryDatabase = Room.databaseBuilder(  
    context,  
    DiaryDatabase::class.java,  
    DiaryDatabase.DB_NAME  
.build()
```

Code Program 68 Provide database di AppModule

Setelah di provide, kita lanjut membuat class Post di domain.



Code Program 69 Class Post

Lalu kita buat class PostDto menjadi seperti ini.

```
data class Post {  
    val uuid : String,  
    val title : String,  
    val value : String,  
    val mood : String,  
    val userId : String,  
    val createdAt : String,  
}  
)  
fun toPostDto() : PostDto = PostDto(  
    uuid = uuid,  
    title = title,  
    value = value,  
    mood = mood,  
    userId = userId,  
    createdAt = createdAt  
)  
}
```

Code Program 70 Class Post

Abaikan saja dulu `fun toPostDto()`. Itu akan kita gunakan nanti. Pada Class PostDto dan PostEntity kita tambahkan menjadi.

```
@Serializable  
data class PostDto(  
    @SerializedName("id") val id: Int? = null,  
    @SerializedName("uuid") val uuid: String,  
    @SerializedName("user_id") val userId: String,  
    @SerializedName("title") val title: String,  
    @SerializedName("value") val value: String,  
    @SerializedName("mood") val mood: String,  
    @SerializedName("published") val published: Boolean = false,  
    @SerializedName("created_at") val createdAt: String  
)  
{  
    @async  
    fun toPostEntity(): PostEntity = PostEntity(  
        uuid = uuid,  
        title = title,  
        value = value,  
        mood = mood,  
        userId = userId,  
        createdAt = createdAt  
    )  
}
```

Code Program 71 fun toPostEntity() pada Class PostDto

```
@Entity(tableName: "post")
data class PostEntity(
    @PrimaryKey
    val uuid : String,
    val title : String,
    val value : String,
    val mood : String,
    val userId : String,
    val createdAt : String,
)
{
    fun toPost() : Post = Post(
        uuid = uuid,
        title = title,
        value = value,
        mood = mood,
        userId = userId,
        createdAt = createdAt
    )
}
```

Code Program 72 Fun toPost() pada Class PostEntity

Pada DiaryRepoImpl, kita lakukan seperti di AuthRepoImpl. Jangan lupa untuk provide dulu di AppModule agar bisa di-Inject.

```
class DiaryRepolmpl @Inject constructor(  
    context: Context,  
    private val client: SupabaseClient,  
    private val dao: DiaryDao,  
) : DiaryRepository {
```

Code Program 73 DiaryRepolmpl

```
@Provides  
@Singleton  
fun provideDiaryRepo(client: SupabaseClient, db: DiaryDatabase, @ApplicationContext context: Context): DiaryRepository =  
    DiaryRepolmpl(context, client, db.dao)
```

Code Program 74 Provide Diary Repo di AppModule

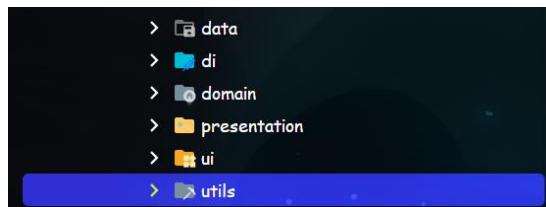
Sebelum kita masuk ke logika, mari kita berkenalan dengan worker terlebih dahulu.

5.2 Worker

Kenapa kita menggunakan worker?. Apakah kalian pernah ketika sedang mengunduh video dari youtube atau sedang mengunduh aplikasi dari google play store, apakah kalian melihat adanya notif ?. Proses unduhan dan notif yang diterima oleh pengguna dijalankan di latar belakang oleh worker. Meskipun pengguna sedang menutup aplikasi, logika yang dibuat akan tetap dijalankan.

Nah, worker ini cocok dengan kasus yang kita miliki. Yaitu, CRUD ke *database* online. Jika pengguna memiliki internet, maka logika akan dijalankan. Jika tidak berhasil dijalankan, maka akan diulang sampai beberapa kali, baru kemudian dianggap *error* atau gagal. Logik yang menggunakan worker akan tetap berjalan meskipun pengguna sudah menutup aplikasi. Jika logika berhasil, kita akan menampilkan notifikasi bahwa logika berhasil dijalankan

Agar worker bisa menerima *data*, kita perlu key agar *data* yang dikirim bisa diambil dengan benar. Jadi, pertama-tama kita akan membuat key untuk mengirim *data* ke worker.



Code Program 75 Membuat package utils

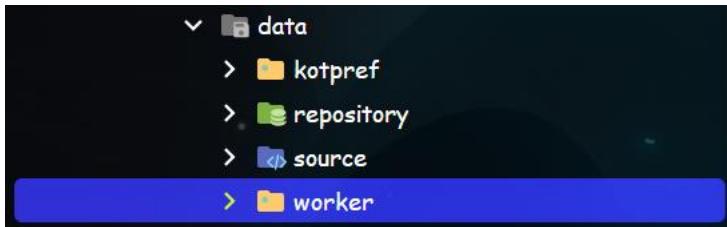


Code Program 76 Membuat File Constant

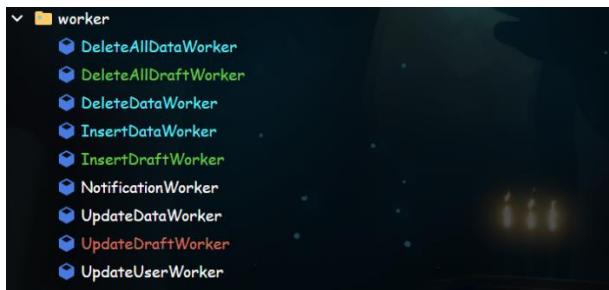
```
const val KEY_UUID = "KEY_UUID"
const val KEY_USER = "KEY_USER"
const val KEY_NAME = "KEY_NAME"
const val KEY_EMAIL = "KEY_EMAIL"
const val KEY_PASSWORD = "KEY_PASSWORD"
const val KEY_TITLE = "KEY_TITLE"
const val KEY_PUBLISH = "KEY_PUBLISH"
const val KEY_VALUE = "KEY_VALUE"
const val KEY_MOOD = "KEY_MOOD"
const val KEY_DATE = "KEY_DATE"
const val KEY_TITLE_NOTIFY = "KEY_TITLE_NOTIFY"
const val KEY_TXT_NOTIFY = "KEY_TXT_NOTIFY"
```

Code Program 77 Key untuk data worker

Setelah itu, kita membuat package baru untuk worker di dalam package *data*. Package ini akan berisi class-class untuk worker yang akan kita buat.



Code Program 78 Package worker



Code Program 79 Class yang ada di dalam package worker

Kita akan mulai dari NotificationWorker, karena NotificationWorker akan dipakai di class worker lainnya untuk dijalankan ketika class worker tersebut success.

```
@HiltWorker
class NotificationWorker @AssistedInject constructor( // untuk worker kita menggunakan @AssistedInject
    @Assisted private val applicationContext: Context, // karena worker menggunakan CoroutineWorker, maka butuh context
    @Assisted params: WorkerParameters, // data akan dikirim lewat sini
) : CoroutineWorker(applicationContext, params) {
    ...
    override suspend fun doWork(): Result { // worker akan dikerjakan lewat function ini
        Notify // kita menggunakan library notification yang di buat oleh Karo, kalau penasaran bisa dilihat di github
            .with(applicationContext)
            .content { this.Payload.Content.Default(
                title = inputData.getString(KEY_TITLE_NOTIFY).toString(),
                // title yang kita dapat dari params melalui inputData.getString()
                text = inputData.getString(KEY_TXT_NOTIFY).toString(),
                // text yang kita dapat dari params melalui inputData.getString()
            ) }
            .show()
        return Result.success() // success
    }
}
```

Code Program 80 NotificationWorker

Saya ingatkan lagi, karena kita pakai hilt-dagger, maka jangan lupa untuk menggunakan annotasi yang disediakan oleh hilt-dagger - `@HiltWorker` -. Untuk worker, hilt menyediakan annotasi yang berbeda untuk *injection* dengan menggunakan `@AssistedInject`.

Kita akan membuat logika di dalam *function* `doWork()`, karena di sanalah logika akan dijalankan. Function ini mengembalikan `Result`. `Result` adalah hasil yang dikembalikan ketika logika dijalankan, apakah dia *success*? atau *failure*. `Result` juga memiliki *retry* untuk menjalankan kembali logika jika diperlukan. `Result` memang mirip dengan `ResponseState`, bedanya `Result` disediakan langsung oleh kotlin.

5.3 CRUD dengan Worker

Seperi ketika kita membuat NotificationWorker, kita akan membuat logika CRUD database di dalam *fun doWork()*. Kita mulai dari InsertDataWorker terlebih dahulu.

```
@HiltWorker
class InsertDataWorker @AssistedInject constructor(
    @Assisted private val applicationContext: Context,
    @Assisted params: WorkerParameters,
    private val client: SupabaseClient, // karena membutuhkan supabase, kita inject saja
    private val db: DiaryDatabase // karena membutuhkan local database, maka di inject
) : CoroutineWorker(applicationContext, params) {

    private val workManager: WorkManager = WorkManager.getInstance(applicationContext) // deklarasi work manager agar bisa menggunakan notification worker
    private val constraint: Constraints.Builder = Constraints.Builder() // membuat constraint untuk notification worker
        .setRequiredNetworkType(NetworkType.CONNECTED) // mengatur agar Worker hanya dijalankan ketika tersambung ke internet
    private val notifyBuilder: OneTimeWorkRequestBuilder = OneTimeWorkRequestBuilder(NotificationWorker::class.java)

    // val ini berisi Notification Worker
    // OneTimeWorkRequest berarti kita hanya akan menjalankan worker ini sekali, bukan selama periode tertentu
}
```

```
override suspend fun doWork(): Result {
    return withContext(Dispatchers.IO) [thisCoroutineScope] // menggunakan Dispatcher.IO agar berjalan di background Thread atau latar belakang

    val postDto = PostDto( // membuat PostDto dengan data yang didapat dari params melalui inputData
        uuid = inputData.getString(KEY_UUID).toString(),
        userId = inputData.getString(KEY_USER).toString(),
        value = inputData.getString(KEY_VALUE).toString(),
        mood = inputData.getString(KEY_MOOD).toString(),
        title = inputData.getString(KEY_TITLE).toString(),
        createdAt = inputData.getString(KEY_DATE).toString()
    )

    return@withContext try { // jangan lupa try catch biar tidak crash
        delay(3000) // delay selama 3 detik

        client.from("posts").insert( // insert ke supabase
            postDto
        )
    } catch (e: Exception) {
        e.printStackTrace()
    }
}
```



```

notifyBuilder.set inputData( // mengirim title dan text yang dibutuhkan di notification worker dengan Data Builder
    Data.Builder()
        .putString(KEY_TITLE_NOTIFY, "Insert Post") // title
        .putString(KEY_TXT_NOTIFY, "Successfully insert your post") // text
        .build() // jangan lupa ini
    )

    notifyBuilder.setConstraints(constraint.build()) // memasukkan constraint ke notifyBuilder

    Result.success().apply { this.Result // kalo success menjalankan work manager agar notification worker berjalan
        workManager.enqueueUniqueWork(
            postDto.uuid, // worker perlu id dengan tipe data string yang unik
            ExistingWorkPolicy.APPEND_OR_REPLACE, // kalo ada worker yang sama akan direplace
            notifyBuilder.build() // jangan lupa dibuild val notifyBuilder nya
        )
    }

} catch (e : Exception){ //catch
    if (runAttemptCount >= 5) { // kalo worker ini dijalankan sampai atau lebih dari lima kali
        db.dao.insert(postDto.toPostEntity()) // insert ke local
        // karena insert menggunakan entity, maka menggunakan ekstensi toPostEntity()

        notifyBuilder.set inputData(
            Data.Builder()
                .putString(KEY_TITLE_NOTIFY, "Insert Post")
                .putString(KEY_TXT_NOTIFY, "Failed insert your post").build()
        )
    }

    notifyBuilder.setConstraints(constraint.build())
    Result.failure().apply { this.Result // kalo gagal menjalankan...
        workManager.enqueueUniqueWork(
            postDto.uuid,
            ExistingWorkPolicy.APPEND_OR_REPLACE,
            notifyBuilder.build()
        )
    }
}

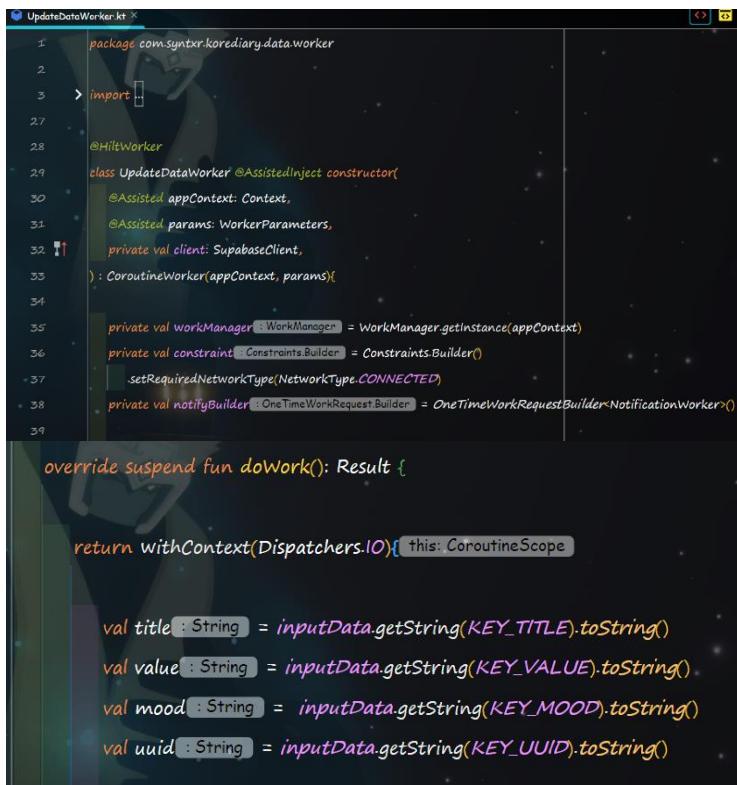
} // batas akhir run attempt count
Result.retry() // coba ulang

```

Code Program 81 InsertDataWorker

Untuk yang *retry*, jadi apabila logika ternyata *error*, logika akan dijalankan ulang sampai atau lebih dari lima kali, baru kemudian dianggap *failure*.

Kita lakukan ke class worker lainnya, hampir sama dengan yang ada di `InsertDataWorker`.



The screenshot shows the code for `UpdateDataWorker.kt` in an Android Studio code editor. The code is annotated with line numbers from 1 to 39. It defines a `CoroutineWorker` named `UpdateDataWorker` that takes `appContext` and `params` as parameters. Inside the class, it initializes `workManager`, `constraint`, and `notifyBuilder`. The `doWork` method uses `withContext(Dispatchers.IO)` to perform work on the IO dispatcher. It then extracts values from `inputData` using `getString` methods for `KEY_TITLE`, `KEY_VALUE`, `KEY_MOOD`, and `KEY_UUID`.

```
1 package com.syntxk.koredairy.data.worker
2
3 > import ...
4
5 @HiltWorker
6
7 class UpdateDataWorker @AssistedInject constructor(
8     @Assisted appContext: Context,
9     @Assisted params: WorkerParameters,
10    private val client: SupabaseClient,
11 ) : CoroutineWorker(appContext, params){
12
13
14     private val workManager: WorkManager = WorkManager.getInstance(appContext)
15     private val constraint: ConstraintsBuilder = ConstraintsBuilder()
16         .setRequiredNetworkType(NetworkType.CONNECTED)
17     private val notifyBuilder: OneTimeWorkRequestBuilder<NotificationWorker>()
18
19
20     override suspend fun doWork(): Result {
21
22         return withContext(Dispatchers.IO)(this: CoroutineScope) {
23
24             val title: String = inputData.getString(KEY_TITLE).toString()
25             val value: String = inputData.getString(KEY_VALUE).toString()
26             val mood: String = inputData.getString(KEY_MOOD).toString()
27             val uuid: String = inputData.getString(KEY_UUID).toString()
28
29         }
30     }
31
32 }
```

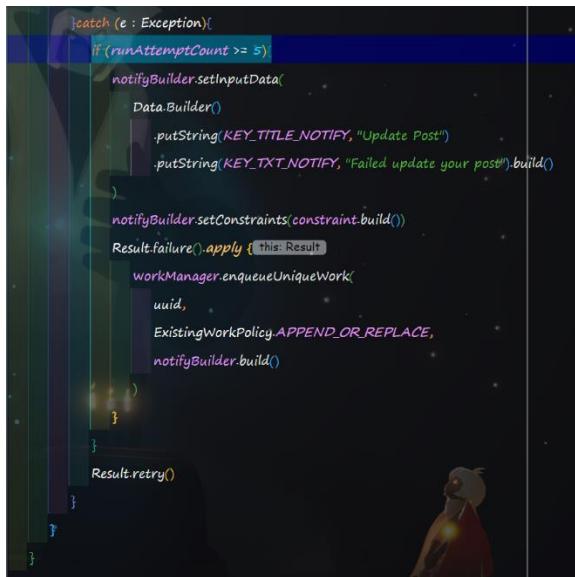
```
return@withContext try {
    delay(timeMillis: 2000)
    client.from(table: "posts").update(
        update = { this: PostgresUpdate<Post> ->
            PostDto::title setTo title
            PostDto::mood setTo mood
            PostDto::value setTo value
        },
        request = { this: PostgresRequestBuilder<Post> ->
            filter { this: PostgresFilterBuilder<Post> -
                PostDto::uuid eq uuid
            }
        }
    )
}

notifyBuilder.setInputData(
    Data.Builder<*>(
        .putString(KEY_TITLE_NOTIFY, "Update Post")
        .putString(KEY_TXT_NOTIFY, "Successfully update your post")
    ).build()
)
```

```
notifyBuilder.setInputData(
    Data.Builder<*>(
        .putString(KEY_TITLE_NOTIFY, "Update Post")
        .putString(KEY_TXT_NOTIFY, "Successfully update your post")
    ).build()
)

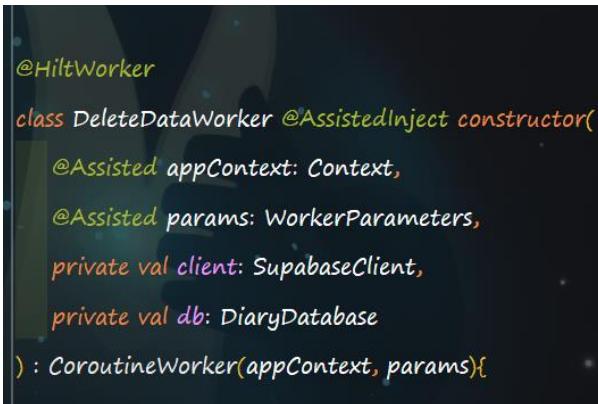
notifyBuilder.setConstraints(constraint.build())

Result.success().apply { this: Result<*> -
    workManager.enqueueUniqueWork(
        uuid,
        ExistingWorkPolicy.APPEND_OR_REPLACE,
        notifyBuilder.build()
    )
}
```



```
    } catch (e : Exception){
        if (runAttemptCount >= 5)
            notifyBuilder.setInputData(
                Data.Builder()
                    .putString(KEY_TITLE_NOTIFY, "Update Post")
                    .putString(KEY_TXT_NOTIFY, "Failed update your post").build()
            )
        notifyBuilder.setConstraints(constraint.build())
    }
    Result.failure().apply { this: Result
        workManager.enqueueUniqueWork(
            uuid,
            ExistingWorkPolicy.APPEND_OR_REPLACE,
            notifyBuilder.build()
        )
    }
}
Result.retry()
}
}
}
```

Code Program 82 UpdateDataWorker



```
@HiltWorker
class DeleteDataWorker @AssistedInject constructor(
    @Assisted applicationContext: Context,
    @Assisted params: WorkerParameters,
    private val client: SupabaseClient,
    private val db: DiaryDatabase
) : CoroutineWorker(applicationContext, params){
```

```
override suspend fun doWork(): Result {
    return withContext(Dispatchers.IO) {
        val uuid: String = inputData.getString(KEY_UUID, "0")
        return@withContext try {
            delay(timeMillis: 3000)

            client.from(table: "posts").delete(this: PostgresRequestBuilder)
                .filter(this: PostgresFilterBuilder {
                    Post::uuid eq uuid
                    // memeriksa apakah ada post dengan uid yang sesuai
                })
                .execute()
                .Result.success()
        } catch (e: Exception) {
            db.dao.delete(uuid)
            Result.retry()
        }
    }
}
```

Code Program 83 DeleteDataWorker

```
@HiltWorker
class DeleteAllDraftWorker @AssistedInject constructor(
    @Assisted applicationContext: Context,
    @Assisted params: WorkerParameters,
    private val client: SupabaseClient,
) : CoroutineWorker(applicationContext, params) {
```

```
private val workManager : WorkManager = WorkManager.getInstance(appContext)
private val constraint : Constraints.Builder = Constraints.Builder()
    .setRequiredNetworkType(NetworkType.CONNECTED)
private val notifyBuilder : OneTimeWorkRequestBuilder<NotificationWorker>()
```

```
override suspend fun doWork(): Result {
    return withContext(Dispatchers.IO) { this: CoroutineScope
        val uuid : String = LocalUser.uuid
        return@withContext try {
            delay(timeMillis: 2000)
            client.from(table: "posts").delete { this: PostgrexRequestBuilder
                filter { this: PostgrexFILTERBuilder
                    PostDto::userId eq uuid
                    PostDto::published eq false
                }
            }
            notifyBuilder.setInputData(
                Data.Builder()
                    .putString(KEY_TITLE_NOTIFY, "Delete All Draft")
                    .putString(KEY_TXT_NOTIFY, "Successfully delete all your draft").build()
            )
            notifyBuilder.setConstraints(constraint.build())
        }
    }
}
```

```
        Result.success().apply { this: Result
            workManager.enqueueUniqueWork(
                uuid,
                ExistingWorkPolicy.APPEND_OR_REPLACE,
                notifyBuilder.build()
            )
        }
    } catch (e: Exception) {
    if (runAttemptCount >= 5L)
        notifyBuilder.setInputData(
            Data.Builder()
                .putString(KEY_TITLE_NOTIFY, "Delete All Draft")
                .putString(KEY_TXT_NOTIFY, "Failed delete all your draft").build()
        )
    notifyBuilder.setConstraints(constraint.build())
}
Result.failure().apply { this: Result
    workManager.enqueueUniqueWork(
        uuid,
        ExistingWorkPolicy.APPEND_OR_REPLACE,
        notifyBuilder.build()
    )
}
}
Result.retry()
```

Code Program 84 DeleteAllDraftWorker

Kurang lebih semuanya sama kecuali logikanya saja.
Kalian bisa menyesuaikan untuk notifikasi pada setiap worker.

```
return@WithContext try {
    delay( timeMillis: 2000 )
    client.from( table: "posts" ).delete {
        this: PostgresRequestBuilder
        filter { this: PostgresFilterBuilder
            PostDto::userId eq uid
            PostDto::published eq true
        }
    }
}
```

Code Program 85 DeleteAllDataWorker

```
return@withContext try {
    delay( timeMillis: 3000 )

    client.from( table: "posts" ).delete( this: PostgresRequestBuilder.RowFilter )
        filter { this: PostgresFilterBuilder -
            Post::uuid eq uuid
            // memeriksa apakah ada post dengan uuid yang sesuai
        }
}
```

Code Program 86 DeleteDataWorker

```
return@withContext(Dispatchers.IO) { this: CoroutineScope -
    val postDto = PostDto(
        uuid = inputData.getString(KEY_UUID).toString(),
        userId = inputData.getString(KEY_USER).toString(),
        value = inputData.getString(KEY_VALUE).toString(),
        mood = inputData.getString(KEY_MOOD).toString(),
        title = inputData.getString(KEY_TITLE).toString(),
        published = inputData.getBoolean(KEY_PUBLISH, defaultValue: false),
        createdAt = inputData.getString(KEY_DATE).toString()
    )

    return@withContext try {
        delay( timeMillis: 3000 )

        client.from( table: "posts" ).insert(
            postDto
        )
    }
}
```

Code Program 87 InsertDraftWorker

```
override suspend fun doWork(): Result {
    return@WithContext Dispatchers.IO { this: CoroutineScope -
        val uuid : String = inputData.getString(KEY_UUID).toString()
        val publish : Boolean = inputData.getBoolean(KEY_PUBLISH, defaultValue: true)

        return@WithContext try {
            client.from(table: "posts").update(
                update = { this: PostgresUpdate -
                    PostDto::published setTo publish
                },
                request = { this: PostgresRequestBuilder -
                    filter { this: PostgresFilterBuilder -
                        PostDto::uuid eq uuid
                    }
                }
            )
        }
    }
}
```

Code Program 88 UpdateDraftWorker

```
@HiltWorker
class UpdateUserWorker @AssistedInject constructor(
    @Assisted applicationContext: Context,
    @Assisted params: WorkerParameters,
    private val client: SupabaseClient,
) : CoroutineWorker(applicationContext, params) {
```

```
private val workManager : WorkManager = WorkManager.getInstance(applicationContext)
private val constraint : Constraints.Builder = Constraints.Builder()
    .setRequiredNetworkType(NetworkType.CONNECTED)
private val notifyBuilder : OneTimeWorkRequest.Builder = OneTimeWorkRequestBuilder<NotificationWorker>()
```

```
override suspend fun doWork(): Result {
    return@withContext Dispatchers.IO { this: CoroutineScope
        return@withContext try {
            delay(1000L)
            val name: String = inputData.getString(KEY_NAME).toString()
            val email: String = inputData.getString(KEY_EMAIL).toString()
            val password: String = inputData.getString(KEY_PASSWORD).toString()

            client.auth.signInWithEmailAndPassword(this: EmailConfig) // karena update ke auth butuh signin
            this.email = LocalUser.email
            this.password = LocalUser.password
        }
        catch (e: Exception) {
            Log.e("doWork", "Error during sign-in: ${e.message}")
        }
    }
}

client.auth.updateUser { this: UserUpdateBuilder } // update data ke auth
this.email = email
this.password = password
this.data { this: JsonObjectBuilder
    put("username", name)
}
```

```
client.from(table: "users").update() // update username ke user
update = { this: PostgresUpdate
    User.username.setTo(name)
},
request = { this: PostgresRequestBuilder } // filter apakah ada user yang memiliki uuid yang sesuai ?
filter { this: PostgresFilterBuilder
    .User.uuid eq LocalUser.uuid
}
)

LocalUser.apply { this: LocalUser } // simpan perubahan ke local
username = name
this.email = email
this.password = password
}
```

```
        notifyBuilder.setInputData(  
            Data.Builder()  
                .putString(KEY_TITLE_NOTIFY, "Update Profile")  
                .putString(KEY_TXT_NOTIFY, "Successfully update your profile").build()  
        )  
        notifyBuilder.setConstraints(constraint.build())  
  
    }  
  
    Result.success().apply { this: Result  
        workManager.enqueueUniqueWork(  
            LocalUser.uid,  
            ExistingWorkPolicy.APPEND_OR_REPLACE,  
            notifyBuilder.build()  
        )  
    }  
}
```

```
} catch (e: Exception) {  
    if (runAttemptCount >= 5){  
        notifyBuilder.setInputData(  
            Data.Builder()  
                .putString(KEY_TITLE_NOTIFY, "Update Profile")  
                .putString(KEY_TXT_NOTIFY, "Failed update your profile").build()  
        )  
        notifyBuilder.setConstraints(constraint.build())  
        Result.failure().apply { this: Result  
            workManager.enqueueUniqueWork(  
                LocalUser.uid,  
                ExistingWorkPolicy.APPEND_OR_REPLACE,  
                notifyBuilder.build()  
            )  
        }  
    }  
    Result.retry()  
}
```

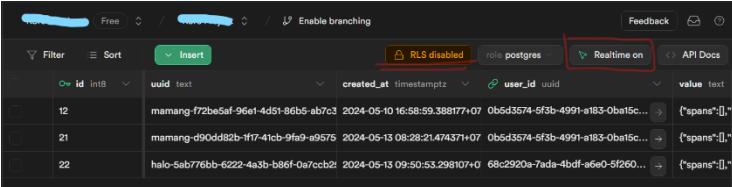
Code Program 89 UpdateUserWorker

Setelah kita selesai membuat worker, saatnya kita masuk ke *implementation* di *repositoryImpl*.

5.4 Get data dengan realtime

Kalau kalian sadari, kita tidak membuat `GetDataWorker` untuk mendapatkan *data* yang ada pada supabase. Kenapa tuh?. Kita akan menggunakan Realtime sebagai gantinya.

Lalu, bagaimana cara menggunakannya?. Pertama-tama kita pergi dulu ke supabase. Pastikan kalian telah mengaktifkan realtime pada table “*posts*” kalian.



id	uuid	created_at	user_id	value
12	mamang-f72be5af-96ef-4d51-86b5-ab7c3	2024-05-10 16:58:59.388177+07	0b5d3574-5f3b-4991-a183-0ba15c...	{"spans":[], "...
21	mamang-d90dd82b-1f17-41cb-9fa9-a9575	2024-05-13 08:28:21.474371+07	0b5d3574-5f3b-4991-a183-0ba15c...	{"spans":[], "...
22	halo-5ab776bb-6222-4a3b-b86f-0a7ccb21	2024-05-13 09:50:53.298101+0	68c2920a-7ada-4bdf-a6e0-5f260...	{"spans":[], "...

Untuk Row-Level-Security (RLS) kita buat *disabled* saja, agar kita tidak perlu menambahkan *policy*. Hal yang sama berlaku pada table *users* (RLS-nya aja, jangan realtime).

Selanjutnya kita deklarasikan dulu sebuah channel. Jadi, realtime itu ibarat kita menggunakan youtube. Kalau kita ingin mendapatkan video dan pembaruan video dari sebuah channel, kita harus *subscribe* terlebih dahulu channel tersebut. Nama channel nya bebas, di sini saya pakai channel “diaries”.

```
class DiaryRepoImpl @Inject constructor(  
    context: Context,  
    private val client: SupabaseClient,  
    private val dao: DiaryDao,  
) : DiaryRepository {  
  
    private val diaryChannel: RealtimeChannel = client.channel(channelId: "diaries")
```

Code Program 90 Private val channel

Baru setelah itu kita buat logikanya. Kita subscribe ke channel agar mendapatkan pembaruan *data* secara langsung, baik ketika *user* telah *insert* atau bahkan telah *delete*. Setelah membuat logika untuk mendapatkan *data* dengan *subscribe*, kita juga perlu untuk membuat fungsi *unsubscribe*. Hal ini baru terlihat nanti ketika kita mengimplementasikannya pada screen.

```
override suspend fun fetch(): Result<Flow<List<Post>>> { // mengembalikan ke result

    val data : Flow<List<PostDto>> = diaryChannel.postgresListDataFlow(
        schema = "public",
        table = "posts",
        primaryKey = PostDto::id,
        filter = FilterOperation( // filter, karena hanya mengambil post yang dimiliki user saat ini
            column = "user_id",
            operator = FilterOperator.EQ,
            value = LocalUser.uuid
        ),
        flowOn(Dispatchers.IO)
    diaryChannel.subscribe() // subscribe ke channel yang dibuat

    return Result.success(data.map { list : List<PostDto> ->
        // pakai ekstensi toPost() karena perlu List<Post>
        list.filter { it.published }.map { it.toPostEntity().toPost() }
    })
}
```

Code Program 91 Fun fetch() untuk mendapatkan data secara realtime dari supabase

```
override suspend fun draft(): Result<Flow<List<Post>>> {
    val data : Flow<List<PostDto>> = diaryChannel.postgresListDataFlow(
        schema = "public",
        table = "posts",
        primaryKey = PostDto::id,
        filter = FilterOperation( // filter, karena hanya mengambil post yang dimiliki user saat ini
            column = "user_id",
            operator = FilterOperator.EQ,
            value = LocalUser.uuid
        ),
        flowOn(Dispatchers.IO)
    diaryChannel.subscribe()

    return Result.success(data.map { list : List<PostDto> ->
        // pakai ekstensi toPost() karena perlu List<Post>
        list.filter { it.published }.map { it.toPostEntity().toPost() }
    })
}
```

Code Program 92 Mendapatkan draft, tidak realtime

```
    override fun favorite(): Flow<List<Post>> =  
        dao.getStoredData().map { list: List<PostEntity> -> list.map { it.toPost() } }
```

Code Program 93 Mengambil data favorite

```
override suspend fun unsubscribe() {  
    diaryChannel.unsubscribe()  
    client.realtime.removeChannel(diaryChannel)  
}
```

Code Program 94 Fun `unsubscribe()` untuk berhenti mengikuti pembaruan

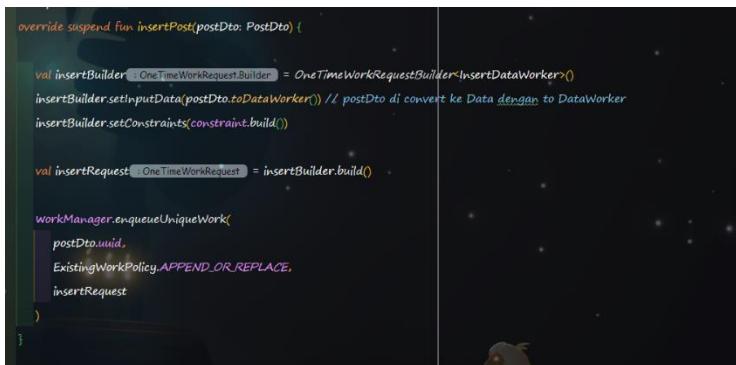
Dengan begitu, kita telah selesai mendapatkan *data* dari *remote (online)* source. Selanjutnya yang akan kita lakukan adalah implementasi worker.

5.5 Implementasi worker pada *repository*

Cara implementasi worker pada *repository* sebenarnya tidak jauh berbeda dengan yang dilakukan pada `NotificationWorker` di `InsertDataWorker` misalnya.

```
private val workManager : WorkManager = WorkManager.getInstance(context)
private val constraint : Constraints.Builder = Constraints.Builder()
    .setRequiredNetworkType(NetworkType.CONNECTED)
```

Code Program 95 Deklarasi Work manager dan constraint



Code Program 96 override suspend fun insert post

Darimana ekstensi `toDataWorker` berasal ?. Kita bisa membuatnya di File Helper seperti ini.



Code Program 97 File helper

```
fun PostDto.toDataWorker(): Data = Data.Builder()
    .putString(KEY_UUID, this.uuid)
    .putString(KEY_USER, this.userId)
    .putString(KEY_TITLE, this.title)
    .putString(KEY_VALUE, this.value)
    .putString(KEY_MOOD, this.mood)
    .putString(KEY_DATE, this.createdAt)
    .build()
```

Code Program 98 Ekstensi `toDataWorker`

Lakukan hal yang sama pada override fun yang membutuhkan *worker*.

```
override suspend fun publish(uuid: String, publish: Boolean) {
    val updateBuilder: OneTimeWorkRequest.Builder = OneTimeWorkRequestBuilder<UpdateDraftWorker>()

    updateBuilder.setInputData(
        Data.Builder().putBoolean(KEY_PUBLISH, publish)
            .putString(KEY_UUID, uuid).build()
    )
    updateBuilder.setConstraints(constraint.build())

    val updateRequest: OneTimeWorkRequest = updateBuilder.build()

    workManager.enqueueUniqueWork(
        uuid,
        ExistingWorkPolicy.APPEND_OR_REPLACE,
        updateRequest
    )
}
```

Code Program 99 Override fun `publish`, merubah draft menjadi posts

```
val updateBuilder : OneTimeWorkRequest.Builder = OneTimeWorkRequestBuilder<UpdateDataWorker>()

updateBuilder.setInputData( // kirims data ke worker
    Data.Builder()
        .putString(KEY_UUID, uuid)
        .putString(KEY_TITLE, title)
        .putString(KEY_VALUE, value)
        .putString(KEY_MOOD, mood).build()
)
updateBuilder.setConstraints(constraint.build())

val updateRequest : OneTimeWorkRequest = updateBuilder.build()

workManager.enqueueUniqueWork(
    uuid,
    ExistingWorkPolicy.APPEND_OR_REPLACE,
    updateRequest
)
```

Code Program 100 Override suspend fun update

```
override suspend fun deletePost(uuid: String) {

    val deleteBuilder : OneTimeWorkRequest.Builder = OneTimeWorkRequestBuilder<DeleteDataWorker>()

    deleteBuilder.setInputData(Data.Builder().putString(KEY_UUID, uuid).build())
    deleteBuilder.setConstraints(constraint.build())

    val deleteRequest : OneTimeWorkRequest = deleteBuilder.build()

    workManager.enqueueUniqueWork(
        uuid,
        ExistingWorkPolicy.APPEND_OR_REPLACE,
        deleteRequest
    )
}
```

Code Program 101 Override suspend fun delete post

```
new *  
override suspend fun deleteFavorite(uuid: String) : Unit = dao.delete(uuid)
```

Code Program 102 deleteFavorite

```
override suspend fun insertDraft(postDto: PostDto) {  
    val insertBuilder : OneTimeWorkRequest.Builder = OneTimeWorkRequestBuilder<InsertDraftWorker>()  
    insertBuilder.setInputData(postDto.toDataWorker())  
    insertBuilder.setConstraints(constraint.build())  
  
    val insertRequest : OneTimeWorkRequest = insertBuilder.build()  
  
    workManager.enqueueUniqueWork(  
        postDto.uuid,  
        ExistingWorkPolicy.APPEND_OR_REPLACE,  
        insertRequest  
    )  
}
```

Code Program 103 insertDraft

```
override suspend fun deleteAllPost() {  
    val deleteBuilder : OneTimeWorkRequest.Builder = OneTimeWorkRequestBuilder<DeleteAllDataWorker>()  
    deleteBuilder.setConstraints(constraint.build())  
  
    val deleteRequest : OneTimeWorkRequest = deleteBuilder.build()  
  
    workManager.enqueueUniqueWork(  
        uniqueWorkName + "${LocalUser.username} ${LocalUser.uuid}",  
        ExistingWorkPolicy.APPEND_OR_REPLACE,  
        deleteRequest  
    )  
}
```

Code Program 104 Override suspend fun delete all post

```
override suspend fun deleteAllDraft() {
    val deleteBuilder = OneTimeWorkRequestBuilder<DeleteAllDraftWorker>()
    deleteBuilder.setConstraints(constraint.build())

    val deleteRequest: OneTimeWorkRequest = deleteBuilder.build()
    workManager.enqueueUniqueWork(
        uniqueWorkName = "${LocalUser.username}${LocalUser.uuid}",
        ExistingWorkPolicy.APPEND_OR_REPLACE,
        deleteRequest
    )
}
```

Code Program 105 deleteAllDraft

```
new *
override fun deleteFavoriteAll() {CoroutineScope(Dispatchers.IO).launch{dao.clear()} }
```

Code Program 106 deleteFavoriteAll

Kalau kalian tidak membuat terlebih dahulu pada class interface *diary repository*, kalian tidak akan menjumpainya pada *diaryRepoImpl*.

```
copy:
suspend fun unsubscribe()

new *
fun favorite(): Flow<List<Post>>

new *
suspend fun draft(): Result<Flow<List<Post>>>

@sync
suspend fun fetch(): Result<Flow<List<Post>>>
@sync
suspend fun update(uid: String, title: String, value: String, mood: String)
new *
suspend fun publish(uuid: String, publish: Boolean)

new *
suspend fun insertPost(postDto: PostDto)
new *
```

```
suspend fun insertDraft(postDto: PostDto)
new ✘
suspend fun insertFavorite(postD|o: PostDto)

new ✘
suspend fun deletePost(uuid: String)
new ✘
suspend fun deleteFavorite(uuid: String)

new ✘
suspend fun search(query: String): Flow<List<Post>>

new ✘
suspend fun deleteAllPost()

new ✘
suspend fun deleteAllDraft()

new ✘
fun deleteFavoriteAll()
```

Code Program 107 Diary repository

Untuk Network.checkConnectivity, itu saya buat sendiri di file helper. Fungsi untuk memeriksa apakah pada perangkat pengguna terdapat koneksi internet.

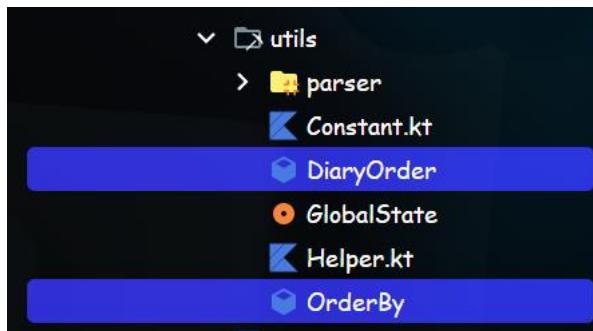
```
@SuppressLint("NewApi")
object Network {
    private const val NETWORK_STATUS_NOT_CONNECTED = 0
    private const val NETWORK_STATUS_WIFI = 1
    private const val NETWORK_STATUS_MOBILE = 2
    private const val TYPE_WIFI = 1
    private const val TYPE_MOBILE = 2
    private const val TYPE_NOT_CONNECTED = 0
    private fun connectivityStatus(context: Context): Int {
        val connectivityManager: ConnectivityManager =
            context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
        val activeNetwork: NetworkInfo? = connectivityManager.activeNetworkInfo
        if (null != activeNetwork) {
            if (activeNetwork.type == ConnectivityManager.TYPE_WIFI) return TYPE_WIFI
            if (activeNetwork.type == ConnectivityManager.TYPE_MOBILE) return TYPE_MOBILE
        }
        return TYPE_NOT_CONNECTED
    }
}
```

Code Program 108 Object Network

Setelah itu kita akan membuat *fun* untuk CRUD local *database*, kita juga akan menyatukan *data* yang didapat secara remote dan *data* local.

5.6 Membuat Sort & Order by

Kita perlu membuat Sealed Class DiaryOrder dan Sealed Class OrderBy. Mengapa demikian ?. Nantinya, akan ada fitur urutkan sesuai tanggal, judul, atau mood. Fitur tersebut juga bisa diurutkan apakah *ascending* (naik) atau *descending* (turun).



Code Program 109 Membuat sealed class baru



Code Program 110 Sealed Class Diary Order



Code Program 111 Sealed Class OrderBy

Setelah itu, kita buat dahulu *diary use case*.

```
class DiaryUseCase @Inject constructor(
    private val repository: DiaryRepository,
    private val context: Context,
) {
    new *
    suspend fun draft(): Flow<ResponseState<List<Post>>> = flow { thisFlowCollector<ResponseState<List<Post>>>
        if (Network.checkConnectivity(context)) {
            emit(ResponseState.Loading)
            try {
                repository.draft().onSuccess { flow { FlowListPost() ->
                    emitAll(
                        flow.map { it in ListCom()
                            ResponseState.Success(it)
                        }
                    )
                }
            } catch (e: Throwable) {
                emit(ResponseState.Error(e.message.toString()))
                Log.d("AAAAAAA4", "Fetch ${e.message}")
            }
        } catch (e: Exception) {
            emit(ResponseState.Error(e.message.toString()))
            Log.d("AAAAAAA3", "Fetch ${e.message}")
        }
    } else {
        emit(ResponseState.Error("Make sure you have connection", emptyList()))
    }
}

suspend fun [sortByOrder: DiaryOrder = DiaryOrder.Date, orderBy: OrderBy = OrderBy.Descending]: Flow<ResponseState<List<Post>>> = flow { thisFlowCollector<ResponseState<List<Post>>>
    if (Network.checkConnectivity(context)) {
        emit(ResponseState.Loading)
        try {
            repository.fetchC().onSuccess { flow { FlowListPost() ->
                val date = sortByOrder == DiaryOrder.Date ? flow.map { post: Post ->
                    post.date
                } : emptyFlow()
                val mode = orderBy == OrderBy.Ascending ? DiaryOrder.Mode.Ascending : DiaryOrder.Mode.Descending
                val title = orderBy == OrderBy.Ascending ? DiaryOrder.Title.Ascending : DiaryOrder.Title.Descending
                posts.sortedBy { it.createdAt }
                    .map { post -> post.copy(createdAt = date.invoke(post)) }
                    .sortedWith(compareBy(mode, title))
            }
        }
    }
}
```

```
        OrderByDescending -> {
            when (diaryOrder) {
                DiaryOrder.Date -> posts.sortedByDescending { it.createdAt }  
                DiaryOrder.Mood -> posts.sortedByDescending { it.mood.lowercase() }  
                DiaryOrder.Title -> posts.sortedByDescending { it.title.lowercase() }  
            }
        }
    }
}

emitter.on("data", (data) => {
    data.map((it) => {
        ResponseState.Success(it)
    })
})

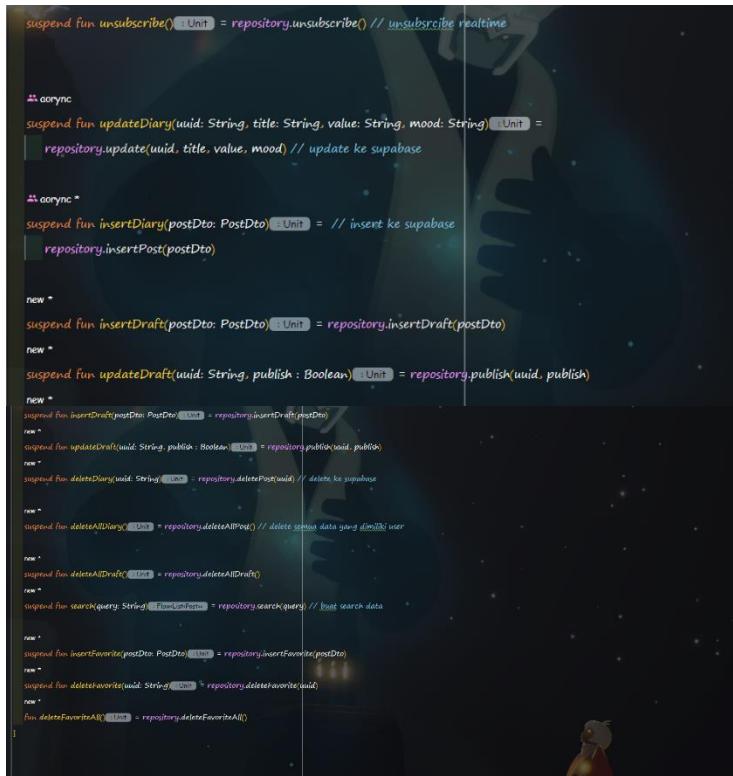
onFailure((t: Throwable) => {
    emit(ResponseState.Error(t.message.toString()))
    Log.d("tag", "AAAAAAA*", msg = "fetch ${t.message}")
})

catch (e: Exception) => {
    emit(ResponseState.Error(e.message.toString()))
    Log.d("tag", "AAAAAAA*", msg = "fetch ${e.message}")
}

} else {
    emit(ResponseState.Error("Make sure you have connection"))
}

fun getFavourite(): DiaryOrder = DiaryOrder.Date.orderBy.OrderByDescending

): Flow<List<Post>> = repository.favorite().map { posts: List<Post> ->
    when (orderBy) {
        OrderByAscending -> {
            posts.sortedBy { it.createdAt }  
        }
        OrderByDescending -> {
            posts.sortedByDescending { it.createdAt }  
        }
        DiaryOrder.Mood -> posts.sortedBy { it.mood.lowercase() }  
        DiaryOrder.Title -> posts.sortedBy { it.title.lowercase() }  
    }
}
}
```



```
    suspend fun unsubscribe(): Unit = repository.unsubscribe() // unsubscribe realtime

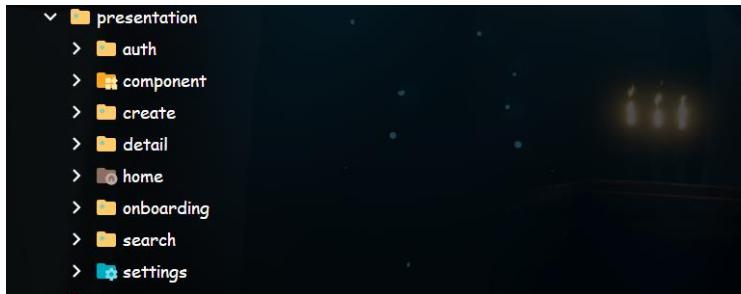
    @OptIn(ExperimentalCoroutinesApi::class)
    suspend fun updateDiary(uuid: String, title: String, value: String, mood: String): Unit =
        repository.update(uuid, title, value, mood) // update ke supabase

    @OptIn(ExperimentalCoroutinesApi::class)
    suspend fun insertDiary(postDto: PostDto): Unit = // insert ke supabase
        repository.insertPost(postDto)

    new *
    suspend fun insertDraft(postDto: PostDto): Unit = repository.insertDraft(postDto)
    new *
    suspend fun updateDraft(uuid: String, publish: Boolean): Unit = repository.publish(uuid, publish)
    new *
    suspend fun insertDraft(postDto: PostDto): Unit = repository.insertDraft(postDto)
    new *
    suspend fun updateDraft(uuid: String, publish: Boolean): Unit = repository.publish(uuid, publish)
    new *
    suspend fun deleteDiary(uuid: String): Unit = repository.deletePost(uuid) // delete ke supabase
    new *
    suspend fun deleteAllDiary(): Unit = repository.deleteAllPost() // delete semua data yang dimiliki user
    new *
    suspend fun deleteAllDraft(): Unit = repository.deleteAllDraft()
    new *
    suspend fun search(query: String): Flow<List<Post>> = repository.searchQuery() // fungsi search data
    new *
    suspend fun insertFavorite(postDto: PostDto): Unit = repository.insertFavorite(postDto)
    new *
    suspend fun deleteFavorite(uuid: String): Unit = repository.deleteFavorite(uuid)
    new *
    fun deleteFavoriteAll(): Unit = repository.deleteFavoriteAll()
```

Code Program 112 Diary use case

Kalau selesai, kita akan lanjut ke viewModel. Buatlah package berikut di dalam package presentation. Isilah masing-masing package dengan Screen dan ViewModel (kecuali onBoarding).



Code Program 113 Package-package pada presentation

Kita akan memulai dari home terlebih dahulu. Buatlah HomeViewModel menjadi seperti ini.

```
fun subscribe(  
    diaryOrder: DiaryOrder = DiaryOrder.Date, orderBy: OrderBy = OrderBy.Descending,  
) {  
   getPost?.cancel()  
getFavorite?.cancel()  
getDraft?.cancel()  
CoroutinesScope(Dispatchers.IO).launch { this.launchingScope // berjalan di latar belakang  
    _getPost = launch { postState.emitAll(diaryUC.fetch(diaryOrder, orderBy)) }  
  
getFavorite = diaryUC.getFavourite(diaryOrder, orderBy).onEach { it.listPosts  
    state.value = state.value.copy(  
        favourite = it  
    )  
}  
}.launchIn(scope: this)  
  
getDraft = launch { _draftState.emitAll(diaryUC.draft()) }  
unsubscribe()  
}
```

The code is a Kotlin function named 'subscribe'. It takes two parameters: 'diaryOrder' (defaulting to 'DiaryOrder.Date') and 'orderBy' (defaulting to 'OrderBy.Descending'). Inside the function, three cancellable jobs are canceled: '_getPost', '_getFavorite', and '_getDraft'. Then, a new job is launched on the IO dispatcher using 'CoroutinesScope'. This job starts another job in the background ('this.launchingScope') which emits all posts from the 'fetch' method of the 'diaryUC' object. The 'fetch' method takes 'diaryOrder' and 'orderBy' as parameters. Finally, a 'getFavorite' job is started, which emits favorite posts from the 'getFavourite' method of 'diaryUC'. The 'getFavourite' method also takes 'diaryOrder' and 'orderBy' as parameters and emits them into the 'listPosts' field of the 'state' object. The 'state' object is updated with a copy of itself where the 'favourite' field is set to the value from the emitted 'it' variable. A 'launchIn' block is used to start a new job in the 'scope' provided. The function concludes with an 'unsubscribe' call.

```
fun unsubscribe() {
    viewModelScope.launch (thisCoroutineScope)
        diaryUC.unsubscribe()
    }
}

now
fun refresh(){
    viewModelScope.launch (thisCoroutineScope)
        subscribe()
    }
}

@composable
fun onSelect(sort: DiaryOrder, order: OrderBy) { // ketika ada perubahan dalam menyeleksi
    viewModelScope.launch (thisCoroutineScope)
        subscribe(sort, order)
}
}

fun insertFavorite(postDto: PostDto) {
    CoroutineScope(Dispatchers.IO).launch (thisCoroutineScope)
        diaryIC.insertFavorite(postDto)
}

now
fun publish(uuid: String, published: Boolean) {
    viewModelScope.launch (thisCoroutineScope)
        diaryICupdateDraft(uuid, published)
}
}

now
fun deleteAllDrafts() {
    viewModelScope.launch (thisCoroutineScope)
        diaryICdeleteAllDraft()
}
}

@composable
fun deletePost(uuid: String) {
    viewModelScope.launch (thisCoroutineScope)
        diaryICdeleted Diary(uuid)
}
}

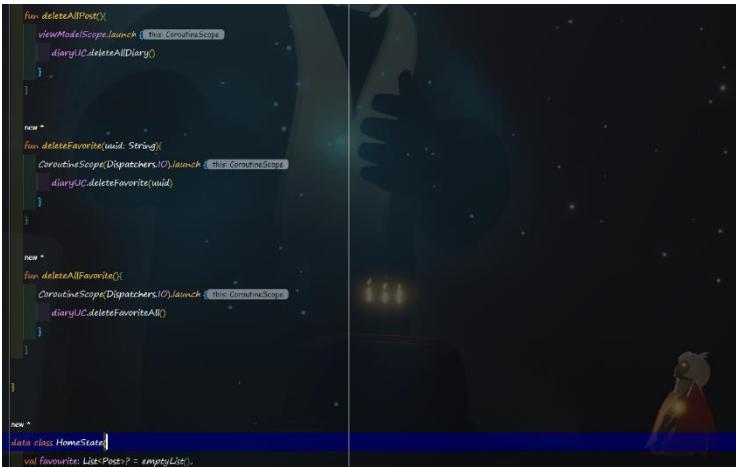
@HiltViewModel
class HomeViewModel @Inject constructor(
    globalUseCase: GlobalUseCase,
) : ViewModel() {
    private val diaryUC: DiaryUseCase = globalUseCase.diaryUseCase // Diary Use Case

    private val state: MutableState<HomeState> = mutableStateOf(HomeState())
    val state: State<HomeState> = state

    private val postState: MutableStateFlow<ResponseState<List<Post>>> = MutableStateFlow<ResponseState<List<Post>>>(ResponseState.Loading)
    val postState: State<ResponseState<List<Post>>> = postState.asStateFlow()

    private val draftState: MutableStateFlow<ResponseState<List<Post>>> = MutableStateFlow<ResponseState<List<Post>>>(ResponseState.Loading)
    val draftState: State<ResponseState<List<Post>>> = draftState.asStateFlow()

    private var getFavorite: Job? = null
    private var getPost: Job? = null
    private var getDraft: Job? = null
```

A screenshot of an Android Studio code editor. The file is named 'HomeViewModel.kt'. The code contains several 'new' sections, each defining a function that performs a delete operation using a CoroutineScope and Dispatchers.IO. The first section is 'fun deleteAllPost()'. The second section is 'fun deleteFavorite(uuid: String)'. The third section is 'fun deleteAllFavorite()'. The final section is 'data class HomeState'.

```
fun deleteAllPost(){
    viewModelScope.launch { thisCoroutineScope
        diaryUC.deleteAllDiary()
    }
}

new {
    fun deleteFavorite(uuid: String){
        CoroutineScope(Dispatchers.IO).launch { thisCoroutineScope
            diaryUC.deleteFavorite(uuid)
        }
    }
}

new {
    fun deleteAllFavorite(){
        CoroutineScope(Dispatchers.IO).launch { thisCoroutineScope
            diaryUC.deleteFavoriteAll()
        }
    }
}

}

new {
    data class HomeState(
        val favorite: List<Post>? = emptyList(),
    )
}
```

Code Program 114 HomeViewModel

Kalau sudah, lakukan juga dengan viewmodel yang lain.

```
@HiltViewModel
class EditorViewModel @Inject constructor(
    globalUseCase: GlobalUseCase,
    savedStateHandle: SavedStateHandle
) : ViewModel() {
    private val diaryUC: DiaryUseCase = globalUseCase.diaryUseCase

    private val _state: MutableStateFlow<ResponseState<Boolean>> = MutableStateFlow(ResponseState.Idle)

    val isEditorHandle: Boolean = savedStateHandle.navArgs<EditorScreenNavArgs>().edit // abaiakan terlebih dahulu
    val titleHandle: String = savedStateHandle.navArgs<EditorScreenNavArgs>().title // abaiakan terlebih dahulu
    val valueHandle: String = savedStateHandle.navArgs<EditorScreenNavArgs>().value // abaiakan terlebih dahulu
    val moodHandle: String = savedStateHandle.navArgs<EditorScreenNavArgs>().mood // abaiakan terlebih dahulu
    private val uidHandle: String = savedStateHandle.navArgs<EditorScreenNavArgs>().uid // abaiakan terlebih dahulu

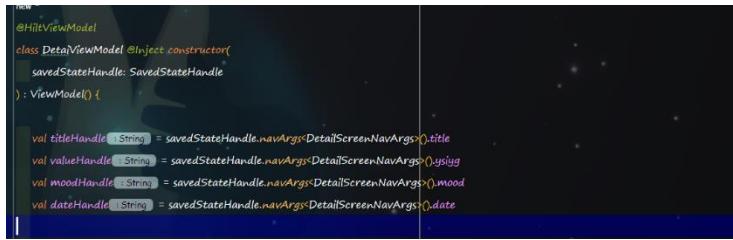
    val state: StateFlow<ResponseState<Boolean>> = _state.stateIn(
        viewModelScope,
        SharingStarted.WhileSubscribed(5000),
        ResponseState.Idle
    )

    fun saveCloud(postDto: PostDto) {
        viewModelScope.launch { diaryUC.insertDiary(postDto) }
    }

    fun saveDraft(postDto: PostDto) {
        viewModelScope.launch { diaryUC.insertDraft(postDto) }
    }

    async
fun updateCloud(title: String, value: String, mood: String) {
    viewModelScope.launch { diaryUC.updateDiary(uidHandle, title, value, mood) }
}
}
```

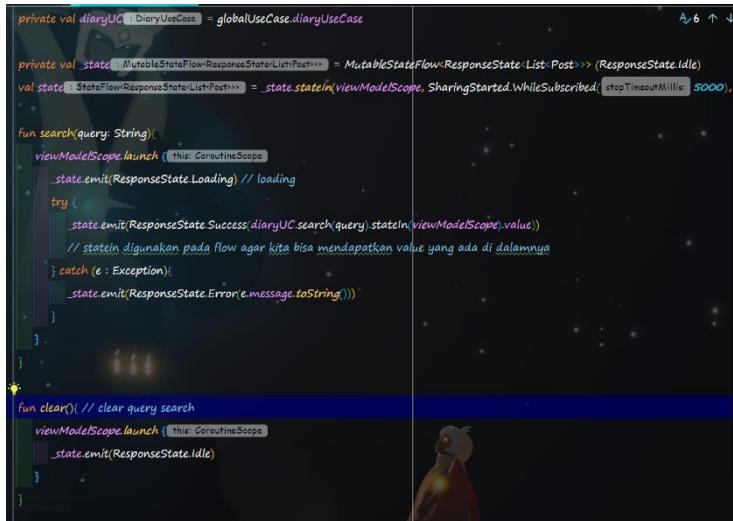
Code Program 115 CreateViewModel



```
new
@HiltViewModel
class DetailViewModel @Inject constructor(
    savedStateHandle: SavedStateHandle
) : ViewModel() {

    val titleHandle : String = savedStateHandle.navArgs<DetailScreenNavArgs>.title
    val valueHandle : String = savedStateHandle.navArgs<DetailScreenNavArgs>.value
    val moodHandle : String = savedStateHandle.navArgs<DetailScreenNavArgs>.mood
    val dateHandle : String = savedStateHandle.navArgs<DetailScreenNavArgs>.date
}
```

Code Program 116 DetailViewModel



```
private val diaryUC : DiaryUseCase = globalUseCase.diaryUseCase

private val state : MutableStateFlow<ResponseState<List<Post>>> = MutableStateFlow<ResponseState<List<Post>>>(ResponseState.Idle)
val state : StateFlow<ResponseState<List<Post>>> = state.stateIn(viewModelScope, SharingStarted.WhileSubscribed(5000), R

fun search(query: String) {
    viewModelScope.launch { thisCoroutineScope {
        state.emit(ResponseState.Loading) // loading
        try {
            state.emit(ResponseState.Success(diaryUC.search(query).stateIn(viewModelScope).value))
            // stateIn digunakan pada flow agar kita bisa mendapatkan value yang ada di dalamnya
        } catch (e: Exception) {
            state.emit(ResponseState.Error(e.message.toString()))
        }
    }
}
}

fun clear() // clear query search
viewModelScope.launch { thisCoroutineScope {
    state.emit(ResponseState.Idle)
}
}
```

Code Program 117 SearchViewModel

```
class SettingsViewModel @Inject constructor(
    globalUseCase: GlobalUseCase
) : ViewModel() {

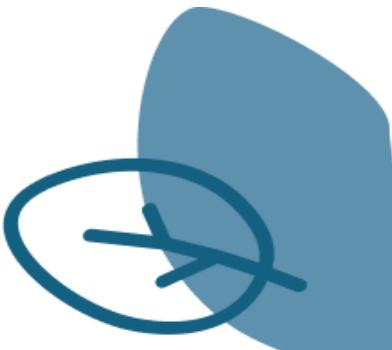
    private val userUC: AuthUseCase = globalUseCase.authUseCase
    private val diaryUC: DiaryUseCase = globalUseCase.diaryUseCase

    @Async
    fun update(name: String, email: String, password: String) {
        viewModelScope.launch { thisCoroutineScope
            userUC.update(name, email, password)
        }
    }

    @New
    fun clear() {
        viewModelScope.launch { thisCoroutineScope
            diaryUC.deleteFavoriteAll()
        }
    }
}
```

Code Program 118 SettingsViewModel

Dengan begini, maka selesailah tahapan dalam membuat logika. Selanjutnya kita akan mengimplementasikan User Interface kedalam screen dengan jetpack compose. Mari kita lanjutkan pada bab selanjutnya.



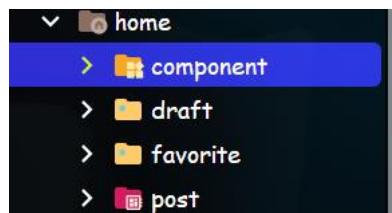
Bagian 6 :

Membuat Aplikasi (Screen)

6.1 HomeScreen

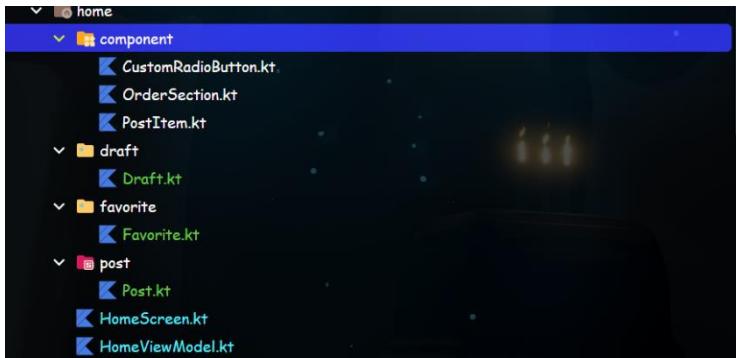
Akhirnya, kita akan membuat *screen* untuk ditampilkan kepada pengguna. Biasanya, ada component yang dibuat secara custom (dibuat sendiri) untuk ditampilkan pada *screen*.

Pada package home, kita akan membuat package component untuk menampung custom compose yang akan kita gunakan di HomeScreen.



Code Program 119 Package component

Lalu buatlah file-file berikut.



Code Program 120 File-file pada setiap package

Bagaimana cara membuat compose component?.

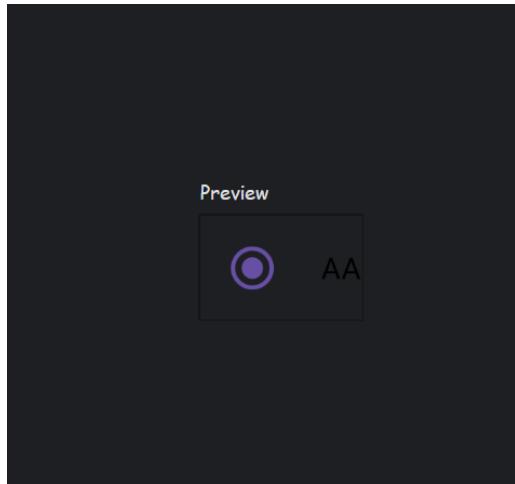
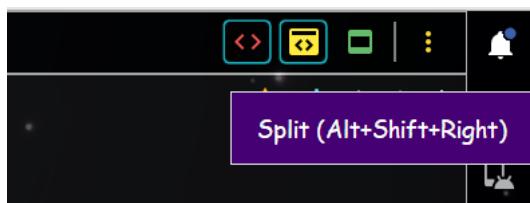
Cara seperti gambar berikut ini.

```
@Composable // pastikan pakai ini untuk menjadi komponen compose
fun CustomRadioButton(
    text: String, // text akan ditampilkan sebagai menu custom radio button
    selected: Boolean, // menggunakan boolean untuk memberi kondisi apakah di pilih?
    onSelect: () -> Unit, // perubahan pada kondisi select
    modifier: Modifier = Modifier // untuk mengatur komponen compose
) {
    Row(
        modifier = modifier,
        verticalAlignment = Alignment.CenterVertically // alignment child berada di tengah secara vertikal
    ) {
        RadioButton(
            selected = selected,
            onClick = onSelect,
            colors = RadioButtonDefaults.colors() // pewarnaan
                .selectedColor(MaterialTheme.colorScheme.primary,
                .unselectedColor = MaterialTheme.colorScheme.onBackground
            )
        )
        Spacer(modifier = Modifier.width(8.dp))
        Text(text = text, style = MaterialTheme.typography.bodyMedium) // ukuran text
    }
}
```

Code Program 121 CustomRadioButton

Kita bisa melihat preview dari composable yang dibuat. Tapi jika kalian memiliki parameter yang ada di *fun compose*, kalian harus mengisinya terlebih dahulu. Disarankan ketika membuat sebuah component, jangan memasukkan parameter terlebih dahulu pada *fun* jika kita ingin melihat preview.

```
@Preview // kalau ingin melihat preview  
@Composable  
fun Preview(){  
    CustomRadioButton(text = "AA", selected = true, onSelect = { })  
}
```



Code Program 122 Menggunakan preview

Kalau kalian ingin membuat komponen sendiri akan lebih baik, karena seterusnya saya hanya memberi contoh dasar. Ayo buat komponen lain untuk HomeScreen.

```
@Composable
fun PostItem(
    title : String,
    date : String,
    mood : String,
    modifier: Modifier = Modifier
) {
    Card(
        modifier = modifier
            .padding(8.dp),
        colors = CardDefaults.cardColors(
            containerColor = MaterialTheme.colorScheme.primary,
            contentColor = MaterialTheme.colorScheme.surface
    ),
)
```

```
content = { this: ColumnScope }  
Column(  
    modifier = Modifier  
        .padding(8.dp)  
        .fillMaxWidth()  
) { this: ColumnScope }  
Row(  
    modifier = Modifier  
        .fillMaxWidth(),  
    verticalAlignment = Alignment.CenterVertically,  
    horizontalArrangement = Arrangement.SpaceBetween  
) { this: RowScope }  
Text(  
    text = title,  
    style = MaterialTheme.typography.titleMedium,  
)  
  
Text(text = mood)  
}  
Spacer(modifier = Modifier.height(8.dp))  
Text(  
    text = date  
)
```

Code Program 123 PostItem

```
@Composable
fun OrderSection(
    onOrderChange: (DiaryOrder, OrderBy) -> Unit,
    modifier: Modifier = Modifier // kalau kita membuat custom component dan bukan screen, lebih baik menyediakan Modifier seperti ini
) {
    var selected: DiaryOrder by remember {
        // menggunakan remember dan mutableState agar kondisi dapat berubah dan tersimpan selama screen dibuka
        mutableStateOf(diaryOrder)
    }

    var ordered: OrderBy by remember {
        mutableStateOf(order)
    }
}
```

```
Column( // column itu kebawah atau vertical
    modifier = modifier
) { this: ColumnScope
    Row( // row itu kesamping atau horizontal
        modifier = Modifier.fillMaxWidth()
    ) { this: RowScope
        CustomRadioButton(
            text = "Title",
            selected = selected is Title,
            onSelect = {
                selected = Title
                onOrderChange(selected, order)
            }
        )
        Spacer(modifier = Modifier.width(8.dp))
        CustomRadioButton( // menggunakan Custom radio button yang telah dibuat
            text = "Date",
            selected = selected is Date,
            onSelect = {
                selected = Date
                onOrderChange(selected, order)
            }
        )
    }
}
```

```
Spacer(modifier = Modifier.width(8.dp)) // berfungsi sebagai jarak // width itu lebar & height itu tinggi
CustomRadioButton(
    text = "Mood",
    selected = selected is Mood,
    onSelect = {
        selected = Mood
        onOrderChange(selected, order)
    }
)
}
Spacer(modifier = Modifier.width(26.dp))
Row(
    modifier = Modifier.fillMaxWidth()
) { this: RowScope
    CustomRadioButton(
        text = "Ascending",
        selected = ordered is Ascending,
        onSelect = {
            ordered = Ascending
            onOrderChange(selected, ordered)
        }
    )
}
```

```
)
}
Spacer(modifier = Modifier.width(8.dp))
CustomRadioButton(
    text = "Descending",
    selected = ordered is Descending,
    onSelect = {
        ordered = Descending
        onOrderChange(selected, ordered)
    }
)
```

Code Program 124 OrderSection

```
@Composable
fun SaveDialog(
    onDismissRequest: () -> Unit,
    onConfirmation: () -> Unit,
    text: String = "do you want to do this action?"
) {
    AlertDialog(
        onDismissRequest = onDismissRequest,
        confirmButton = {
            TextButton(onClick = onConfirmation) { this: RowScope -
                Text(text = "Sure")
            }
        },
        dismissButton = {
            TextButton(onClick = onDismissRequest) { this: RowScope -
                Text(text = "Eh, cancel!")
            }
        },
        text = {
            Text(text = text)
        }
    )
}
```

Code Program 125 SaveDialog

Kita buat juga file-file lainnya.

```
@OptIn(ExperimentalFoundationApi::class, ExperimentalMaterial3Api::class)
@Composable
fun DraftContent(
    data: ResponseState<List<Post>>,
    lazyState: LazyListState,
    flingBehavior: FlingBehavior,
    delete: (uid: String) -> Unit,
    publish: (uid: String, published: Boolean) -> Unit,
    deleteAll: () -> Unit,
) {
    var expanded: Boolean by remember { mutableStateOf(value = false) }
    var isDelete: Boolean by remember {
        mutableStateOf(value = false)
    }

    var selected: String by remember {
        mutableStateOf(value = "")
    }

    var isPublish: Boolean by remember {
        mutableStateOf(value = false)
    }

    var isDeleteAll: Boolean by remember {
        mutableStateOf(value = false)
    }

    var uid: String by remember { // uid yang bisa diubah datanya, digunakan untuk dialog
        mutableStateOf(value = "") }
    var publishState: Boolean by remember {
        mutableStateOf(value = false)
    }
}
```

```
data.DisplayResult(  
    onLoading = {  
        Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) { this.BoxScope  
            | LinearProgressIndicator()  
        }  
    },  
    onSuccess = { data : List ->  
        if (data.isNotEmpty()) {  
            Column(  
                modifier = Modifier.fillMaxSize()  
            ) { this.ColumnScope  
                TextButton(onClick = { isDeleteAll = true }) { this.RowScope  
                    Text(text = "Delete All")  
                }  
                Spacer(modifier = Modifier.height(16.dp))  
                LazyColumn(  
                    state = lazyState,  
                    flingBehavior = FlingBehavior.  
                    content = { this.LazyListScope  
                        items(data) { this.ListItemScope post ->  
                            val menu: List = listOf  
                                Menu(text = "publish") {  
                                    uid = post.uid  
                                    publishState = post.toPostDto?.published  
                                    isPublish = true  
                                },  
                                PublishState = post.toPostDto?.published  
                                isPublish = true  
                            };  
                            Menu(  
                                text = "delete"  
                            ) {  
                                uid = post.uid  
                                isDelete = true  
                            }  
                        }  
                    }  
                    ExpandedDropdownMenuBox(  
                        // dropdown  
                        expanded = expanded, // memanggil boolean, apakah dropdown di tampilkan  
                        onExpandedChange = { it.value } // ketika kondisi berubah, mengembalikan boolean untuk expanded  
                        expanded = expanded  
                    ),  
                ) { this.ExposedDropdownMenuScope  
                    PostItem(  
                        title = post.title,  
                        date = post.createdAt.parseToDate().formatToString(),  
                        mood = post.mood,  
                        modifier = Modifier  
                            .menuAnchor()  
                            .combinedClickable(  
                                onClick = {  
                                    isPublish = true  
                                }  
                            )  
                    )  
                }  
            }  
        }  
    }  
)
```

```
        onLongClick = {
            selected = post.uuid
            expanded = true
        }
    }

    AnimatedVisibility(visible = selected === post.uuid) {
        ExposedDropdownMenu(
            expanded = expanded,
            onDismissRequest = { expanded = false }) {
            menus.forEach { menu : Menu -> // setup value dari list tema yang dibuat
                DropdownMenuItem( // dropdown item
                    text = { Text(text = menu.text) }, // nama tema
                    onClick = { // ketika di klik
                        menu.onClick()
                    }
                )
            }
        }
    }
}

else {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        this.BoxScope {
            Text(text = "You have nothing... ::_::")
        }
    }
}

},
onError = { it: String } {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        this.BoxScope {
            Text(text = "You have nothing... ::_::")
        }
    }
}
}
```

```
AnimatedVisibility(visible = isPublish) { this: AnimatedVisibilityScope
    SaveDialog(
        text = "Do you want delete this ?",
        onDismissRequest = { isPublish = false },
        onConfirmation = {
            publish(uuid, publishState)
            isPublish = false
        }
    )
}

AnimatedVisibility(visible = isDelete) { this: AnimatedVisibilityScope
    SaveDialog(
        text = "Do you want delete this ?",
        onDismissRequest = { isDelete = false },
        onConfirmation = {
            delete(uuid)
            isDelete = false
        }
    )
}

AnimatedVisibility(visible = isDeleteAll) { this: AnimatedVisibilityScope
    SaveDialog(
        text = "Do you want delete all data ?",
        onDismissRequest = { isDeleteAll = false },
        onConfirmation = {
            deleteAll()
            isDeleteAll = false
        }
    )
}
```

Code Program 126 Draft.kt

```
@OptIn(ExperimentalFoundationApi::class, ExperimentalMaterial3Api::class)
@Composable
fun FavoriteContent(
    data: List<Post>,
    lazyState: LazyListState,
    flingBehavior: FlingBehavior,
    delete: (String) -> Unit,
    deleteAll: () -> Unit,
    toDetail: (String, value: String, mood: String, date: String) -> Unit,
) {
    var expanded: Boolean by remember { mutableStateOf(false) }
    var isDelete: Boolean by remember {
        mutableStateOf(value = false)
    }

    var selected: String by remember {
        mutableStateOf("")
    }

    var isDeleteAll: Boolean by remember {
        mutableStateOf(value = false)
    }

    var uuid: String by remember { // uid yang bisa diambil datanya, digunakan untuk dialog
        mutableStateOf("") }
}


```

```
if (data.isNotEmpty()) {
    Column(modifier = Modifier.fillMaxSize()) {
        LazyColumn(
            state = lazyState,
            flingBehavior = FlingBehavior,
            content = {
                items(data) { itemScope(post: Post) ->
                    val menus: List = listOf(
                        Menu(text = "delete") {
                            uuid = post.uuid
                            isDelete = true
                        },
                        Menu(text = "detail") {
                            toDetail(
                                post.title,
                                post.value,
                                post.mood,
                                post.createdAt.parseToDate().formatToString()
                            )
                        }
                    )
                }
            }
        )
    }
}
```

A 19 1

```
ExposedDropdownMenuItemBox
    // dropdown
    expanded = expanded, // memanggil boolean, apakah dropdown di tampilkan
    onExpandedChange = { [int Boolean] // ketika kondisi berubah, mengembalikan boolean untuk expanded
        expanded = !expanded
    }
)
| this: ExposedDropdownMenuItemBoxScope
PostItem()
    title = post.title,
    date = post.createdAt.parseToDate().formatToString(),
    mood = post.mood,
    modifier = Modifier
        .menuAnchor()
        .combinedClickable(
            onClick = {
                toDetail(
                    post.title,
                    post.value,
                    post.mood,
                    post.createdAt.toString(),
                    parseToDate(date),
                    formatToString()
                )
            }
        )
        .onLongClick = (
            selected = post.uid
            expanded = true
        )
    )
)
)

AnimatedVisibility(visible = selected === post.uid) ( this: AnimatedVisibilityScope
ExposedDropdownMenu(
    expanded = expanded,
    onDismissRequest = { expanded = false } // this.ColumnScope
)
menus.forEach { menu |> // setiap value dari list tema yang dibuat
    DropdownMenuItem // dropdown item
        text = [ Text(text = menu.text) ], // nama tema
        onClick = { // ketika di klik
            menu.onClick()
            expanded = false
        }
)
)
```

```
    }
    TextButton(onClick = { isDeleteAll = true }) { this: RowScope
        Text(text = "delete All")
    }
    Spacer(modifier = Modifier.height(16.dp))
}

AnimatedVisibility(visible = isDelete) { this: AnimatedVisibilityScope
    SaveDialog(
        text = "Do you want delete this ?",
        onDismissRequest = { isDelete = false },
        onConfirmation = {
            delete(item)
            isDelete = false
        }
    )
}

AnimatedVisibility(visible = isDeleteAll) { this: AnimatedVisibilityScope
    SaveDialog(
        text = "Do you want delete all data ?",
        onDismissRequest = { isDeleteAll = false },
        onConfirmation = {
            deleteAll()
            isDeleteAll = false
        }
    )
} else {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) { this: BoxScope
        Text(text = "You have nothing.. ::>_<::")
    }
}
}
```



Code Program 127 Favorite.kt

```
@OptIn(ExperimentalFoundationApi::class, ExperimentalMaterial3Api::class)
@Composable
fun PostContent(
    data: ResponseState<List<Post>>,
    lazyState: LazyListState,
    flingBehavior: FlingBehavior,
    delete: (uuid: String) -> Unit,
    favorite: (Post) -> Unit,
    deleteAll: () -> Unit,
    toEdit: (title: String, value: String, mood: String, uuid: String) -> Unit,
    toDetail: (title: String, value: String, mood: String, date: String) -> Unit,
) {
    var expanded: Boolean by remember { mutableStateOf(true) }
    var isDelete: Boolean by remember {
        mutableStateOf(false)
    }

    var selected: String by remember {
        mutableStateOf("")
    }

    var isDeleteAll: Boolean by remember {
        mutableStateOf(false)
    }

    var uuid: String by remember { // uuid yang bisa diambil datanya, digunakan untuk dialog
        mutableStateOf("")
    }

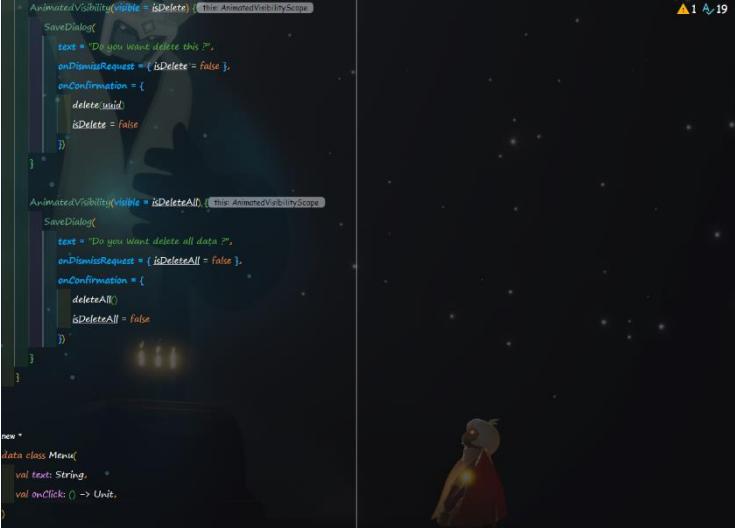
    data.DisplayResult(
        onLoading = { Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
            LinearProgressIndicator()
        } },
        onSuccess = { data: List<Post> ->
            if (data.isNotEmpty()) {
                Column {
                    LazyColumn(
                        state = lazyState,
                        flingBehavior = FlingBehavior,
                        content = {
                            LazyListScope {
                                items(data) { this.LazyItemScope(post: Post) ->
                                    val menus: List<MenuItem> = listOf(
                                        MenuItem(text = "edit") {
                                            toEdit(
                                                post.title,
                                                post.value,
                                                post.mood,
                                                post.uuid
                                            )
                                        }
                                    )
                                }
                            }
                        }
                    )
                }
            }
        }
    )
}
```

```
        },
        Menu text: "favorite") {
            favorite.post;
        },
        Menu text: "delete") {
            uid = post.uid;
            bDelete = true;
        },
        Menu text: "detail") {
            toDetail(
                post.title,
                post.value,
                post.mood,
                post.createdAt.parseToDate().formatToString()
            )
        }
    )
}

ExposedDropdownMenuBox(
    // dropdown
    expanded = expanded, // memanggil boolean, apakah dropdown di tampilkan
    onExpandedChange = [|| Boolean] // ketika kondisi berubah, mengembalikan boolean untuk expanded
    expanded = expanded
),
this.ExposedDropdownMenuScope,
PostItem(
    title = post.title,
    date = post.createdAt.parseToDate().formatToString(),
    mood = post.mood,
    modifier = Modifier
        .menuAnchor()
        .combinedClickable(
            onClick = {
                toDetail(
                    post.title,
                    post.value,
                    post.mood,
                    post.createdAt.parseToDate().formatToString()
                )
            }
        )
)
```

```
        onLongClick = {
            selected = post.uid
            expanded = true
        }
    )
}

AnimatedVisibility(visible = selected === post.uid) [ this AnimatedVisibilityScope ]
ExposedDropdownMenu(
    expanded = expanded,
    onDismissRequest = { expanded = false } [ this ColumnScope ]
) menus.forEach { menu [ Item ] -> // setting value dari list tema yang diubah
    DropdownMenuItem( // dropdown item
        text = { Text(text = menu.text) } // nama tema
        onClick = { // ketika di klik
            menu.onClick()
        }
        expanded = false
    )
}
} else {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) [ this BoxScope ]
    Text(text = "You have nothing... ::>::")
}
},
onError = { (it: String)
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) [ this BoxScope ]
    Text(text = "You have nothing... ::>::")
}
}
```

A screenshot of a mobile application interface. At the top, there is a navigation bar with icons for back, forward, and search. Below the navigation bar, there is a large, semi-transparent dark overlay. In the center of this overlay, there is a white rectangular dialog box. The dialog box contains the following text:

SaveDialog(
 text = "Do you want to delete this?",
 onDismissRequest = { isDelete = false },
 onConfirmation = {
 deleteUnit()
 isDelete = false
 }
)

SaveDialog(
 text = "Do you want to delete all data?",
 onDismissRequest = { isDeleteAll = false },
 onConfirmation = {
 deleteAll()
 isDeleteAll = false
 }
)

new *
data class Menu(
 val text: String,
 val onClick: () -> Unit,
)

Code Program 128 Post.kt

Kalau kita menggunakan @Destination pada screen, pastikan kita telah mem-build ulang program dengan menekan tombol palu. Karena @Destination perlu membuat Object dari screen yang kita buat.

Code Program 129 HomeScreen

6.2 EditorScreen

Seperti biasa, kita buat dulu komponennya. Komponen pertama cukup panjang, karena membutuhkan komponen lain.



```
@Composable
fun DropDownItem(
    text: String,
    isSelected: Boolean,
    onItemSelected: () -> Unit
) {
    DropdownMenuItem(
        text = {
            Text(text = text)
        }, onClick = onItemSelected,
        modifier = Modifier.background(
            color = if (isSelected) {
                Color.Gray.copy(alpha = 0.2f)
            } else {
                Color.Transparent
            }, shape = RoundedCornerShape(6.dp)
        )
    )
}
```

```
@Composable  
fun TitleStyleButton(  
    value: RichEditorState  
) {  
    var expanded : Boolean by remember { mutableStateOf(value == false) }  
  
    val onItemSelected : (TextSpanStyle) -> Unit = { style: TextSpanStyle ->  
        value.updateStyle(style)  
        expanded = false  
    }  
  
    Row { this: RowScope  
        IconButton(  
            modifier = Modifier  
                .padding(2.dp)  
                .size(48.dp),  
            onClick = { expanded = true },  
        ) {  
            Row { this: RowScope  
                Icon(  
                    imageVector = Icons.Rounded.Title, contentDescription = null,  
                    modifier = Modifier.size(24.dp)  
                )  
            }  
        }  
    }  
}
```

```
Row { this: RowScope
    icon(
        imageVector = Icons.Rounded.Title, contentDescription = null,
        modifier = Modifier.size(24.dp)
    )
    icon(Icons.Default.ArrowDropDown, contentDescription = null)
}
}

DropdownMenu(
    expanded = expanded,
    onDismissRequest = { expanded = false },
    modifier = Modifier.wrapContentSize(),
    properties = PopupProperties(focusable = false)
) { this: ColumnScope

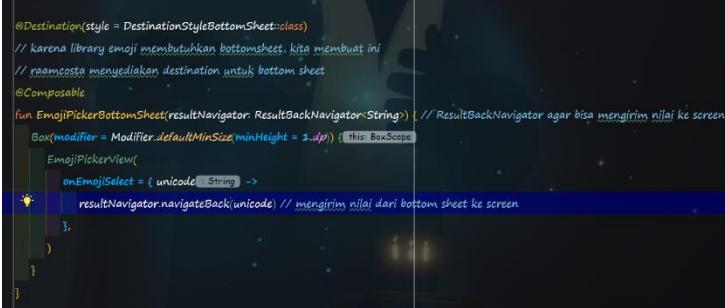
    DropdownItem(text = "Text",
        isSelected = value.hasStyle(TextSpanStyle.Default),
        onItemSelected = { onItemSelected(TextSpanStyle.Default) })
    DropdownItem(text = "Header 1", isSelected = value.hasStyle(TextSpanStyle.H1Style),
        onItemSelected = { onItemSelected(TextSpanStyle.H1Style) })
    DropdownItem(text = "Header 2", isSelected = value.hasStyle(TextSpanStyle.H2Style),
        onItemSelected = { onItemSelected(TextSpanStyle.H2Style) })
    DropdownItem(text = "Header 3", isSelected = value.hasStyle(TextSpanStyle.H3Style),
        onItemSelected = { onItemSelected(TextSpanStyle.H3Style) })
    DropdownItem(text = "Header 4", isSelected = value.hasStyle(TextSpanStyle.H4Style),
        onItemSelected = { onItemSelected(TextSpanStyle.H4Style) })
}
```

```
DropdownMenu(  
    expanded = expanded,  
    onDismissRequest = { expanded = false },  
    modifier = Modifier.wrapContentSize(),  
    properties = PopupProperties(focusable = false)  
) { this: ColumnScope  
  
    DropDownItem(text = "Text",  
        isSelected = value.hasStyle(TextSpanStyle.Default),  
        onItemSelected = { onItemSelected(TextSpanStyle.Default) })  
    DropDownItem(text = "Header 1", isSelected = value.hasStyle(TextSpanStyle.H1Style),  
        onItemSelected = { onItemSelected(TextSpanStyle.H1Style) })  
    DropDownItem(text = "Header 2", isSelected = value.hasStyle(TextSpanStyle.H2Style),  
        onItemSelected = { onItemSelected(TextSpanStyle.H2Style) })  
    DropDownItem(text = "Header 3", isSelected = value.hasStyle(TextSpanStyle.H3Style),  
        onItemSelected = { onItemSelected(TextSpanStyle.H3Style) })  
    DropDownItem(text = "Header 4", isSelected = value.hasStyle(TextSpanStyle.H4Style),  
        onItemSelected = { onItemSelected(TextSpanStyle.H4Style) })  
    DropDownItem(text = "Header 5", isSelected = value.hasStyle(TextSpanStyle.H5Style),  
        onItemSelected = { onItemSelected(TextSpanStyle.H5Style) })  
    DropDownItem(text = "Header 6", isSelected = value.hasStyle(TextSpanStyle.H6Style),  
        onItemSelected = { onItemSelected(TextSpanStyle.H6Style) })
```

```
fun StyleContainer(  
    state: RichEditorState, // memerlukan RichEditorState dari library RichEditor  
    onSave: () -> Unit // ketika save data  
) {  
    Row(  
        Modifier  
            .fillMaxWidth()  
            .height(52.dp)  
            .padding(horizontal = 10.dp),  
        horizontalArrangement = Arrangement.Start,  
    ) {  
        this: RowScope  
  
        TitleStyleButton(state)  
        StyleButton(  
            icon = Icons.Rounded.FormatBold,  
            style = TextSpanStyle.BoldStyle,  
            value = state  
        )  
  
        StyleButton(  
            icon = Icons.Rounded.FormatItalic,  
            style = TextSpanStyle.ItalicStyle,  
            value = state,  
        )  
    }  
}
```

```
    StyleButton(  
        icon = Icons.Rounded.FormatItalic,  
        style = TextSpanStyle.ItalicStyle,  
        value = state,  
    )  
  
    StyleButton(  
        icon = Icons.Rounded.FormatUnderlined,  
        style = TextSpanStyle.UnderlineStyle,  
        value = state,  
    )  
  
    IconButton(  
        modifier = Modifier  
            .padding(2.dp)  
            .size(48.dp),  
        onClick = onSave,  
    ) {  
        Icon(  
            Icons.Rounded.Check, contentDescription = null,  
            modifier = Modifier.size(24.dp)  
        )  
    }  
}
```

Code Program 130 File EditorControls



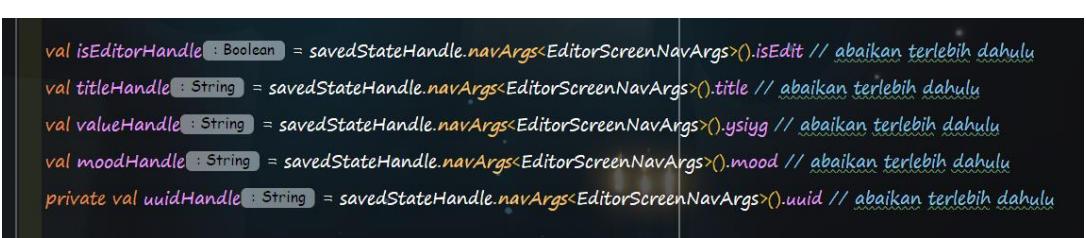
```
@Destination(style = DestinationStyleBottomSheet::class)
// karena library emoji membutuhkan bottomsheet, kita membuat ini
// karena coba menyediakan destination untuk bottom sheet
@Composable
fun EmojiPickerBottomSheet(resultNavigator: ResultBackNavigator<String>) { // ResultBackNavigator agar bisa mengirim nilai ke screen
    Box(modifier = Modifier.defaultMinSize(minHeight = 3.dp)) {
        this: BoxScope
        EmojipickerView(
            onEmojiSelect = { unicode: String ->
                resultNavigator.navigateBack(unicode) // mengirim nilai dari bottom sheet ke screen
            },
        )
    }
}
```

Code Program 131 EmojiPickerBottomSheet

Saatnya membuat EditorScreen. EditorScreen akan menampilkan *input title*, editor view dan sebuah *button* untuk memilih emoji. Sebelumnya, kita tambahkan dulu hal berikut ke viewmodel.

savedStateHandle: SavedStateHandle

Code Program 132 Tambahkan pada parameter



```
val isEditorHandle : Boolean = savedStateHandle.navArgs<EditorScreenNavArgs>().isEdit // abaikan terlebih dahulu
val titleHandle : String = savedStateHandle.navArgs<EditorScreenNavArgs>().title // abaikan terlebih dahulu
val valueHandle : String = savedStateHandle.navArgs<EditorScreenNavArgs>().ysiig // abaikan terlebih dahulu
val moodHandle : String = savedStateHandle.navArgs<EditorScreenNavArgs>().mood // abaikan terlebih dahulu
private val uuidHandle : String = savedStateHandle.navArgs<EditorScreenNavArgs>().uuid // abaikan terlebih dahulu
```

Code Program 133 Menerima data yang dikirim dari screen lain

```
@Composable
fun EditorScreen(
    navigator: DestinationsNavigator,
    resultRecipient: ResultRecipient<EmojiPickerBottomSheetDestination, String>,
    viewModel: EditorViewModel = hiltViewModel(),
) {
    val context: Context = LocalContext.current
    var titleState: String by remember {
        mutableStateOf(viewModel.titleHandle)
    }

    var selectedEmoji: String by remember {
        mutableStateOf(viewModel.moodHandle)
    }

    var showSaveDialog: Boolean by remember { mutableStateOf(value = false) }

    data class EditorScreenNavArgs // agar bisa menerima kiriman data
        val isEdit: Boolean,
        val uid: String,
        val title: String,
        val ysiyg: String,
        val mood: String,
    )

    @Destination(
        navArgsDelegate = EditorScreenNavArgs::class
    )
}
```

```
Scalfold { it: PaddingValues
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
            .padding(bottom = it.calculateBottomPadding())
            .padding(top = it.calculateTopPadding()),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) { this: ColumnScope

        IconButton(
            onClick = { navigator.navigateUp() /* kembali ke halaman sebelumnya */ },
            modifier = Modifier.align(Alignment.Start)
        ) {
            Icon(imageVector = Icons.Rounded.ArrowBackIosNew, contentDescription = null)
        }
    }
}

resultRecipient.onNavResultListener = { navResult: NavResult<String> -> // menerima nilai yang dikirim dari bottom sheet
    when (navResult) {
        is NavResult.Canceled -> {} // Toast.makeText(context, "Bottom sheet canceled", Toast.LENGTH_SHORT).show()
        is NavResult.Value -> {
            selectedEmoji = navResult.value
        }
    }
}

val newstate: RichEditorState = remember { // state untuk rich editor, berbasis JSON
    val input: String = viewModel.valueHandle
    RichEditorState.Builder()
        .setInput(input)
        .adapter(JsonEditorParser())
        .build()
    remember { newstate }
}
```

```
Row(
    verticalAlignment = Alignment.CenterVertically
) { this: RowScope
    TextField(
        modifier = Modifier.weight(1f),
        value = titleState,
        onValueChange = { value: String ->
            titleState = value
        },
        colors = TextFieldDefaults.colors(
            focusedIndicatorColor = Color.Transparent,
            unfocusedIndicatorColor = Color.Transparent,
            errorIndicatorColor = Color.Transparent,
            focusedContainerColor = Color.Transparent,
            unfocusedContainerColor = Color.Transparent,
            errorContainerColor = Color.Transparent
        ),
        label = {
            Text(text = "Title Here")
        },
        maxLines = 1
    )
    Spacer(modifier = Modifier.width(8.dp))
    TextButton(onClick = { navigator.navigate(EmojiPickerBottomSheetDestination) }) {
        Text(text = selectedEmoji)
    }
}

StyleContainer(
    state = newState,
    onSave = {
        showSaveDialog = true
    }
)
RichEditor(
    state = newState,
    modifier = Modifier
        .fillMaxWidth()
        .weight(1f)
        .border(1.dp, Color.Gray)
        .padding(5.dp)
        .background(Color.White)
)
```

```
AnimatedVisibility(visible = showSaveDialog) { this.AnimatedVisibilityScope
    SaveDialog(
        onDismissRequest = { showSaveDialog = false },
        onConfirmation = {
            val postDto = PostDto(
                createdAt = "now",
                uuid = "${LocalUser.username}-${UUID.randomUUID()}" /* memakai uuid agar safe, karena uuid tidak mudah ditebak */,
                userId = LocalUser.uuid,
                title = titleState,
                value = newState.output(),
                mood = selectedEmoji
            )
            if (titleState.isNotEmpty() && newState.output().isNotEmpty() && selectedEmoji.isNotEmpty()) {
                if (viewModel.isEditorHandle) {
                    viewModel.updateCloud(titleState, newState.output(), selectedEmoji)
                } else {
                    viewModel.saveCloud(postDto)
                }
                showSaveDialog = false
                navigator.navigateUp()
            } else {
                if (titleState.isNotEmpty() && newState.output().isNotEmpty() && selectedEmoji.isNotEmpty()) {
                    if (viewModel.isEditorHandle) {
                        viewModel.updateCloud(titleState, newState.output(), selectedEmoji)
                    } else {
                        viewModel.saveCloud(postDto)
                    }
                    showSaveDialog = false
                    navigator.navigateUp()
                } else {
                    Toast.makeText(context, text = "These blank (*>_<*)", Toast.LENGTH_SHORT).show()
                    showSaveDialog = false
                }
            }
        }
    )
}
```

Code Program 134 EditorScreen

Logika *input* kita masukkan ke dalam dialog. Mengapa begitu?. Mungkin saja pengguna tidak sengaja menekan tombol save padahal belum selesai mengetik. Untuk itulah kita menggunakan dialog, menanyakan kepada pengguna apakah benar ingin menyimpan *data*.

Maaf, ternyata kita melewatkkan *screen* untuk *login* dan *register*. Selanjutnya kita akan buat terlebih dahulu, baru kemudian kita lanjutkan ke *SearchScreen*.

6.3 LoginScreen

Dalam *LoginScreen*, terdapat dua *input text*, tombol save dan tombol beralih ke *register*. Input pertama digunakan untuk *email* dan input kedua digunakan untuk *password*. Input *password* memiliki *button* untuk melihat *password* (visible) karena biasanya *password* diubah menjadi titik-titik.

Sebelum kita membuat *screen*, kita memerlukan ekstensi untuk memeriksa apakah format *email* valid dan apakah format *password* valid. Tambahkan codingan berikut ke *Helper.kt*.

```
fun String.isValidEmail() : Boolean = !Patterns.EMAIL_ADDRESS.matcher(input: this).matches()
```

Code Program 135 Fun String.isValidEmail

```
fun String.isMixedCase() : Boolean = any(Char::isLowerCase) && any(Char::isUpperCase)
fun String.hasSpecialChar() : Boolean = any { it in "!,+^<" }

val requirements : List<XFunction1<String, Boolean>> = listOf(String::isMixedCase, String::hasSpecialChar)

val String.meetsRequirements : Boolean get() = requirements.all { check : XFunction1<String, Boolean> -> check(this) }
```

Code Program 136 Ekstensi untuk memeriksa password

Ketika *email* atau *password* tidak sesuai dengan format yang ditentukan, maka akan menampilkan *error*. Baik dari warna dan ditampilkannya support *text*.

```
@Destination
@Composable
fun LoginScreen(
    navigator: DestinationsNavigator,
    viewModel: LoginViewModel = hiltViewModel(),
) {
    val snackBarHostState = SnackbarHostState() // state snackbar

    val email : State<String> = viewModel.email.collectAsState(initial = "") // mendapatkan value email view model
    val password : State<String> = viewModel.password.collectAsState( initial: "") // value password dari viewmodel

    var showPassword : Boolean by remember { mutableStateOf( value: false) } // kondisi visible password
    var isPasswordError : Boolean by remember { mutableStateOf( value: false) } // apakah password error ?
    var isEmailError : Boolean by remember { mutableStateOf( value: false) } // apakah email error ?
    var passwordErrorMsg : String by remember { mutableStateOf( value: "") } // text yang ditampilkan ketika passworderror

    val loginState : State<ResponseState<Boolean>> = viewModel.loginState.collectAsStateWithLifecycle()
    val context : Context = LocalContext.current // context
```

```
Scaffold(  
    snackBarHost = {  
        SnackBarHost(hostState = snackBarHostState) // snackbar  
    }  
    ) { it: PaddingValues  
    Column(  
        modifier = Modifier  
            .padding(it)  
            .padding(horizontal = 16.dp)  
            .fillMaxSize(),  
        verticalArrangement = Arrangement.Center,  
        horizontalAlignment = Alignment.CenterHorizontally  
    ) { this: ColumnScope  
        Spacer(modifier = Modifier.height(64.dp))  
        Text(  
            text = "Login",  
            style = MaterialTheme.typography.titleLarge,  
            fontWeight = FontWeight.Medium  
        )  
  
        //  
        email  
        Spacer(modifier = Modifier.height(32.dp))  
        //  
        email  
        Spacer(modifier = Modifier.height(32.dp))  
        TextField( // input text view  
            value = email.value, // set value  
            onValueChange = { value: String // ketika value berubah, mengembalikan string  
                isEmailError = value.isValidEmail()  
                viewModel.onEmailChange(value)  
            },  
            label = { // label input text  
                Text(text = "Email")  
            },  
            singleLine = true,  
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),  
            // agar keyboard menyesuaikan dengan input text email  
            isError = isEmailError, // set error  
            supportingText = { // menampilkan text ketika error  
                if (isEmailError) {  
                    Text(  
                        modifier = Modifier.fillMaxWidth(),  
                        text = "Email address is not valid",  
                        color = MaterialTheme.colorScheme.error  
                    )  
                }  
            }  
        )  
    }  
}
```

```

        },
        trailingIcon = { // ikon di belakang atau sebelah kanan
            icon(
                imageVector = Icons.Rounded.Email,
                contentDescription = "email trail icon"
            )
        },
        colors = TextFieldDefaults.colors(
            focusedIndicatorColor = Color.Transparent,
            unfocusedIndicatorColor = Color.Transparent,
            errorIndicatorColor = Color.Transparent
        ),
        modifier = Modifier.fillMaxWidth(),
        shape = CircleShape
    )
)

// password
Spacer(modifier = Modifier.height(16.dp))
TextField(
    value = password.value,
    onValueChange = { value : String ->
        if (value.length >= 8 && value.isNotEmpty()) {
            isPasswordError = value.meetsRequirements
        }
    }
)

// password
Spacer(modifier = Modifier.height(16.dp))
TextField(
    value = password.value,
    onValueChange = { value : String ->
        if (value.length >= 8 && value.isNotEmpty()) {
            isPasswordError = value.meetsRequirements
            if (!isPasswordError) {
                passwordErrorMsg = "Use letter, number, and unique character"
            } else {
                isPasswordError = true
                passwordErrorMsg = "Password must have at least 8 character"
            }
        }
    },
    viewModel.onPasswordChange(value)
),
label = (
    Text(text = "Password")
),
singleLine = true,
keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
visualTransformation = if (showPassword) VisualTransformation.None else PasswordVisualTransformation(),
// untuk melihat password
isError = isPasswordError,

```

```
        isError = isPasswordError,
        supportingText = {
            if (isPasswordError) {
                Text(
                    modifier = Modifier.fillMaxWidth(),
                    text = passwordErrorMsg,
                    color = MaterialTheme.colorScheme.error
                )
            }
        },
        trailingIcon = {
            val icon: ImageVector = if (showPassword)
                Icons.Rounded.LockOpen
            else Icons.Rounded.Lock

            IconButton(onClick = { showPassword = !showPassword }) {
                Icon(
                    imageVector = icon,
                    contentDescription = "Visibility"
                )
            }
        },
        colors = TextFieldDefaults.colors(
            focusedIndicatorColor = Color.Transparent,
            unfocusedIndicatorColor = Color.Transparent,
            errorIndicatorColor = Color.Transparent
        ),
        modifier = Modifier.fillMaxWidth(),
        shape = CircleShape
    )

    Spacer(modifier = Modifier.height(16.dp))


```

```
Button(
    onClick = {
        if (email.value.isNotEmpty() && password.value.isNotEmpty()) {
            // kalau email dan password tidak kosong
            viewModel.login() // login
            viewModel.onEmailChange(email) // set email menjadi kosong
            viewModel.onPasswordChange(password)
        } else { // kalau kosong
            Toast.makeText(context, text = "Fill the field first", Toast.LENGTH_SHORT).show()
        }
    }
)
```

```
        toast.makeText(context, "Fill the field first", Toast.LENGTH_SHORT).show()
    }
}

enabled = !loginState.value.isLoading, // bisa di klik kalau lagi tidak loading
modifier = Modifier
    .fillMaxWidth()
) { this: RowScope
    if (loginState.value.isLoading())
        CircularProgressIndicator()
    else
        Text(
            text = "Login",
            style = MaterialTheme.typography.bodyLarge
        )
}

loginState.value.DisplayResult(
    onSuccess = { it: Boolean
        navigator.navigate(HomeScreenDestination) { this: NavOptionsBuilder
            popUpTo(LoginScreenDestination.route) { this: PopUpToBuilder
                inclusive = true
            }
            launchSingleTop = true
        }
    },
    onError = { message: String ->
        Toast.makeText(LocalContext.current, message, Toast.LENGTH_SHORT).show()
    },
    onLoading = {}
)

Row(
    modifier = Modifier.wrapContentSize(),
    verticalAlignment = Alignment.CenterVertically
) { this: RowScope
    Text(text = "New Here (..v..)?")
    TextButton(onClick = {
        // berpindah ke register
        navigator.navigate(RegisterScreenDestination) { this: NavOptionsBuilder
            popUpTo(LoginScreenDestination.route) { this: PopUpToBuilder
                inclusive = true // tidak bisa kembali ke screen ini, jika menekan back akan menutup aplikasi
            }
            launchSingleTop = true
        }
    }) { this: RowScope
        Text(text = "Sign Up")
    }
}
```

Code Program 137 LoginScreen

Ketika pengguna berhasil *login*, secara otomatis pengguna diarahkan ke HomeScreen. Tentu saja pengguna tidak bisa kembali ke LoginScreen. Ketika *error*, pengguna harus mengisi ulang *email* dan *password*. Karena, *email* dan *password* telah diatur agar menjadi kosong ketika menekan tombol “*login*”.

6.4 RegisterScreen

RegisterScreen tidak jauh berbeda dengan LoginScreen. Bedanya, ada satu tambahan *input* untuk pengguna yaitu *input username*. Ketika pengguna belum memiliki akun, maka pengguna perlu untuk mendaftar melalui RegisterScreen.

```
val email : State<String> = viewModel.email.collectAsState(initial = "")  
// email yang masih kosong untuk di set ke input ext  
val password : State<String> = viewModel.password.collectAsState(initial: "")  
// password yang masih kosong  
val username : State<String> = viewModel.username.collectAsState(initial: "")  
// username yang masih kosong
```

Input *text* perlu nilai default untuk bisa di ubah. Karena itu initial *valuanya* masih kosong atau “”.

```
var showPassword : Boolean by remember { mutableStateOf(value: false) }
// tampilkan password ?

var isPasswordError : Boolean by remember { mutableStateOf(value: false) }
// apakah password error ?

var isEmailError : Boolean by remember { mutableStateOf(value: false) }
// apakah email error ?

var passwordErrorMsg : String by remember { mutableStateOf(value: "") }

// menampilkan text error untuk input password

val registerState : State<ResponseState<Boolean>> = viewModel.registerState.collectAsStateWithLifecycle()
// state yang berisi api wrapper (responseState)
```

Karena kita membuat *input* untuk *password*, kita perlu membuat agar *password* bisa dilihat. Kalau tidak, bagaimana pengguna bisa tahu kalau *password*nya sudah benar ?.

```
Scaffold(
    snackBarHost = {
        SnackbarHost(hostState = snackBarHostState)
    }
) { it: PaddingValues
    Column(
        modifier = Modifier
            .padding(it)
            .padding(horizontal = 16.dp)
            .fillMaxSize(), // fill max size = sepenuh layar
        verticalArrangement = Arrangement.Center, // ke tengah secara vertical
        horizontalAlignment = Alignment.CenterHorizontally // ke tengah secara horizontal
    ) { this: ColumnScope
        Spacer(modifier = Modifier.height(64.dp)) // jarak dengan tinggi
        Text(
            text = "Register",
            style = MaterialTheme.typography.titleLarge,
            fontWeight = FontWeight.Medium
        )
    }
}
```

```
Spacer(modifier = Modifier.height(32.dp))  
TextField(  
    value = email.value,  
    onValueChange = { value : String ->  
        isEmailError = value.isValidEmail() // memeriksa apakah email valid ?  
        viewModel.onEmailChange(value) // menyimpan perubahan value email  
    },  
    label = {  
        Text(text = "Email")  
    },  
    singleLine = true,  
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),  
    isError = isEmailError,
```

Kenapa `singleLine` diisi `true`? Karena tidak mungkin `input email, password, dan username` bisa diisi multiline. Value digunakan untuk mengatur nilai awal yang akan ditampilkan kepada pengguna. Kita mengurnya agar kosong terlebih dahulu. `IsError` digunakan agar kita bisa menampilkan `error` ketika kondisi tertentu, seperti ketika `email` atau `password` tidak valid.

Dulu sebelum compose digunakan, developer biasanya menggunakan margin untuk memberikan jarak antara komponen. Jetpack Compose tidak menyediakan margin pada Modifier, sehingga kita menggunakan Spacer untuk memberi jarak.

```
supportingText = {  
    if (isEmailError) {  
        Text(  
            modifier = Modifier.fillMaxWidth(),  
            text = "Email address is not valid",  
            color = MaterialTheme.colorScheme.error  
        )  
    }  
},  
  
trailingIcon = {  
    Icon(  
        imageVector = Icons.Rounded.Email,  
        contentDescription = "email trail icon"  
    )  
},  
  
colors = TextFieldDefaults.colors(  
    focusedIndicatorColor = Color.Transparent,  
    unfocusedIndicatorColor = Color.Transparent,  
    errorIndicatorColor = Color.Transparent  
,  
    modifier = Modifier.fillMaxWidth(),
```

Code Program 138 TextFieldEmail

```
onValueChange = { value : String } ->
    if (value.length >= 8 && value.isNotEmpty()) {
        isPasswordError = value.meetsRequirements // apakah password valid
        if (!isPasswordError)
            passwordErrorMsg = "Use letter, number, and unique character"
            // menyimpan text error
    } else {
        isPasswordError = true // mengatur bahwa error ditampilkan
        passwordErrorMsg = "Password must have at least 8 character"
    }
    viewModel.onPasswordChange(value)
}
```

Code Program 139 Memeriksa error pada TextField password

```
visualTransformation = if (showPassword) VisualTransformation.None else PasswordVisualTransformation(),
```

Code Program 140 Mengatur visibility untuk password

Visual Transformation digunakan untuk mengubah tampilan *text* yang muncul pada *textfield*. Apabila *showpassword = true*, *text* akan ditampilkan seperti *text* biasa. Apabila *showpassword = false*, *text* akan ditampilkan seperti *password* (titik-titik).

Untuk *textfield username* buat seperti *input* biasa. Tanpa diberi visibility. Tapi kita beri error apabila *input* kosong.

```
var isError : Boolean by remember {  
    mutableStateOf(value: false)  
}
```

Code Program 141 Var baru untuk menampilkan error pada textfield name

```
supportingText = {  
    if (isError) {  
        Text(  
            modifier = Modifier.fillMaxWidth(),  
            text = "Username is empty",  
            color = MaterialTheme.colorScheme.error  
        )  
    }  
},
```

Code Program 142 Mengatur text yang ditampilkan ketika error

Selanjutnya, kita buat *button* untuk *register* dan *text button* agar kita bisa beralih ke *login*. Ketika pengguna berhasil *register*, secara otomatis diarahkan ke *LoginScreen*/

```
Button(
    onClick = {
        if (email.value.isNotEmpty() && password.value.isNotEmpty() && username.value.isNotEmpty()) {
            viewModel.register()
        } else if (username.value.isNotEmpty()) {
            isError = true
        } else {
            Toast.makeText(context, "Make sure you fill all field", Toast.LENGTH_SHORT).show()
        }
    },
    enabled = !registerState.value.isLoading(),
    modifier = Modifier
        .fillMaxWidth()
)
```

Code Program 143 Memeriksa apakah semua textfield tidak kosong ?

Ketika semua *textfield* tidak kosong, pengguna bisa *register*. Ketika *username* kosong, *error* untuk *username* akan ditampilkan. Ketika semua field kosong, akan menampilkan *toast* untuk memberi tahu pengguna agar memeriksa semua *input*. Kita juga mengatur agar *button* bisa di klik ketika *state* tidak sedang *loading* pada *enabled*. Agar *button* memiliki *text*, gunakan *text* pada *rowScope* yang ada pada *button*.

```
RowScope {
    if (registerState.value.isLoading())
        CircularProgressIndicator()
    else
        Text(
            text = "Register",
            style = MaterialTheme.typography.bodyLarge
        )
}
```

Code Program 144 Text pada button

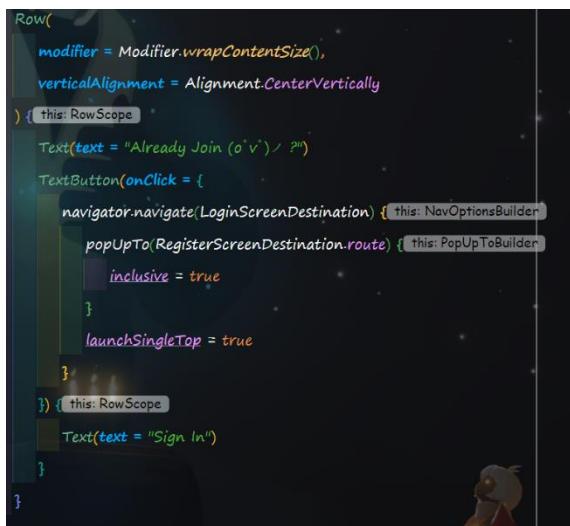
```
registerState.value.DisplayResult(  
    onSuccess = { it: Boolean  
        navigator.navigate(LoginScreenDestination) { this: NavOptionsBuilder  
            popUpTo(RegisterScreenDestination.route) { this.PopUpToBuilder  
                inclusive = true  
            }  
            launchSingleTop = true  
        }  
    }  
)
```

Code Program 145 Momen ketika success

Ketika berhasil *register*, pengguna akan diarahkan ke LoginScreen tanpa bisa kembali ke RegisterScreen. Ketika ada *error* pada proses *register*, *error* akan ditampilkan melalui *toast*.

```
onError = { message : String ->  
    Toast.makeText(LocalContext.current, message, Toast.LENGTH_SHORT).show()  
},
```

Code Program 146 Mengatur error pada *DisplayResult*



Code Program 147 Textbutton untuk berpindah ke LoginScreen

Kalau pengguna ingin berpindah ke LoginScreen, pengguna tinggal klik pada *text button*. Pengguna tidak bisa kembali ke RegisterScreen ketika sudah diarahkan ke LoginScreen.

Pasti codingan kalian *error*, lebih tepatnya yang ada di HomeScreen.

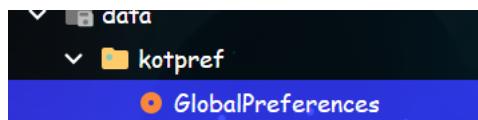


Code Program 148 Error pada HomeScreen

Kalau begitu, kita akan membuat SettingsScreen terlebih dahulu. Error ini terjadi karena kalian membuat Pilihan tema. Benar, kita akan membuat tema agar tema bisa diganti.

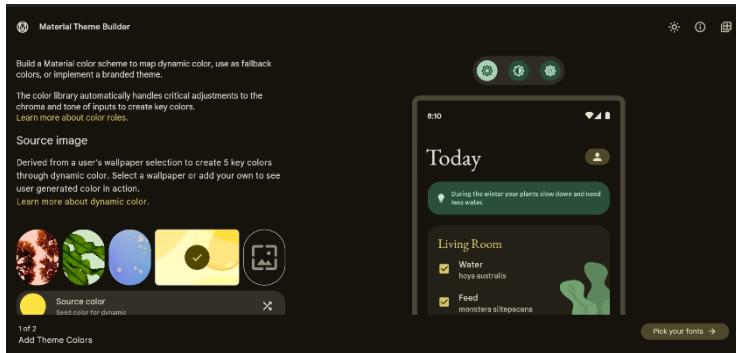
6.5 Theme

Pertama, buatlah Object bernama GlobalPreferences pada package kotpref di *data*.



Code Program 149 GlobalPreferences

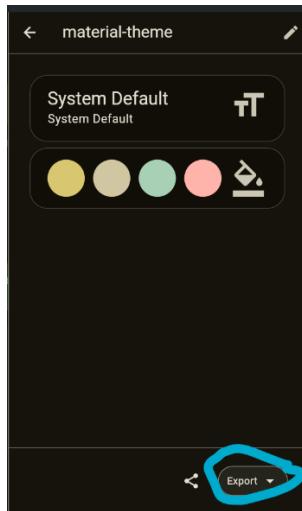
Kemudian, buatlah tema untuk compose pada laman website material design. Saya telah membuat 2 warna berbeda untuk ditampilkan. Pilih jetpack compose ketika eksport tema.



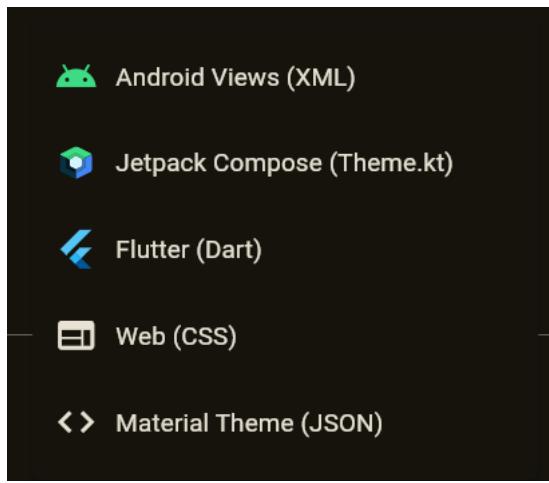
Gambar 34 <https://material-foundation.github.io/material-theme-builder>



Gambar 35 Untuk membuka button eksport

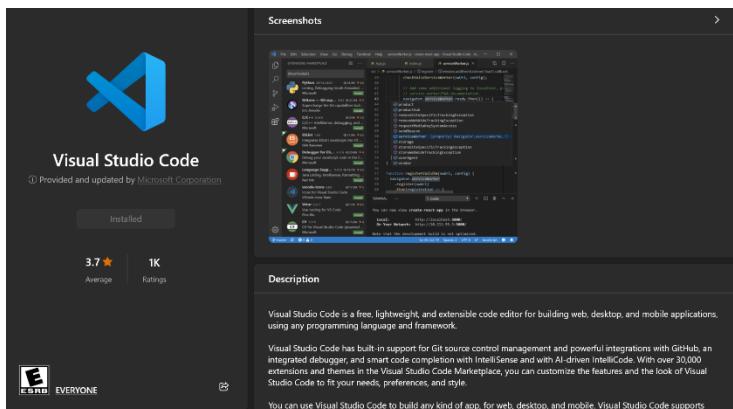


Gambar 36 Export



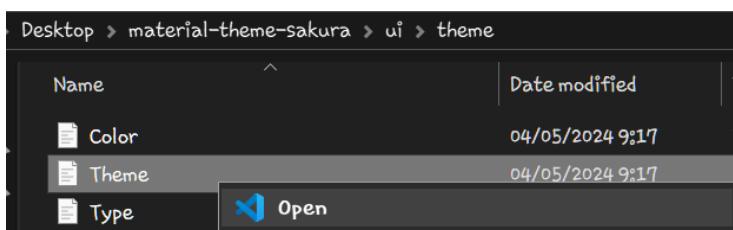
Gambar 37 Pilih Jetpack Compose

Pastikan kalian punya visual studio code. Untuk apa? Agar kita bisa meng-copy warna yang dibutuhkan. Tidak semua warna akan kita butuhkan.



Gambar 38 VS Code di microsoft store

Setelah itu, ekstrak tema yang telah kalian unduh dan buka file theme.kt dan color.kt dengan vs code.



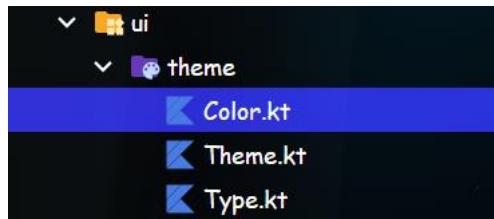
Gambar 39 Open with vsCode

Select susunan val color light dan dark saja. Abaikan susunan yang mengandung contrast.

```
3  val primaryLight = Color(0xFFFFEAA4D)
4  val onPrimaryLight = Color(0xFFFFFFFF)
5  val primaryContainerLight = Color(0xFFFFDADA)
6  val onPrimaryContainerLight = Color(0xFF3B000E)
7  val secondaryLight = Color(0xFF765E57)
8  val onSecondaryLight = Color(0xFFFFFFFF)
9  val secondaryContainerLight = Color(0xFFFFDADA)
10 val onSecondaryContainerLight = Color(0xFF2C1516)
11 val tertiaryLight = Color(0xFF755A2F)
12 val onTertiaryLight = Color(0xFFFFFFFF)
13 val tertiaryContainerLight = Color(0xFFFFDD80)
14 val onTertiaryContainerLight = Color(0xFF291880)
15 val errorLight = Color(0xFFFFBA1A1A)
16 val onErrorLight = Color(0xFFFFFFFF)
17 val errorContainerLight = Color(0xFFFFDDAD)
18 val onErrorContainerLight = Color(0xFF410002)
19 val backgroundLight = Color(0xFFFFFB77)
20 val onBackgroundLight = Color(0xFF222919)
21 val surfaceLight = Color(0xFFFFFB7F)
22 val onSurfaceLight = Color(0xFF221919)
23 val surfaceVariantLight = Color(0xFFFFF4D0D0)
24 val onSurfaceVariantLight = Color(0xFF524343)
25 val outlineLight = Color(0xFFB75373)
26 val outlineVariantLight = Color(0xFFD7C1C1)
27 val scrimLight = Color(0xFF000000)
28 val inverseSurfaceLight = Color(0xFF382E2E)
29 val inverseOnSurfaceLight = Color(0xFFFFFEDC)
30 val inversePrimaryLight = Color(0xFFFFB3B4)
31
```

Code Program 150 Susunan color Light

Kemudian kalian paste-kan pada color.kt di android studio. Jangan lupa untuk mengubah nama val color agar tidak bertabrakan.



Code Program 151 Color.kt

```
//val Mountain
//light
val primaryLightMountain : Color = Color( color: 0xFF34618D)
val onPrimaryLightMountain : Color = Color( color: 0xFFFFFFFF)
val primaryContainerLightMountain : Color = Color( color: 0xFFD9E4FF)
val onPrimaryContainerLightMountain : Color = Color( color: 0xFF001D35)
val secondaryLightMountain : Color = Color( color: 0xFFFF26070)
val onSecondaryLightMountain : Color = Color( color: 0xFFFFFFFF)
val secondaryContainerLightMountain : Color = Color( color: 0xFFD6E4F7)
val onSecondaryContainerLightMountain : Color = Color( color: 0xFF0F1D2A)
val tertiaryLightMountain : Color = Color( color: 0xFF6A5779)
val onTertiaryLightMountain : Color = Color( color: 0xFFFFFFFF)
val tertiaryContainerLightMountain : Color = Color( color: 0xFFFF1DAFF)
```

Code Program 152 Contoh, primaryLight diubah menjadi primaryLightMountain

Kemudian kita copy scheme light dan dark saja dari file theme.kt di vs code dan kita paste di theme.kt android studio. Pastikan kita mengubah nama valnya dan memanggil color sesuai dengan yang dibutuhkan pada parameter scheme.

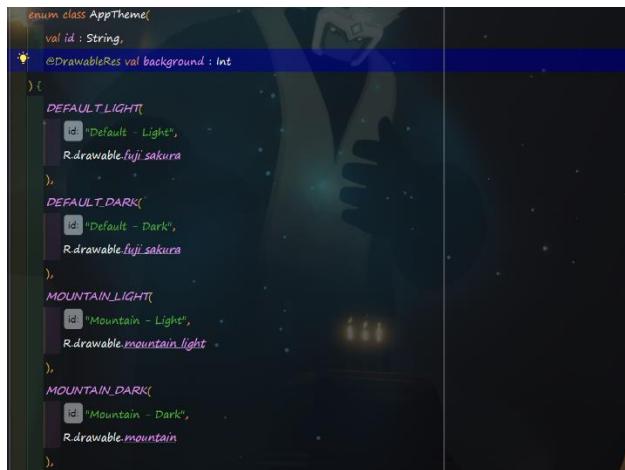
```
private val lightScheme = lightColorScheme(
    primary = primaryLight,
    onPrimary = onPrimaryLight,
    primaryContainer = primaryContainerLight,
    onPrimaryContainer = onPrimaryContainerLight,
    secondary = secondaryLight,
    onSecondary = onSecondaryLight,
    secondaryContainer = secondaryContainerLight,
    onSecondaryContainer = onSecondaryContainerLight,
    tertiary = tertiaryLight,
    onTertiary = onTertiaryLight,
    tertiaryContainer = tertiaryContainerLight,
    onTertiaryContainer = onTertiaryContainerLight,
    error = errorLight,
    onError = onErrorLight,
    errorContainer = errorContainerLight,
    onErrorContainer = onErrorContainerLight,
```

Code Program 153 lightScheme

```
//light
private val MountainLightScheme : ColorScheme = lightColorScheme(
    primary = primaryLightMountain,
    onPrimary = onPrimaryLightMountain,
    primaryContainer = primaryContainerLightMountain,
    onPrimaryContainer = onPrimaryContainerLightMountain,
    secondary = secondaryLightMountain,
    onSecondary = onSecondaryLightMountain,
    secondaryContainer = secondaryContainerLightMountain,
    onSecondaryContainer = onSecondaryContainerLightMountain,
    tertiary = tertiaryLightMountain,
    onTertiary = onTertiaryLightMountain,
```

*Code Program 154 Contoh, lightScheme diubah menjadi
MountainLightScheme*

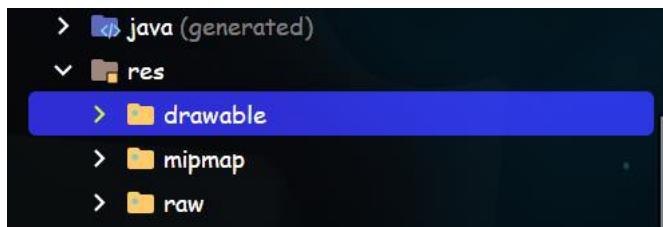
Setelah itu kita kembali ke GlobalPreferences untuk membuat Enum class theme. Tapi sebelum itu, silahkan kalian unduh gambar landscape karena kita akan menampilkan gambar yang sesuai dengan tema ketika dipilih.



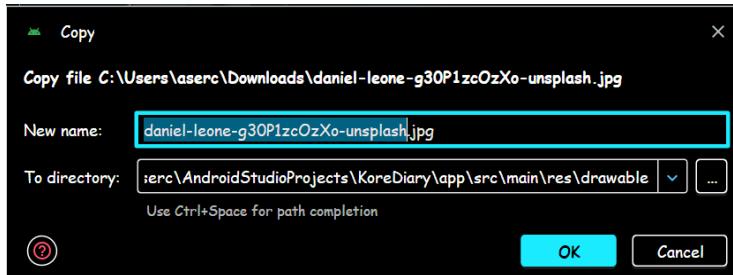
```
enum class AppTheme{
    val id : String,
    @DrawableRes val background : Int
}(
    DEFAULT_LIGHT(
        id: "Default - Light",
        R.drawable.fuji_sakura
    ),
    DEFAULT_DARK(
        id: "Default - Dark",
        R.drawable.fuji_sakura
    ),
    MOUNTAINLIGHT(
        id: "Mountain - Light",
        R.drawable.mountain_light
    ),
    MOUNTAIN_DARK(
        id: "Mountain - Dark",
        R.drawable.mountain
    ),
).
```

Code Program 155 Enum Class AppTheme

Id digunakan untuk menampilkan nama tema yang dipilih. @DrawableRes digunakan untuk membuat val yang dapat menampung gambar. Bagaimana cara menambahkan gambar ?. Kita klik copy pada gambar yang kita unduh di file explorer lalu kita paste pada folder drawable.

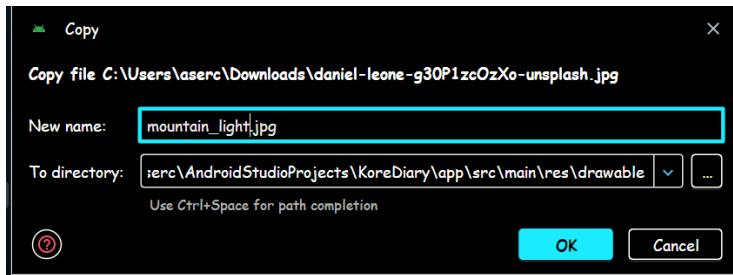


Code Program 156 Package drawable



Code Program 157 Menambahkan gambar ke drawable

Android studio sedikit ketat dalam penamaan file. Agar aman, gunakan *lowercase* (huruf kecil) dan “_”. Beri nama file dengan nama yang mudah diingat.



Code Program 158 Mengubah nama file menggunakan lowercase dan underscore

Kalau sudah, silahkan kalian buat enum didalam AppTheme yang telah kita buat. Pastikan kalian mengisi dengan id dan drawable.

```
enum class AppTheme {
    DEFAULT_LIGHT, // nama class baru, silahkan kalian beri nama
    id: "Default - Light", // karena perlu string, maka diisi dengan nama tema
    R.drawable.fuji_sakura // karena memerlukan gambar, seperti ini cara memanggilnya
},
```

Code Program 159 Contoh enum

Setelah itu, kita buat variable theme menggunakan kotpref dengan nilai defaultnya dari class enum yang dibuat. Disini saya menggunakan Tema default sebagai nilai defaultnya.

```
object GlobalPreferences : KotprefModel() {
    var theme: AppTheme by enumOrdinalPref(AppTheme.DEFAULT_DARK)

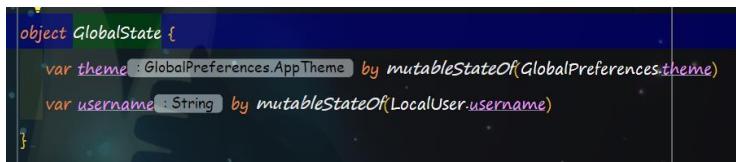
    override fun clear() {
        super.clear()
        theme = AppTheme.DEFAULT_DARK
    }
}
```

Code Program 160 GlobalPreferences

Kenapa ada *fun clear()* disana ?. Ketika *data* aplikasi dihapus, *data* yang disimpan melalui kotpref akan dihapus dan diubah nilainya menjadi default. Fun clear juga bisa dipanggil jika kita memanggil object yang menggunakan kotpref.

Setelah itu, kita buat sebuah object GlobalState. Karena jika kita langsung memanggil dari object kotpref, nilai

yang berubah tidak akan ditampilkan secara langsung. Maka dari itu kita buat var theme di GlobalState.



Code Program 161 GlobalState

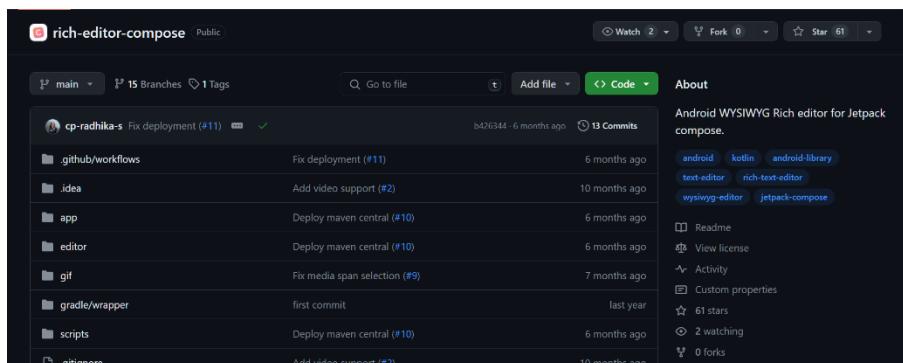
Kalian buat juga val untuk *username*, kita akan membutuhkannya di SettingScreen. Kenapa menggunakan *mutableStateOf*? Karena *mutableStateOf* tetap menyimpan perubahan *data* dan ditampilkan secara langsung.

Oh iya, mungkin kalian juga memiliki error pada CreateScreen (EditorScreen). Error itu terjadi karena kalian belum memiliki parser untuk editor Statenya.



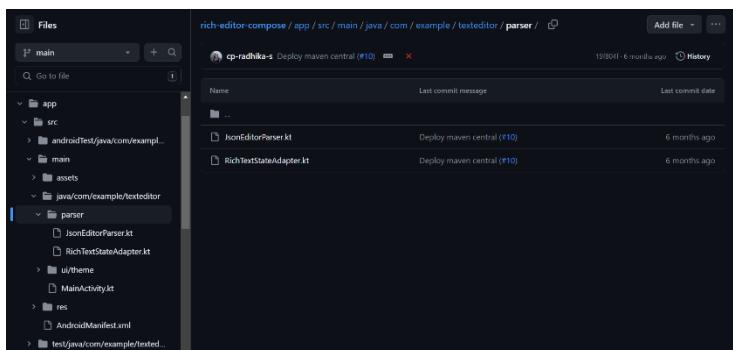
Code Program 162 State untuk rich editor

Silahkan kunjungi laman github canopas/rich-editor-compose.



Gambar 40 <https://github.com/canopas/rich-editor-compose>

Kemudian kalian buka app > src > main > java / com / example / texteditor > parser.

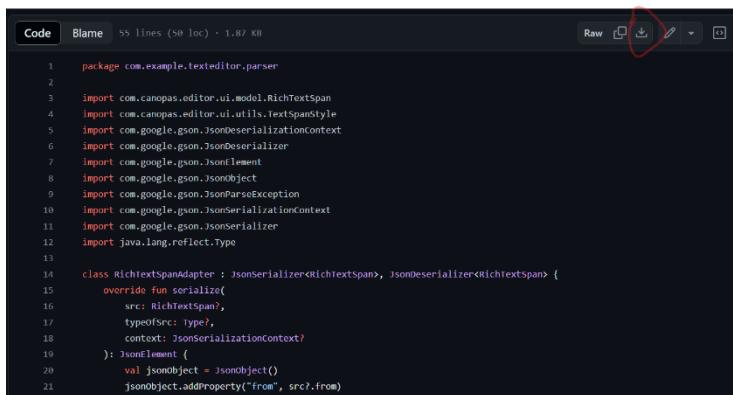


Gambar 41 Folder parser

Di sana kalian akan menjumpai dua file kotlin, JsonEditorParser.kt dan RichTextStateAdapter.kt. Kedua file inilah yang akan dibutuhkan di dalam folder parser.

Silahkan kalian klik masing-masing file tersebut dan mengunduhnya.

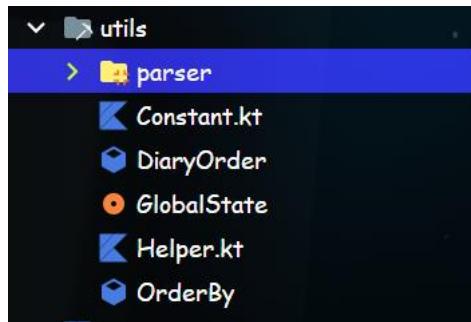
Gambar 42 JsonEditorParser



```
1 package com.example.texteditor.parser
2
3 import com.canopas.editor.ui.model.RichTextSpan
4 import com.canopas.editor.ui.utils.TextSpanStyle
5 import com.google.gson.JsonDeserializationContext
6 import com.google.gson.JsonDeserializer
7 import com.google.gson.JsonElement
8 import com.google.gson.JsonObject
9 import com.google.gson.JsonParseException
10 import com.google.gson.JsonSerializationContext
11 import com.google.gson.JsonSerializer
12 import java.lang.reflect.Type
13
14 class RichTextSpanAdapter : JsonSerializer<RichTextSpan>, JsonDeserializer<RichTextSpan> {
15     override fun serialize(
16         src: RichTextSpan,
17         typedSrc: Type,
18         context: JsonSerializationContext?
19     ): JsonElement {
20         val jsonObject = JsonObject()
21         jsonObject.addProperty("from", src.from)
22         ...
23     }
24
25     override fun deserialize(
26         json: JsonElement,
27         typeOfT: Type,
28         context: JsonDeserializationContext?
29     ): RichTextSpan {
30         ...
31     }
32 }
```

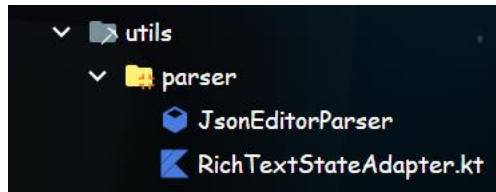
Gambar 43 RichTextSpanAdapter

Kalau sudah, kalian buat package baru untuk menampung kedua file bernama parser di dalam package utils.



Code Program 163 Package parser

Lalu kalian copy dari unduhan dan paste kedalam package tersebut.



Code Program 164 File di dalam package parser

Dengan begitu kita telah selesai mengatasi *error*. Saatnya kembali ke tema. Eh? Tentu saja belum selesai. Kita belum mendeklarasikan schema ke dalam tema aplikasi kita.

Kita buka file theme.kt di package ui. Kita ubah parameternya menjadi seperti di gambar berikut.

```
@Composable
fun KoreDiaryTheme(
    appTheme: AppTheme,
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit
){
```

Code Program 165 Parameter pada tema

KoreDiaryTheme adalah nama tema yang otomatis dibuat saat kita membuat project ini (itu nama aplikasi saya, namanya akan menyesuaikan dengan nama aplikasi kalian).

Terdapat AppTheme disana, apakah kalian mengenalnya ?. AppTheme adalah enum class yang kita buat pada kotpref GlobalPreferences. Saya menyingkat importnya agar ringkas. Sebenarnya cara memanggilnya adalah GlobalPreferences.AppTheme (ini karena saya membuat AppTheme di dalam GlobalPreferences).

Kemudian ada darkTheme yang diisi dengan isSystemInDarkTheme(). Ini berfungsi jika aplikasi kita ingin menyesuaikan dengan tema yang dipilih pada system.

Selanjutnya, kita akan membuat val colorScheme yang berisi scheme (skema tema) yang terpilih nantinya. Karena saya membuat *multi-theme* untuk aplikasi ini, maka saya menggunakan *when* untuk menggunakan kondisi ketika tema mana yang terpilih dari AppTheme dan disesuaikan dengan scheme yang sudah saya buat sebelumnya.

```
val colorScheme : ColorScheme = when(appTheme){  
    AppTheme.DEFAULT_LIGHT -> LightColorScheme  
    AppTheme.DEFAULT_DARK -> DarkColorScheme  
    AppTheme.MOUNTAIN_LIGHT -> MountainLightScheme  
    AppTheme.MOUNTAIN_DARK -> MountainDarkScheme  
    AppTheme.SAKURA_LIGHT -> SakuraLightScheme  
    AppTheme.SAKURA_DARK -> SakuraDarkScheme  
}
```

Code Program 166 Val colorScheme

Sisanya biarkan saja seperti apa adanya, saya menggunakan codingan dari aplikasi saya sebelumnya.

```
val view : View = LocalView.current  
if (!view.isInEditMode) {  
    SideEffect {  
        val window : Window = (view.context as Activity).window  
        window statusBarColor = colorScheme.primary.toArgb()  
        WindowCompat.getInsetsController(window, view).isAppearanceLightStatusBars = darkTheme  
    }  
  
    MaterialTheme(  
        colorScheme = colorScheme,  
        typography = Typography,  
        content = content  
    )
```

Code Program 167 KoreDiaryTheme.kt

Selesai, tapi bohong (ngakak cik 🤣). Selanjutnya kita akan membuat SettingsScreen. Pada Setting, kita akan menampilkan *input text* untuk *update email, password* dan

username. Kita juga akan menampilkan dropdown agar pengguna bisa memilih tema yang sudah dibuat.

6.6 SettingsScreen

@Destination adalah hal yang penting dalam pembuatan screen. Kalau menambahkan ini setelah itu melakukan apa?. Yak, benar. Kita harus build atau menekan tombol “palu”.

```
@OptIn(ExperimentalMaterial3Api::class)
@Destination
@Composable
fun SettingsScreen(
    navigator: DestinationsNavigator,
    viewModel: SettingsViewModel = hiltViewModel(),
) {
```

Lalu, jikalau kita telah membuat viewmodel untuk sebuah screen, jangan lupa untuk memanggilnya.

```
var expanded: Boolean by remember { mutableStateOf(value = false) }
// val mutak dropdown semuanya apakah dipertahankan atau tidak
var selectedTheme: List<GlobalPreferences.AppTheme> by remember {
    mutableStateOf(GlobalState.theme)
    // tema yang terpilih, nilai default mengambil dari GlobalState
}

var isLogoutDialog: Boolean by remember { mutableStateOf(value = false) }

var name: String by remember { mutableStateOf(LocalUser.username) }
// username saat ini, nilai default dari GlobalState

var email: String by remember { mutableStateOf(LocalUser.email) }
// email saat ini

var password: String by remember {
    mutableStateOf(LocalUser.password) // password saat ini
}

var isPasswordError: Boolean by remember { mutableStateOf(value = false) }
var isEmailError: Boolean by remember { mutableStateOf(value = false) }
var passwordErrorMsg: String by remember { mutableStateOf(value = "") }
var showPassword: Boolean by remember { mutableStateOf(value = false) }
var isEdit: Boolean by remember { mutableStateOf(value = false) }

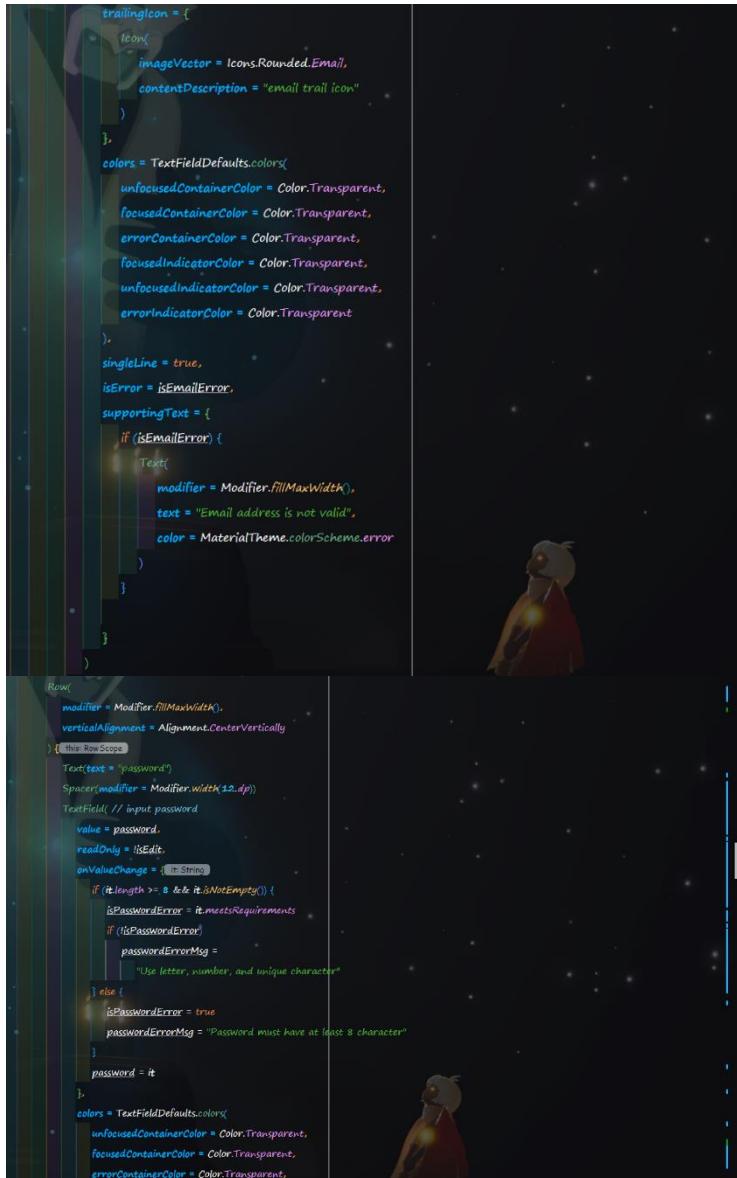
val themes: List<GlobalPreferences.AppTheme> = listOf(
    // list untuk ditampilkan di dropdown
    DEFAULT_DARK,
    DEFAULT_LIGHT,
    MOUNTAIN_DARK,
    MOUNTAIN_LIGHT,
    SAKURA_DARK,
    SAKURA_LIGHT
)

Scaffold { padding: PaddingValues ->
    BoxWithConstraints(
        modifier = Modifier
            .fillMaxSize()
            .padding(padding)
    ) + this.BoxWithConstraintsScope
        val maxHeight: Dp = this.maxHeight
        val headerHeight: Dp = maxHeight / 5
        val bodyHeight: Dp = maxHeight * 5 / 6
    )
}
```

```
Image(
    painter = painterResource(id = selectedTheme.background),
    // background gambar sesuai dengan tema
    contentDescription = null,
    modifier = Modifier
        .height(headerHeight)
        .fillMaxWidth()
        .align(Alignment.TopCenter),
    contentScale = ContentScale.Crop
)

Card(
    modifier = Modifier
        .fillMaxWidth()
        .height(bodyHeight)
        .align(Alignment.BottomCenter),
    shape = RoundedCornerShape( // agar sising komponen bulat
        topStart = 24.dp,
        topEnd = 24.dp,
        bottomEnd = 0.dp,
        bottomStart = 0.dp
    ),
    content = ( this.ColumnScope {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp)
                .verticalScroll(rememberScrollState()),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this.ColumnScope }

        Row(
            modifier = Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically
        ) { this.RowScope
            Text(text = "Email")
            Spacer(modifier = Modifier.width(12.dp))
            TextField( // input email
                value = email,
                readOnly = !isEdit,
                onValueChange = { it.String {
                    isEmailError = it.isValidEmail() }
                email = it
            },
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),
            placeholder = { Text(text = "email") },
            )
        }
    })
)
```



```
colors = TextFieldDefaults.colors(  
    unfocusedContainerColor = Color.Transparent,  
    focusedContainerColor = Color.Transparent,  
    errorContainerColor = Color.Transparent,  
    focusedIndicatorColor = Color.Transparent,  
    unfocusedIndicatorColor = Color.Transparent,  
    errorIndicatorColor = Color.Transparent  
,  
    singleLine = true,  
    placeholder = { Text(text = "password") },  
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),  
    visualTransformation = if (showPassword) VisualTransformation.None else PasswordVisualTransformation(),  
    isError = isPasswordError,  
    supportingText = {  
        if (isPasswordError) {  
            Text(  
                modifier = Modifier.fillMaxWidth(),  
                text = passwordErrorMsg,  
                color = MaterialTheme.colorScheme.error  
            )  
        }  
    },  
>,  
    trailingIcon = {  
        val icon: ImageVector = if (showPassword)  
            Icons.Rounded.LockOpen  
        else Icons.Rounded.Lock  
  
        IconButton(  
            onClick = { showPassword = !showPassword }  
        ) {  
            Icon(  
                imageVector = icon,  
                contentDescription = "Visibility"  
            )  
        }  
    }  
)
```

```
Row(
    modifier = Modifier.fillMaxWidth(),
    verticalAlignment = Alignment.CenterVertically
) { this RowScope {
    Text(text = "Name")
    Spacer(modifier = Modifier.width(12.dp))
    TextField(
        // input username
        value = name,
        readOnly = !isEdit,
        onValueChange = { it: String }
            name = it
    ),
    colors = TextFieldDefaults.colors(
        unfocusedContainerColor = Color.Transparent,
        focusedContainerColor = Color.Transparent,
        errorContainerColor = Color.Transparent,
        focusedIndicatorColor = Color.Transparent,
        unfocusedIndicatorColor = Color.Transparent,
        errorIndicatorColor = Color.Transparent
    ),
    placeholder = { Text(text = "username") },
)
}
}

Spacer(modifier = Modifier.height(8.dp))

Spacer(modifier = Modifier.height(24.dp))

HorizontalDivider(
    color = MaterialTheme.colorsScheme.surface,
    thickness = 1.6.dp
) // untuk membuat garis
Spacer(modifier = Modifier.height(16.dp))

Row(
    modifier = Modifier
        .align(Alignment.Start),
    verticalAlignment = Alignment.CenterVertically
) { this RowScope {
    Icon(
        imageVector = Icons.Rounded.Settings,
        contentDescription = null
    )
    Spacer(modifier = Modifier.width(8.dp))
    Text(
        text = "Settings",
        style = MaterialTheme.typography.titleLarge,
        fontWeight = FontWeight.SemiBold
    )
}
}
```

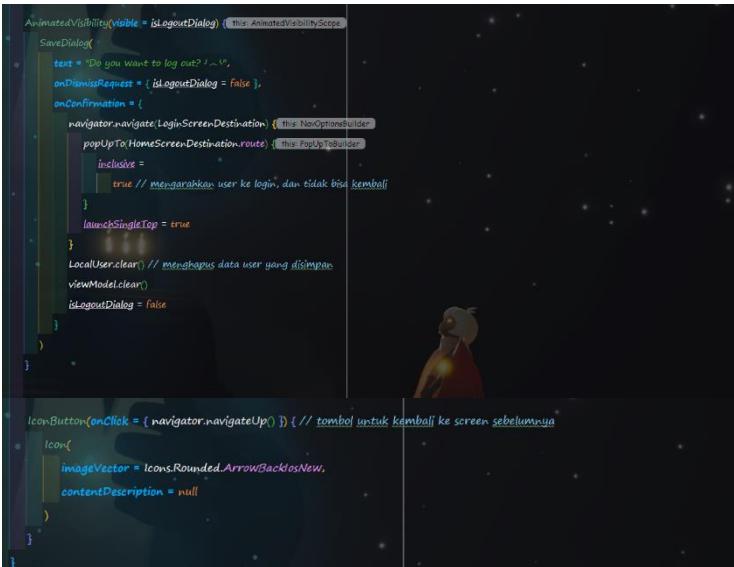
```
        Spacer(modifier = Modifier.height(24.dp))

    Column(
        modifier = Modifier
            .width(IntrinsicSize.Min)
            .align(Alignment.Start)
    ) { this.ColumnScope
        Row(
            verticalAlignment = Alignment.CenterVertically,
        ) { this.RowScope
            Icon(
                imageVector = Icons.Rounded.ColorLens,
                contentDescription = null
            )
            Spacer(modifier = Modifier.width(8.dp))
            Text(
                text = "Theme",
                style = MaterialTheme.typography.titleMedium,
                color = MaterialTheme.colorScheme.onBackground,
            )
        }
        Spacer(modifier = Modifier.height(4.dp))
    }

ExposedDropdownMenuItemBox(
    // dropdown
    expanded = expanded, // memanggil boolean, apakah dropdown di tampilkan
    onExpandedChange = { it Boolean } // ketika kondisi berubah, mengembalikan boolean untuk expanded
    expanded = expanded
),
) { this.ExposedDropdownMenuBoxScope
    OutlinedTextField(
        readOnly = true,
        value = selectedTheme.id,
        onValueChange = {},
        trailingIcon = {
            Icon(
                imageVector = if (!expanded) Icons.Rounded.KeyboardArrowUp else Icons.Rounded.KeyboardArrowDown,
                contentDescription = null,
            )
        },
        modifier = Modifier.menuAnchor()
        // menu anchor digunakan untuk memberi tahu dropdown menu bahwa komponen ini dianggap sebagai jarak dari menu
    )
}
```

```
    ExposedDropdownMenu(
        expanded = expanded,
        onDismissRequest = { expanded = false }) { this.ColumnScope
            themes.forEach { theme | GlobalPreferences.appTheme -> // setting value dari list tema yang dibuat
                DropdownMenuItem( // dropdown item
                    text = { Text(text = theme.id) }, // nama tema
                    onClick = { // ketika di klik
                        expanded = false
                        GlobalState.theme = theme // tema terpilih disimpan
                        GlobalPreferences.theme = theme
                        selectedTheme = theme
                    }
                )
            }
        }
    }

    Button( // button logout
        onClick = {
            isLogoutDialog = true
        }
    ) { this.RowScope
        Text(text = "Log Out")
        Spacer(modifier = Modifier.width(8.dp))
        icon(
            imageVector = Icons.AutoMirrored.Rounded.ExitToApp,
            contentDescription = null,
            modifier = Modifier.size(20.dp)
        )
    }
}
}
```



```
AnimatedVisibility(visible = !isLogoutDialog) { this.AnimatedVisibilityScope
    SaveDialog(
        text = "Do you want to log out? J - V",
        onDismissRequest = { isLogoutDialog = false },
        onConfirmation = {
            navigator.navigate(LoginScreenDestination, this.NoOptionsBuilder)
            popUpTo(HomeScreenDestination.route, inclusive = true)
                // mengarahkan user ke login, dan tidak bisa kembali
            launchSingleTop = true
        }
    )
    LocalUser.clear() // menghapus data user yang disimpan
    viewModel.clear()
    isLogoutDialog = false
}
}

IconButton(onClick = { navigator.navigateUp() }) { // tombol untuk kembali ke screen sebelumnya
    Icon(
        imageVector = Icons.Rounded.ArrowBackIosNew,
        contentDescription = null
    )
}
```

Code Program 168 SettingsScreen

Ketika pengguna memilih tema yang diinginkan, tema tersebut akan disimpan. Jika pengguna ingin logout, maka *data* pengguna tersebut akan dihapus.

6.7 EditorScreen (2)

Mungkin kalian bertanya, *fungsi* navargs itu untuk apa sih?.

```
data class EditorScreenNavArgs( // agar bisa menerima kiriman data
    val isEdit: Boolean,
    val uuid: String,
    val title: String,
    val ysiyg: String,
    val mood: String,
)
```

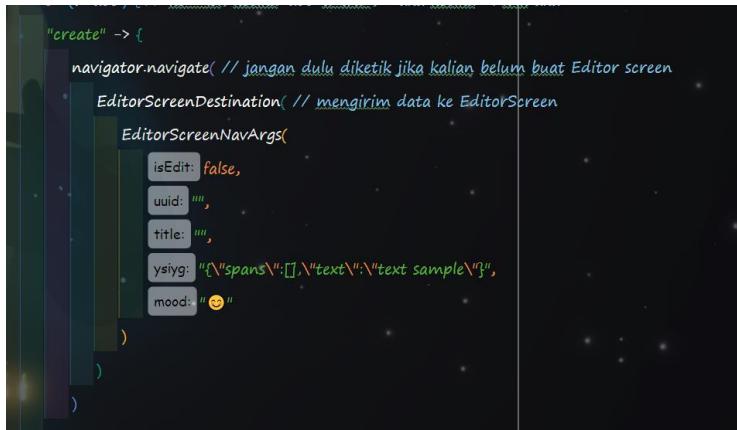
Code Program 169 EditorNavArgs

Kalau kita membuat EditorScreen atau CreateScreen hanya untuk membuat sebuah *post* baru, kenapa menggunakan hal tersebut ?.

Untuk menyingkat codingan, kita hanya perlu menggunakan satu screen untuk membuat ataupun meng-update *post* atau *data* baru.

Ketika *isEdit* bernilai *true*, berarti kita akan menjalankan logika *update* dan bukan *insert*. Begitupun sebaliknya.

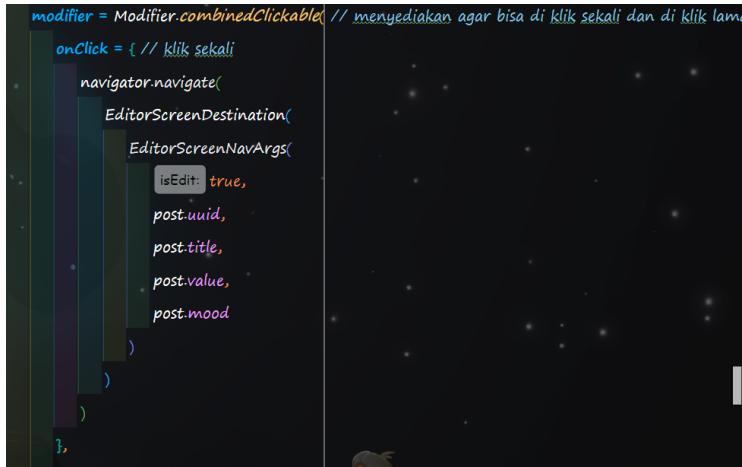
Data yang dikirim ketika kita mau menjalankan *insert* dengan menjalankan *update* itu berbeda.



```
"create" -> {
    navigator.navigate( // jangan dulu diketik jika kalian belum buat Editor screen
        EditorScreenDestination // mengirim data ke EditorScreen
        EditorScreenNavArgs(
            isEdit: false,
            uuid: "", 
            title: "", 
            ysiyg: "[{"spans":[], "text": "text sample"}]", 
            mood: "\ud83d\udcbb"
        )
    )
}
```

Code Program 170 Mengirim data lewat FAB

Ketika kita berpindah ke EditorScreen melalui FAB, kita akan menjalankan *insert*. Lalu kenapa *value rich editor* (ysiyg terinspirasi dari “*what you see is what you get*”) berisi Json?. Setelah saya telusuri lagi, jika kita mengirim string yang kosong, Rich editor tidak bisa membacanya dan malah ditampilkan sebagai *text*, serta tidak bisa diubah atau diedit.



Code Program 171 Mengirim data melalui item post

Sedangkan, jika kita berpindah ke EditorScreen melalui *item post* (jika kita klik sekali), maka kita berpindah dengan mengirim *data* yang didapatkan dari *database*.

Akhirnya, tersisa satu screen lagi sebelum kita mengakhiri bab ini. Ayo kita buat SearchScreen.

6.8 SearchScreen

Ketika *list data* yang dimiliki oleh pengguna terlalu banyak, apa yang akan dilakukan oleh pengguna?. Tentu saja mencarinya. Maka dari itu kita akan membuat SearchScreen.

```
@Destination
@OptIn(ExperimentalMaterial3Api::class, ExperimentalFoundationApi::class)
@Composable
fun SearchScreen(
    navigator: DestinationsNavigator,
    viewModel: SearchViewModel = hiltViewModel(),
) {
```

Seperti sebelumnya, ketika kita menambahkan @Destination kita perlu membuild ulang.

```
49
50 val state : ResponseState<List<Post>> by viewModel.state.collectAsStateWithLifecycle() // state dari viewmodel
51 val lazyState : LazyListState = rememberLazyListState() // state untuk lazy column
52 val flingBehavior : FlingBehavior = rememberSnapFlingBehavior(lazyListState = lazyState)
53 // animasi untuk lazy column, membutuhkan state
```

```
var queryState : String by remember {
    mutableStateOf(value: "") // initial value "" atau kosong, digunakan untuk set query untuk pencarian
}

var activeState : Boolean by remember {
    mutableStateOf(value: false) // untuk kondisi, apakah search bar masih aktif ?
}
```

```
Scaffold { values: PaddingValues } ->
    Column(
        modifier = Modifier
            .padding(values)
            .fillMaxSize()
    ) { this: ColumnScope
        IconButton(
            // tombol kembali ke halaman sebelumnya
            onClick = { navigator.navigateUp() },
            modifier = Modifier.align(Alignment.Start)
        ) {
            Icon(imageVector = Icons.Rounded.ArrowBackIosNew, contentDescription = null)
        }
    }
}
```

```
SearchBar(
    query = queryState, // value untuk search bar
    onQueryChange = { it: String } // ketika ada perubahan value dari search bar
    queryState = it
),
onSearch = { it: String } // apakah lagi search ?
activeState = true // mengubah kondisi menjadi true (aktif)
viewModel.search(it) // memanggil fungsi mencari berdasarkan query yang ada
),
active = activeState, // kondisi
onActiveChange = { it: Boolean } // perubahan kondisi
activeState = it
)
```

Code Program 172 SearchBar

Query di sini adalah kata atau string yg dimasukkan pengguna kedalam search bar. Kita memerlukan query untuk fungsi pencarian. Tapi, pengguna di sini hanya bisa mencari berdasarkan judul dari *item* atau *post* saja.

```
onActiveChange = { it: Boolean } // perubahan kondisi
    activeState = it
},
trailingIcon = {
    if (activeState) { // ketika kondisi true atau aktif
        IconButton(onClick = {
            if (queryState.isNotEmpty()) { // kalau query tidak kosong
                queryState = ""
            } else {
                activeState = false
                viewModel.clear() // menghapus query yang ada
            }
        })
    } {
        Icon(
            imageVector = Icons.Rounded.Close,
            contentDescription = "btn_close"
        )
    }
}
```

Code Program 173 Button Clear

Button untuk me-reset query. Query diubah menjadi kosong dan kondisi apakah sedang search diubah menjadi tidak aktif.

```
state.DisplayResult(  
    onidle = { // kalau tidak sedang melakukan apapun  
        Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) { this.BoxScope  
            Text(  
                text = "Search Something here ⌂",  
            )  
        }  
    },
```

```
onLoading = {  
    Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) { this.BoxScope  
        LinearProgressIndicator()  
    }  
},
```

```
onSuccess = { data : List<Post> ->  
    if (data.isNotEmpty()){ // menampilkan data ketika tidak kosong  
        LazyColumn(  
            modifier = Modifier.fillMaxSize(),  
            state = lazyState,  
            flingBehavior = flingBehavior,  
            content = [this: LazyListScope]  
                items(data) { this: LazyItemScope, post : Post ->  
                    val zoneTime : Date = post.createdAt.parseToDate()  
                    val formatted : String = zoneTime.formatToString()  
                    PostItem(  
                        title = post.title,  
                        date = formatted,  
                        mood = post.mood,  
                        modifier = Modifier.combinedClickable(  
                            onClick = {  
                                navigator.navigate(  
                                    EditorScreenDestination(  
                                        EditorScreenNavArgs(  
                                            isEdit: true,
```

```
PostItem(  
    title = post.title,  
    date = formatted,  
    mood = post.mood,  
    modifier = Modifier.combinedClickable(  
        onClick = {  
            navigator.navigate( // pindah ke editor screen  
                EditorScreenDestination(  
                    EditorScreenNavArgs(  
                        isEdit: true,  
                        post.uuid,  
                        post.title,  
                        post.value,  
                        post.mood  
                    )  
                )  
            )  
        })
```

```
)  
else{ // kalau data tidak ada  
    Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {  
        Text(  
            text = "Can't found it >~<",  
        )  
    }  
}
```

```
onError = { message : String } ->  
    Toast.makeText(LocalContext.current, message, Toast.LENGTH_SHORT).show()  
}
```

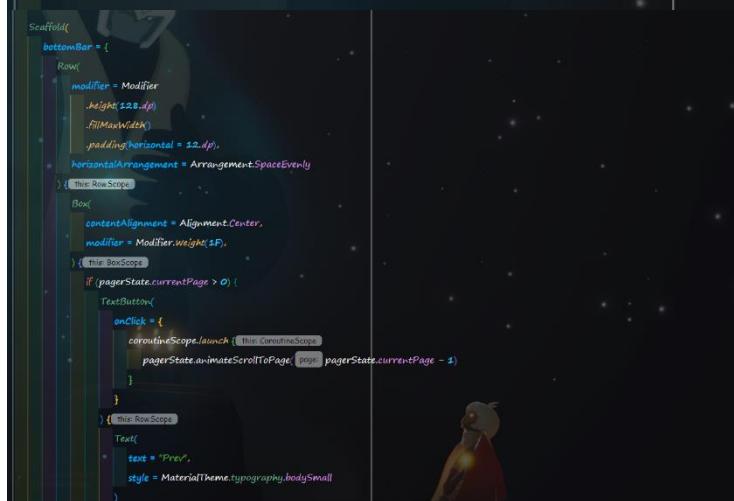
Code Program 174 SearchScreen

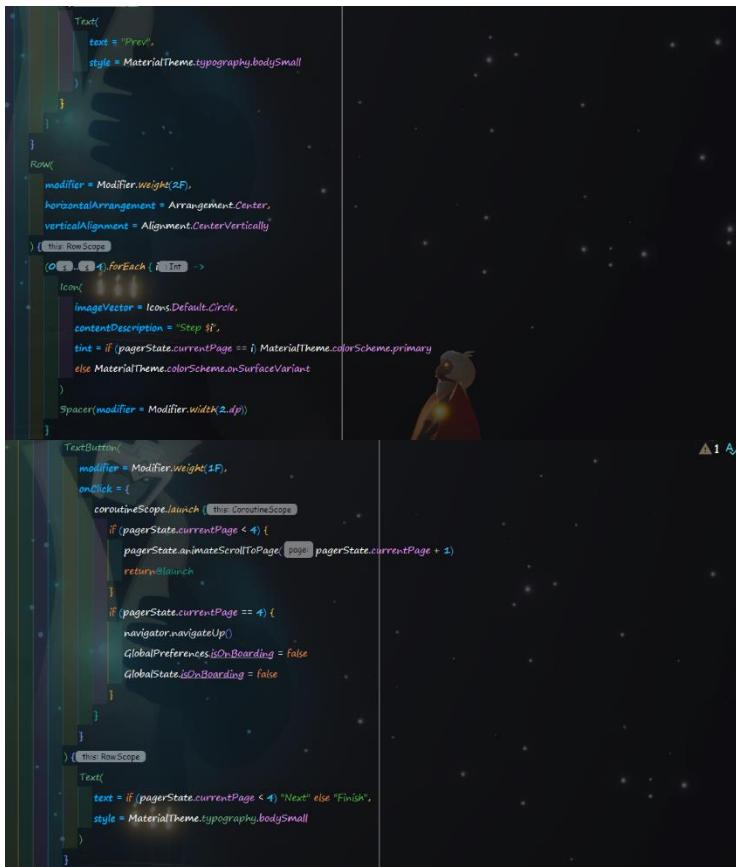
6.9 OnBoarding

Ini akan kita tampilkan ketika pengguna baru pertama kali memakai aplikasi.

```
@OptIn(ExperimentalFoundationApi::class)
@Destination
@Composable
fun OnBoardingScreen(
    navigator: DestinationsNavigator,
) {

    val pagerState: PagerState = rememberPagerState(pageCount = { 5 })
    val coroutineScope: CoroutineScope = rememberCoroutineScope()
```







```
    ) { innerPadding = PaddingValues( ... )
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(innerPadding)
        ) { this: ColumnScope
            HorizontalPager(
                state = pagerState,
                userScrollEnabled = false,
                modifier = Modifier.fillMaxSize()
            ) { this: PagerScope<Int>
                When (it) {
                    0 -> OnWelcome()
                    1 -> OnSync()
                    2 -> OnPress()
                    3 -> OnTheme()
                    4 -> OnFinish()
                }
            }
        }
    }
}

@Composable
fun OnWelcome() {
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) { this: ColumnScope
        Icon(
            painter = painterResource(id = R.drawable.icon_app),
            contentDescription = "Quran App Logo",
            tint = MaterialTheme.colorScheme.onSurface,
            modifier = Modifier.size(120.dp)
        )
        Spacer(modifier = Modifier.height(12.dp))
        Text(
            text = "Hi, welcome! 😊",
            textAlign = TextAlign.Center,
            style = MaterialTheme.typography.headlineMedium,
            fontWeight = FontWeight.SemiBold,
            color = MaterialTheme.colorScheme.onSurface
        )
    }
}
```

```
Spacer(modifier = Modifier.height(8.dp))

Text(
    modifier = Modifier.fillMaxWidth(),
    textAlign = TextAlign.Center,
    text = "Next to start",
    style = MaterialTheme.typography.titleLarge,
    color = MaterialTheme.colorScheme.onSurface
)

fun ObjSync() {
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) { this.ColumnScope

        Text(
            text = "You need internet to access your diary",
            textAlign = TextAlign.Center,
            style = MaterialTheme.typography.headlineMedium,
            fontWeight = FontWeight.SemiBold,
            color = MaterialTheme.colorScheme.onSurface
        )

        Spacer(modifier = Modifier.height(8.dp))

        Icon(
            imageVector = Icons.Rounded.Sync,
            contentDescription = null,
            modifier = Modifier.size(24.dp)
        )

        Spacer(modifier = Modifier.height(8.dp))

        Spacer(modifier = Modifier.height(8.dp))

        Text(
            modifier = Modifier.fillMaxWidth(),
            textAlign = TextAlign.Center,
            text = "Press 2x to sync",
            style = MaterialTheme.typography.titleLarge,
            color = MaterialTheme.colorScheme.onSurface
        )
    }
}
```

```
@OptIn(ExperimentalMaterialsApi::class, ExperimentalFoundationApi::class)
@Composable
fun OnPress() {
    var expanded : Boolean by remember {
        mutableStateOf(value = false)
    }

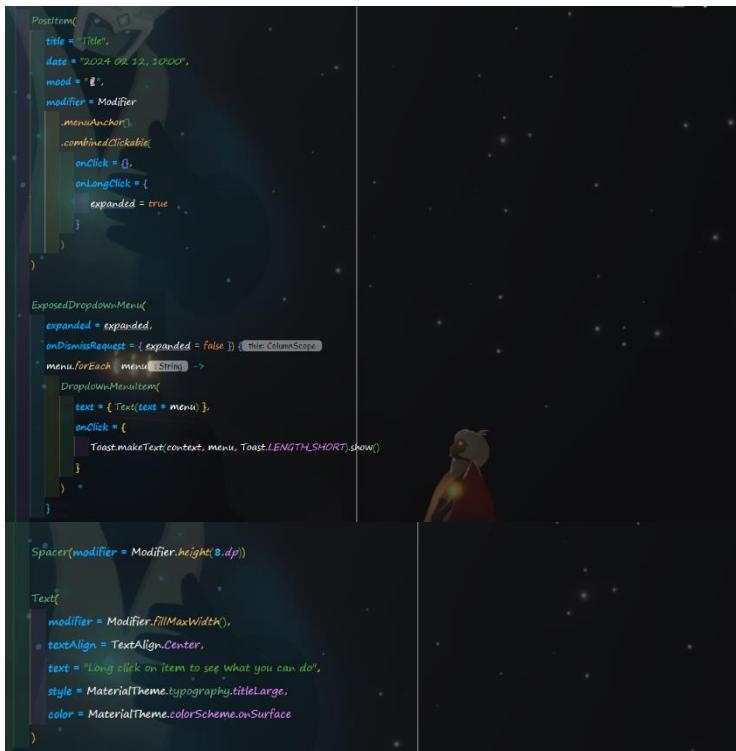
    val context : Context = LocalContext.current

    val menu : List<String> = listOf(
        "Hi",
        "Hello",
        "Halo",
        "Ha"
    )

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) { // this: ColumnScope
        Text(
            text = "Sub menu",
            textAlign = TextAlign.Center,
            style = MaterialTheme.typography.headlineMedium,
            fontWeight = FontWeight.SemiBold,
            color = MaterialTheme.colorScheme.onSurface
        )

        Spacer(modifier = Modifier.height(8.dp))

        ExposedDropdownMenuBox(
            // dropdown
            expanded = expanded, // memanggil boolean, apakah dropdown di tampilkan
            onExpandedChange = { it: Boolean // ketika kondisi berubah, mengembalikan boolean untuk expanded
                expanded = !expanded
            }
        ) { // this: ExposedDropdownMenuBoxScope
            
```



```
@Composable
fun OnTheme() {
    var expanded: Boolean by remember {
        mutableStateOf(value = false)
    }

    var selectedTheme: GlobalPreferences.AppTheme by remember {
        mutableStateOf(GlobalState.theme)
    }

    val themes: List = listOf(
        GlobalPreferences.AppTheme.DEFAULT_DARK,
        GlobalPreferences.AppTheme.DEFAULT_LIGHT,
        GlobalPreferences.AppTheme.MOUNTAIN_DARK,
        GlobalPreferences.AppTheme.MOUNTAIN_LIGHT,
        GlobalPreferences.AppTheme.SAKURA_DARK,
        GlobalPreferences.AppTheme.SAKURA_LIGHT
    )

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) { this.ColumnScope

        Text(
            text = "Sub menu",
            textAlign = TextAlign.Center,
            style = MaterialTheme.typography.headlineMedium,
            fontWeight = FontWeight.SemiBold,
            color = MaterialTheme.colorScheme.onSurface
        )

        Spacer(modifier = Modifier.height(8.dp))

        ExposedDropdownMenuBox(
            // dropdown
            expanded = expanded, // memanggil boolean, apakah dropdown di tampilkan
            onExpandedChange = { it: Boolean // ketika korang berubah, menjemputkan boolean untuk expanded
                expanded = !expanded
            },
        ) { this.ExposedDropdownMenuBoxScope
    }
}
```

```
OutlinedTextField(   
    readOnly = true,  
    value = selectedTheme.id,  
    onValueChange = {},  
    trailingIcon = {  
        Icon(  
            imageVector = if (expanded, Icons.Rounded.KeyboardArrowUp else Icons.Rounded.KeyboardArrowDown,  
            contentDescription = null  
        )  
    },  
    modifier = Modifier.menuAnchor()  
)  
  
ExposedDropdownMenu(   
    expanded = expanded,  
    onDismissRequest = { expanded = false } ) { this@ColumnScope  
    themes.forEach { theme : GlobalPreferences.AppTheme ->  
        DropdownMenuItem( // dropdown item  
            text = { Text(text = theme.id) }, // nama tema  
            onClick = { // ketika di klik  
                expanded = false  
                GlobalState.theme = theme // tema terpilih disimpan  
                GlobalPreferences.theme = theme  
                selectedTheme = theme  
            }  
        )  
    }  
    Spacer(modifier = Modifier.height(8.dp))  
  
    Text(  
        modifier = Modifier.fillMaxWidth(),  
        textAlign = TextAlign.Center,  
        text = "Long click on item to see what you can do",  
        style = MaterialTheme.typography.titleLarge,  
        color = MaterialTheme.colorScheme.onSurface  
    )  
}
```

```
fun OnFinish(
    modifier: Modifier = Modifier,
) {
    Column(
        modifier = modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) { this: ColumnScope
        Text(
            text = "Finish",
            textAlign = TextAlign.Center,
            style = MaterialTheme.typography.headlineLarge,
            fontWeight = FontWeight.Black
        )

        Spacer(modifier = Modifier.height(8.dp))

        Text(
            modifier = Modifier.fillWidth(),
            textAlign = TextAlign.Center,
            text = "You can use the app now",
            style = MaterialTheme.typography.titleLarge
        )
    }
}
```

Code Program 175 OnBoarding.kt

6.10 DetailScreen

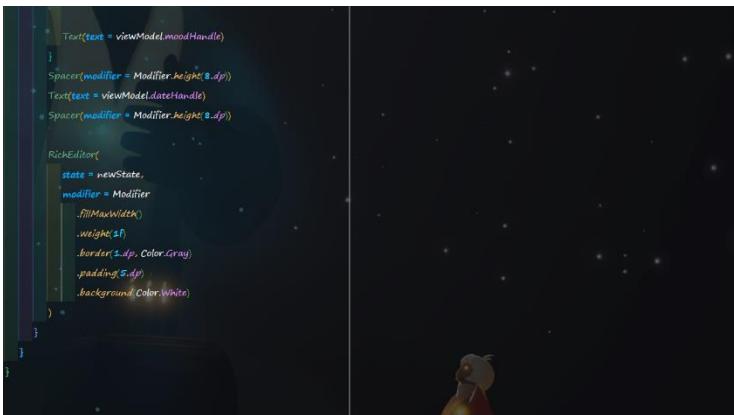
Pada layar pengguna bisa membagikan diary dalam bentuk gambar. Pengguna perlu meng-klik layar baru kemudian membagikannya melalui tombol *share*.

```
new =  
data class DetailScreenNavArgs(  
    // agar bisa menerima kirimman data  
    val title: String,  
    val year: String,  
    val mood: String,  
    val date: String,  
)  
  
new =  
@Destination(navArgsDelegate = DetailScreenNavArgs::class)  
@Composable  
fun DetailScreen(  
    navigator: DestinationsNavigator,  
    viewModel: DetailViewModel = hiltViewModel(),  
) {  
    val newState: RichEditorState = remember { // state untuk rich editor, berbasis JSON  
        val input: String = viewModel.valueHandle  
        RichEditorState.Builder()  
            .setInput(input)  
            .adapter(JsonEditorParser())  
            .build() //remember  
    }  
  
    val scope: CoroutineScope = rememberCoroutineScope()  
    val screenshotState: ScreenshotState = rememberScreenshotState()  
    val context: Context = LocalContext.current  
  
    Column(  
        modifier = Modifier  
            .fillMaxSize()  
            .padding(16.dp)  
    ) {  
        this@ColumnScope  
        Row(  
            modifier = Modifier.fillMaxWidth(),  
            horizontalArrangement = Arrangement.SpaceBetween  
        ) {  
            this@RowScope  
            IconButton(onClick = { navigator.navigateUp() }) {  
                Icon(imageVector = Icons.Rounded.ArrowBack_ios_New, contentDescription = null)  
            }
        }
    }
}
```

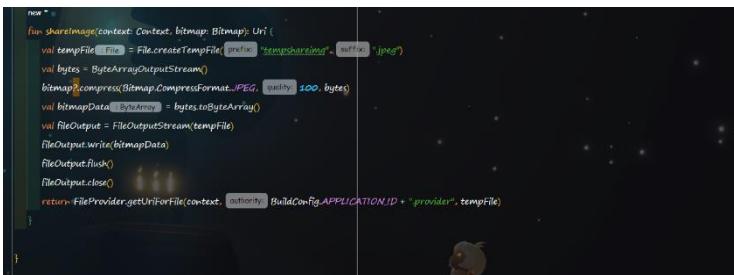
```
IconButton(
    onClick = {
        if (screenshotState.imageBitmap != null) {
            scope.launch { thisCoroutineScope.launch {
                val intent = Intent(Intent.ACTION_SEND)
                intent.type = "image/*"
                intent.putExtra(
                    Intent.EXTRA_STREAM,
                    screenshotState.imageBitmap?.let { it: ImageBitmap ->
                        viewModel.shareImage(
                            context,
                            it.asAndroidBitmap()
                        )
                    }
                )
            })
            context.startActivity(
                Intent.createChooser(intent, "Share With")
            )
        } else {
            Toast.makeText(context, text = "click on screen first to capture", Toast.LENGTH_SHORT).show()
        }
    }
), icon = Icons.Rounded.ArrowOutward, contentDescription = null)

Spacer(modifier = Modifier.height(8.dp))

ScreenshotBox(
    screenshotState = screenshotState,
    modifier = Modifier.clickable {
        scope.launch { thisCoroutineScope.launch {
            screenshotState.capture()
        }}
    }
) {
    Column(
        modifier = Modifier.padding(16.dp)
    ) {
        this.ColumnScope
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            this.RowScope
            Text(
                text = viewModel.titleHandle,
                style = MaterialTheme.typography.titleLarge,
                fontWeight = FontWeight.SemiBold
            )
        }
    }
}
```



Code Program 176 DetailScreen



Code Program 177 Convert bitmap to uri, DetailViewModel

Selesai, kita telah membuat seluruh screen. Tapi kita masih belum

bisa menjalankan aplikasi yang kita buat. Sebelum menjalankan aplikasi, ada beberapa hal yang harus kita persiapkan atau tambahkan. Ayo, lihat bab selanjutnya.

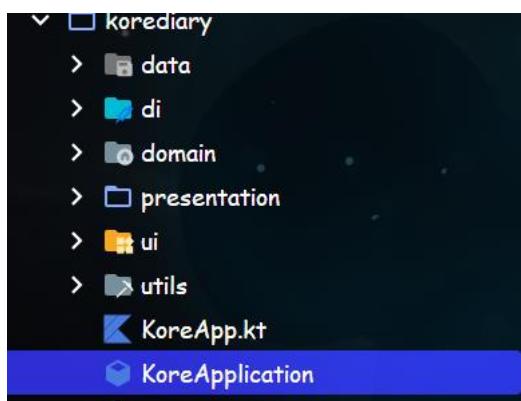


Bagian 7 :

Menyelesaikan Aplikasi

7.1 Application

Pada package aplikasi, buatlah sebuah class baru dan beri nama dengan NamaAplikasiApplication.



Code Program 178 Class Application

Karena kita menggunakan hilt-dagger, kita membutuhkan sebuah application agar aplikasi kita bisa berjalan.

Di dalam class application, kita buat seperti ini.

```
@HiltAndroidApp  
class KoreApplication : Application()
```

Code Program 179 Class Application (2)

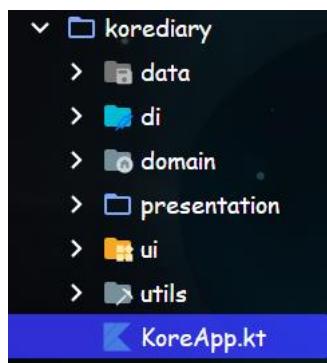
Oh, iya. Karena kita menggunakan worker, maka buatlah class application menjadi seperti ini.

```
@HiltAndroidApp  
class KoreApplication : Application(), Configuration.Provider {  
  
    @Inject  
    lateinit var workerFactory: HiltWorkerFactory  
  
    override val workManagerConfiguration: Configuration  
        get() = Configuration.Builder()  
            .setWorkerFactory(workerFactory)  
            .build()  
}
```

Code Program 180 Class Application (3)

7.2 App

Karena kita menggunakan bottom sheet, kita perlu membuat sebuah file composable App untuk menampung bottomsheet dan *screen* yang kita buat.



Code Program 181 File App

```
var sOnBoarding : Boolean by booleanPref(default: true)  
var isOnBoarding : Boolean by mutableStateOf(GlobalPreferences.isOnBoarding)
```

Code Program 182 var isOnBoarding pada GlobalPreferences & Global State

```
new"
    @Composable
    fun BottomBar(
        navController: NavController,
    ) {
        val currentDestination : TypedDestination<out Any> = navController.appCurrentDestinationAsState().value ?: NavGraphs.root.startAppDestination

        val barItems : List = listOf(
            BarItem(HomeScreenDestination, Icons.Rounded.Home, label: "Home"),
            BarItem(SearchScreenDestination, Icons.Rounded.Search, label: "Search"),
            BarItem(SettingsScreenDestination, Icons.Rounded.Settings, label: "Settings")
        )
    }

    NavigationBar { This is Row Scope
        barItems.forEach { barItem: BarItem ->
            NavigationBarItem(
                selected = currentDestination == barItem.directions,
                onClick = {
                    navController.navigate(barItem.directions) { thisNonNullable.popUpTo(currentDestination.route) { this.popUpToBuilder
                        inclusive = true
                    }
                    launchSingleTop = true
                },
                label = { Text(text = barItem.label) },
                icon = {
                    Icon(
                        imageVector = barItem.icon,
                        contentDescription = barItem.label
                    )
                }
            )
        }
    }
}
```

```
data class BarItem(
    val directions: DirectionDestinationSpec,
    val icon: ImageVector,
    val label: String,
)
```

Code Program 183 BottomBar.kt pada package component pada package presentation

```
@OptIn(ExperimentalMaterialNavigationApi::class, ExperimentalAnimationApi::class)
@Composable
fun KoroApp() {
    navController = NavHostController = rememberNavController(),
) {
    val bottomSheetNavigator = BottomSheetNavigator = rememberBottomSheetNavigator()
    navController.navigatorProvider += bottomSheetNavigator // agar bisa memunculkan bottom sheet

    val navBackStackEntry: NavBackStackEntry? by navController.currentBackStackEntryAsState()
    val currentState: String = navBackStackEntry?.destination?.route ?: HomeScreenDestination.route

    ModalBottomSheetLayout // bottom sheet
        bottomSheetNavigator = bottomSheetNavigator,
        sheetShape = RoundedCornerShape(topStart = 12.dp, topEnd = 12.dp)
    ) {
        Scaffold(
            bottomBar = {
                if (currentState == HomeScreenDestination.route || currentState == SearchScreenDestination.route || currentState == SettingsScreenDestination.route) {
                    BottomBar(navController = navController)
                }
            }
        )
    } // End of BottomSheetLayout

    DestinationsNavHost // nav host
        modifier = Modifier.padding(16.dp),
        navGraph = NavGraphs.root,
        navController = navController,
        startRoute = if (GlobalState.isOnBoarding) {
            OnBoardingScreenDestination
        } else {
            if (LocalUser.userDataIsEmpty()) {
                LoginScreenDestination
            } else {
                HomeScreenDestination
            }
        },
        engine = rememberAnimatedNavHostEngine()
    ) { this. // Manajemen Controller
        composable(HomeScreenDestination) { this. // AnnotatedDestinasiScopeUnit
            HomeScreen // karena di screen membutuhkan navigator, kita isi seperti ini
            navigator = destinationNavigator
        }
    }

    composable(RegisterScreenDestination) { this. // AnnotatedDestinasiScopeUnit
        RegisterScreen(navigator = destinationNavigator)
    }

    composable(LoginScreenDestination) { this. // AnnotatedDestinasiScopeUnit
        LoginScreen(navigator = destinationNavigator)
    }
}
```

```
composable(EditorScreenDestination) [this AnimatedDestinationScope<Unit>]
EditorScreen(
    navigator = destinationsNavigator,
    resultRecipient = resultRecipient()
)
}

composable(SettingsScreenDestination) [this AnimatedDestinationScope<Unit>]
SettingsScreen(navigator = destinationsNavigator)
}

composable(SearchScreenDestination) [this AnimatedDestinationScope<Unit>]
SearchScreen(navigator = destinationsNavigator)
}

composable(DetailScreenDestination) [this AnimatedDestinationScope<DetailScreenArgs>]
DetailScreen(navigator = destinationsNavigator)
} *

composable(OnBoardingScreenDestination) [this AnimatedDestinationScope<Unit>]
OnBoardingScreen(navigator = destinationsNavigator)
}
```

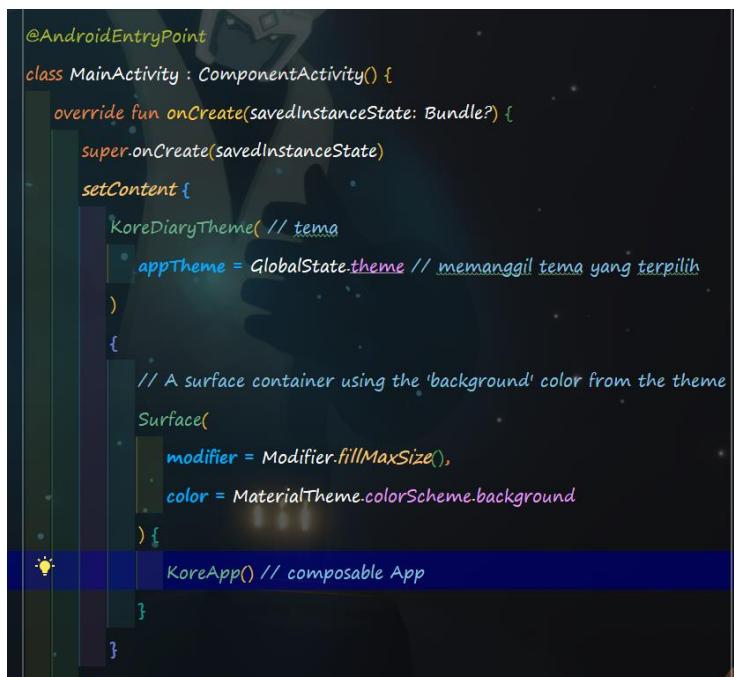
Code Program 184 KoreApp.kt

Pada BottomBar, kita mendaftarkan 3 screen yang akan ditampilkan pada *bottom bar navigation*. Kemudian untuk mengetahui ada di mana screen saat ini, kita menggunakan *current state* untuk memeriksa apakah screen saat ini *selected?*.

Kemudian kita membuat logika *if else*, apakah pengguna pertama kali menggunakan aplikasi ini?. Jika iya maka pengguna akan ditampilkan *OnBoardingScreen*.

7.3 MainActivity

Semua screen yang kita buat telah ditampung pada file App. Tapi bagaimana itu semua bisa ditampilkan?. Kita perlu menaruh composable App pada Main Activity. Karena kita menggunakan hilt-dagger, kita perlu memberi @Android Entry Point pada Activity.



```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            KoreDiaryTheme( // tema
                appTheme = GlobalState.theme // memanggil tema yang terpilih
            )
        }
        // A surface container using the 'background' color from the theme
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
            KoreApp() // composable App
        }
    }
}
```

A screenshot of the Android Studio code editor showing the MainActivity.kt file. The code defines a MainActivity that extends ComponentActivity. It overrides the onCreate method and sets its content to a composable function. Inside this function, it uses the KoreDiaryTheme composable, passing the appTheme parameter set to GlobalState.theme. It then creates a Surface composable with a modifier of fillMaxSize and a background color of MaterialTheme.colorScheme.background. Finally, it calls the KoreApp composable. The code editor has syntax highlighting and a dark theme.

Code Program 185 MainActivity

7.4 AndroidManifest

Aplikasi yang kita buat memerlukan internet dan worker. Kita perlu menambahkan beberapa hal pada file ini.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="32" />
<application>
```

Code Program 186 Uses Permission

```
<application
    android:name=".KoreApplication"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    ...>
```

Code Program 187 Menambahkan name pada application

```
<provider
    android:name="androidx.startup.InitializationProvider"
    android:authorities="${applicationId}.androidx-startup"
    android:exported="false"
    tools:node="merge">
    <meta-data
        android:name="androidx.work.WorkManagerInitializer"
        android:value="androidx.startup"
        tools:node="remove" />
</provider>
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/filepaths"/>
</provider>
```

Code Program 188 Menambahkan provider

Karena kita menggunakan *worker* dan *fileprovider*, kita perlu menambahkannya pada *manifest*. Kita perlu membuat *filepaths.xml* untuk *FileProvider*.

```
</> backup_rules.xml  
</> data_extraction_rules.xml  
</> filepaths.xml
```

```
<paths>  
    <files-path name="files" path="/" />  
    <external-files-path name="external_files" path="/" />  
    <external-path name="external_files" path="/" />  
    <cache-path name="cached_files" path="/" />  
    <external-cache-path name="cached_files" path="/" />  
    <root-path name="root" path="/" />  
</paths>
```

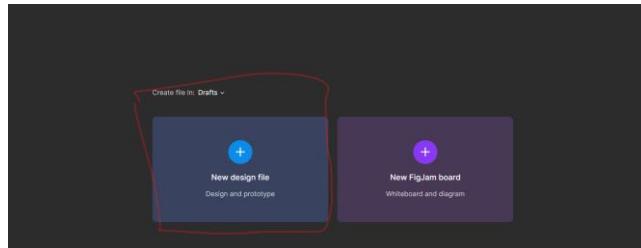
Code Program 189 filepaths.xml

7.5 SplashScreen & Icon

Apakah kita akan membuat sebuah screen lagi?. Tidak. Apakah kalian tahu apa itu splashscreen?. SplashScreen biasanya menampilkan logo aplikasi sebelum masuk ke aplikasi. Itulah yang akan kita buat kali ini.

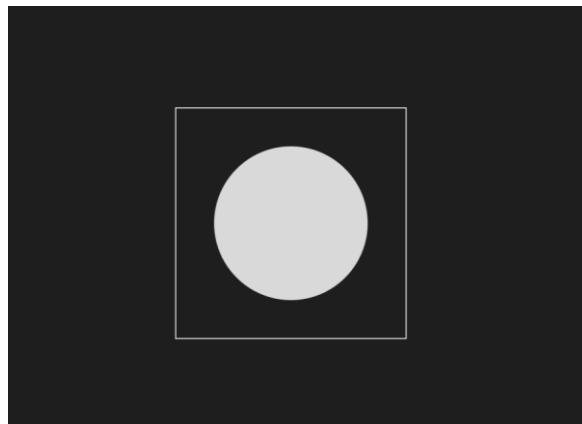
Biasanya, SplashScreen dibuat menggunakan compose screen atau activity baru. Kita tidak akan membuatnya secara manual seperti itu. Google telah menyediakan library untuk api splash screen. Kita hanya memerlukan logo aplikasi yang akan kita tampilkan.

Kalian bisa membuat logo kalian di figma misalnya. Saya akan membuat logo saya sendiri di figma. Kita buat draft baru untuk membuat logo aplikasi.



Gambar 44 Membuat draft baru

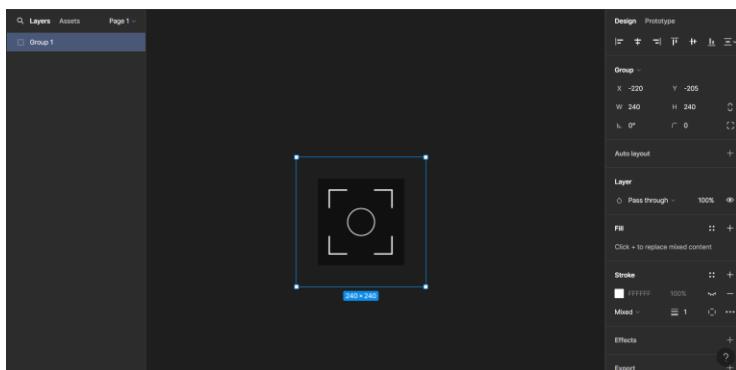
Lalu kita buat sebuah persegi dengan ukuran 240 x 240 dp. Lalu kita buat lingkaran di tengah nya dengan ukuran 160 x 160 dp.



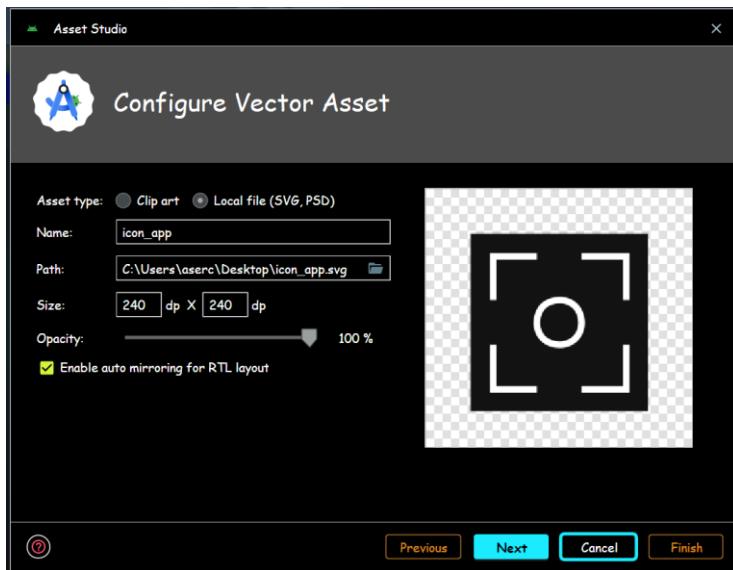
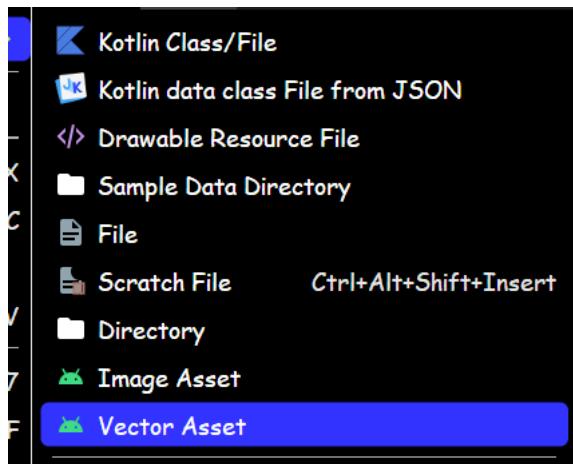
Gambar 45 Ukuran logo

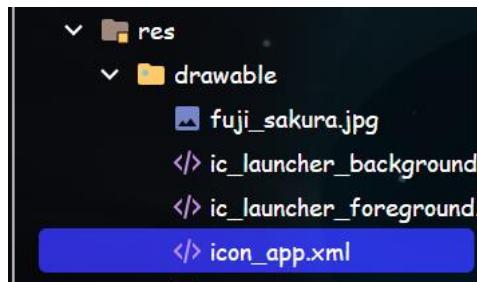
Untuk apa lingkaran tersebut?. Lingkaran tersebut menjadi standar agar logo aplikasi kita bisa terlihat dengan sempurna. Jika logo aplikasi kita melebihi lingkaran tersebut, logo aplikasi kita akan terpotong dan tidak terlihat dengan baik.

Hilangkan garis pada kotak dan buat warnanya menjadi transparan. Kemudian eksport sebagai svg agar bisa ditambahkan sebagai vector pada drawable.



Gambar 46 Pastikan gambar di group





Gambar 47 Menambahkan vector image

Kemudian kita buka file themes.xml pada package `values`. Kita buat tema baru di dalam file tersebut.

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

    <style name="Theme.KoreDiary" parent="android:Theme.Material.Light.NoActionBar" />

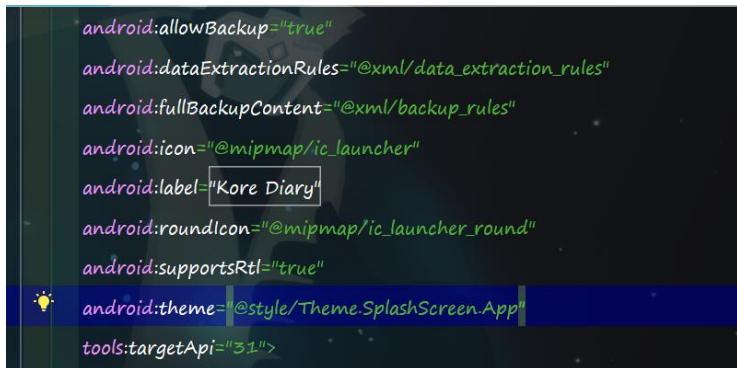
    <style name="Theme.SplashScreen.App" parent="Theme.SplashScreen">
        <item name="windowSplashScreenBackground">#123456</item>
        <item name="windowSplashScreenAnimatedIcon">@drawable/ic_app_icon</item>
        // memanggil icon
        <item name="postSplashScreenTheme">@style/Theme.KoreDiary</item>
        // memanggil tema di atas setelah splash screen
    </style>

</resources>
```

The screenshot shows the Android Studio code editor with the XML file 'themes.xml' open. The code defines two styles: 'Theme.KoreDiary' which inherits from 'Theme.Material.Light.NoActionBar', and 'Theme.SplashScreen.App' which inherits from 'Theme.SplashScreen'. The 'Theme.SplashScreen.App' style includes items for 'windowSplashScreenBackground' and 'windowSplashScreenAnimatedIcon', both set to reference '@drawable/ic_app_icon'. It also includes items for 'postSplashScreenTheme' which points to 'Theme.KoreDiary' and a comment indicating it calls the theme after the splash screen.

Code Program 190 Menambahkan tema

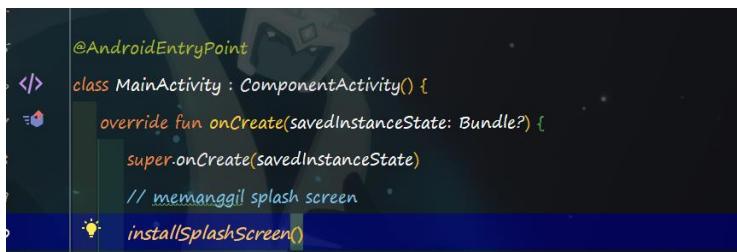
Lalu pada `AndroidManifest`, panggil tema yang kita buat di dalam application.



```
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Kore Diary"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SplashScreen.App"
        tools:targetApi="31"/>
```

Code Program 191 Mengganti tema

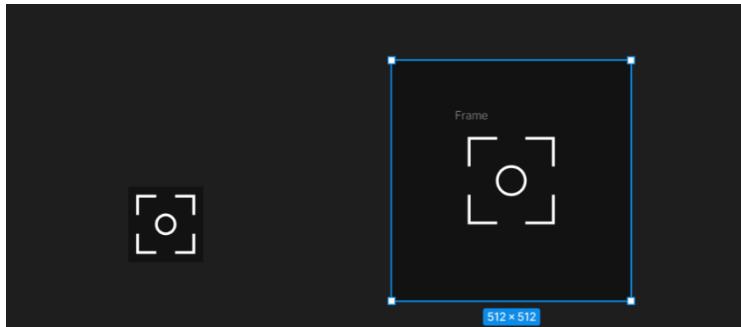
Setelah itu kita panggil pada MainActivity.



```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // memanggil splash screen
        installSplashScreen()
    }
}
```

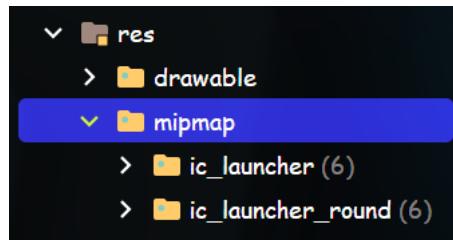
Code Program 192 InstallSplashScreen

Untuk menambahkan icon untuk aplikasi, kita perlu membuat lagi logo dengan ukuran 512 x 512.

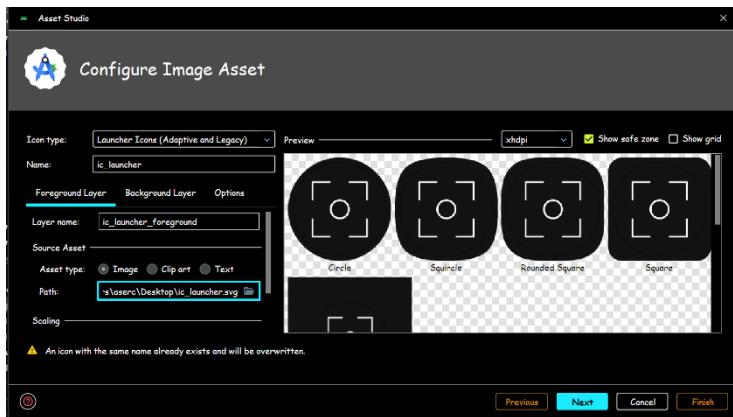


Gambar 48 Logo ukuran 512 x 512

Setelah itu kalian tambahkan image asset pada package mipmap di res. Kalian sesuaikan image dan backgroundnya.



Gambar 49 Package mipmap



Gambar 50 Menambah image asset

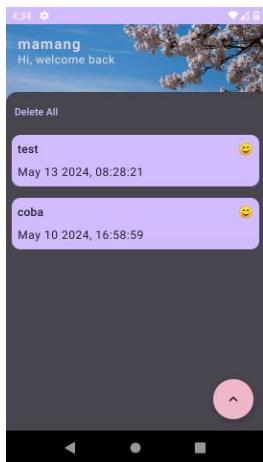
Lalu, kita gunakan iconnya pada manifest.

```
<application
    android:name=".KoreApplication"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher" | cursor over
    android:label="Kore Diary"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.SplashScreen.App"
    tools:targetApi="31">
</application>
```

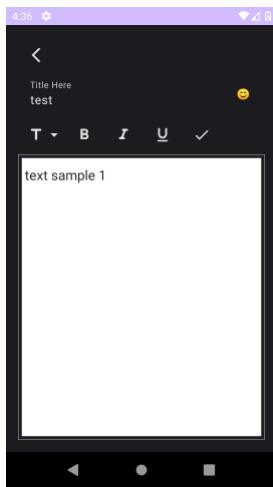
Code Program 193 AndroidManifest

Hore, kita telah menyelesaikan aplikasi yang kita buat. Terima kasih telah membaca buku ini meski banyak kurangnya. Hello World!

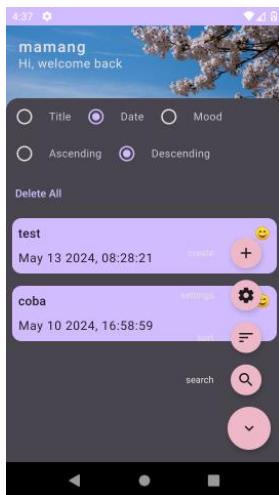
7.5 Preview



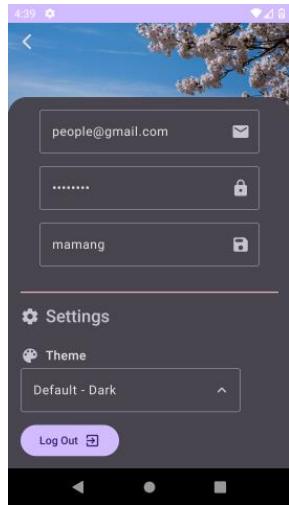
Gambar 51 Home Screen



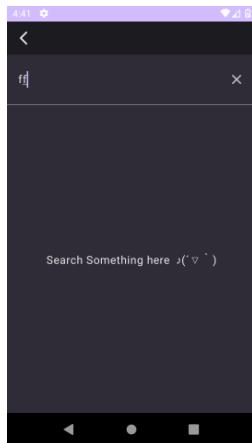
Gambar 52 EditorScreen



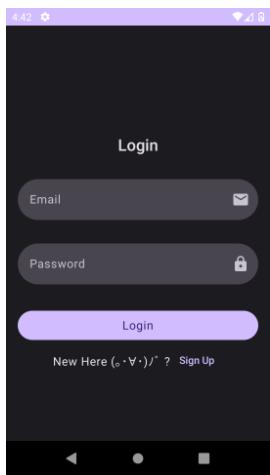
Gambar 53 HomeScreen (2)



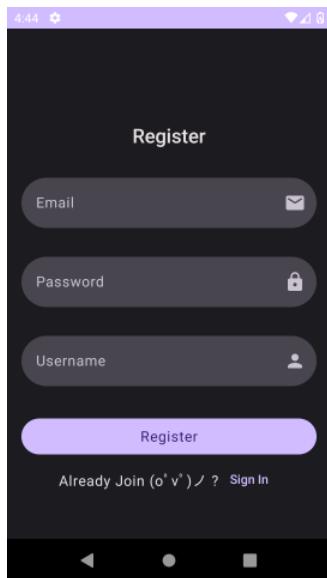
Gambar 54 Profile Screen (wkwkwk)



Gambar 55 SearchScreen



Gambar 56 LoginScreen



Gambar 57 RegisterScreen

Tentang Penulis



Penulis bernama Muhammad Akna Mafaid Ilmi Anshori. Penulis lahir pada 10 Februari 2007 di Sragen.

Sebelum menulis buku ini, penulis mendapatkan beberapa sertifikat nasional dalam pengembangan android.

Penulis juga membuat beberapa project dan beberapaanya diterbitkan di google play store. Buku ini ditulis sebagai hasil dari pembelajaran penulis selama ini.

Penulis masih aktif pada jenjang pendidikan sebagai siswa SMK di sekolah Rabbaanii Islamic School. Penulis mendalami minatnya pada jurusan Rekayasa Perangkat Lunak.

Referensi

Teori

<https://www.dicoding.com/academies/51/tutorials/1179>

<https://developer.android.com/>

Code Program

<https://stackoverflow.com/questions/71925505/how-to-convert-bitmapdrawable-or-bitmap-to-imagebitmap-in-kotlin>

<https://stackoverflow.com/questions/41754799/android-illegalargumentexception-failed-to-find-configured-root-that-contains>

<https://stackoverflow.com/questions/18249007/how-to-use-support-fileprovider-for-sharing-content-to-other-apps>

<https://stackoverflow.com/questions/68870406/share-button-in-compose>