به نام خدا

ساختار پوشه‌ای ماژول Core در پروژه Jetpack Compose

core/

├── src/main/java/com/example/core/

│ ├── di/ # مدیریت وابستگی‌ها (Dependency Injection)

│ │ ├── CoreModule.kt # ماژول‌های اصلی DI (Koin)

│ │ ├── NetworkModule.kt # پیکربندی Ktor برای ارتباطات شبکه

│ │ ├── RepositoryModule.kt # ماژول‌های Repository

│ │ └── LoggerModule.kt # پیکربندی Timber برای لاگ‌ها

│ │

│ ├── network/ # مدیریت شبکه و ارتباطات

│ │ ├── ApiResponse.kt # مدل پاسخ API

│ │ ├── NetworkResult.kt # مدیریت نتایج شبکه

│ │ ├── Resource.kt # مدیریت وضعیت منابع (State: Loading, Success, Error)

│ │ ├── ErrorHandler.kt # مدیریت خطاهای شبکه

│ │ ├── NetworkException.kt # استثناهای شبکه

│ │ └── NetworkUtils.kt # ابزارهای کمکی شبکه

│ │

│ ├── base/ # کلاس‌های پایه

│ │ ├── BaseViewModel.kt # ViewModel پایه

│ │ ├── BaseRepository.kt # Repository پایه

│ │ └── BaseActivity.kt # Activity پایه

│ │

│ ├── ui/ # کامپوننت‌ها و ابزارهای UI

│ │ ├── compose/ # کامپوننت‌ها و ویجت‌های Compose

│ │ │ ├── BaseComposable.kt # کامپوننت‌های پایه

│ │ │ ├── NetworkResultHandler.kt# مدیریت نمایش نتایج شبکه در UI

│ │ │ └── ThemeUtils.kt # تنظیمات تم‌ها

│ │ ├── images/ # پیکربندی تصاویر با Coil

│ │ ├── navigation/ # مدیریت ناوبری در Compose

│ │ │ └── NavigationGraph.kt # گراف ناوبری

│ │ └── video/ # پیکربندی پخش ویدیو (ExoPlayer)

│ │ └── ExoPlayerManager.kt # پیکربندی ExoPlayer

│ │

│ ├── utils/ # توابع و ابزارهای کمکی عمومی

│ │ ├── extension/ # Extension Functions برای کلاس‌ها

│ │ ├── constants/ # ثابت‌ها و تنظیمات پروژه

│ │ ├── network/ # ابزارهای شبکه

│ │ │ └── NetworkResultHandler.kt # نمایش وضعیت‌های شبکه

│ │ └── logger/ # ابزارهای مدیریت لاگ‌ها

│ │

│ └── config/ # پیکربندی‌های عمومی پروژه

│ ├── constants/ # ثابت‌های سراسری

│ └── di/ # پیکربندی‌های DI

└── build.gradle # پیکربندی Gradle ماژول Core

توضیحات جزئی بخش‌ها:

1. مدیریت وابستگی‌ها (DI)

* CoreModule.kt: ماژول Koin که تمامی وابستگی‌ها را تزریق می‌کند. در اینجا می‌توانید سرویس‌ها، ViewModel‌ها و Repositories را ثبت کنید.
* NetworkModule.kt: پیکربندی Ktor برای ارتباطات شبکه (پیکربندی کلاینت شبکه).
* RepositoryModule.kt: ماژول‌های مربوط به Repository که می‌تواند شامل اتصال به شبکه، دیتابیس و داده‌های محلی باشد.
* LoggerModule.kt: پیکربندی Timber برای مدیریت لاگ‌ها و بررسی خطاهای پروژه.

2. مدیریت شبکه

* ApiResponse.kt: مدل پاسخ از API که شامل ساختارهای مختلف داده‌های دریافتی از API است.
* NetworkResult.kt: یک sealed class برای مدیریت وضعیت‌های مختلف شبکه (مثل Loading, Success, Error).
* Resource.kt: مدل جامع برای مدیریت وضعیت منابع، مانند Resource.loading(), Resource.success(), Resource.error().
* ErrorHandler.kt: کلاسی برای مدیریت و تبدیل خطاهای شبکه به پیام‌های قابل نمایش برای کاربر.
* NetworkException.kt: مدل استثناها برای مدیریت خطاهای شبکه مانند 404 و 500.
* NetworkUtils.kt: ابزارهای کمکی برای بررسی وضعیت اتصال شبکه، تبدیل کدهای خطا و مدیریت ارتباطات شبکه.

3. کلاس‌های پایه

* BaseViewModel.kt: ViewModel پایه که مدیریت وضعیت‌ها و داده‌ها را انجام می‌دهد.
* BaseRepository.kt: Repository پایه برای استفاده مجدد در سایر Repository‌ها. در این کلاس می‌توانید متدهایی برای دسترسی به داده‌ها از منابع مختلف (مثل شبکه یا دیتابیس) ایجاد کنید.
* BaseActivity.kt: Activity پایه برای مدیریت ویژگی‌های عمومی در تمام فعالیت‌ها.

4. کامپوننت‌ها و ابزارهای UI

* BaseComposable.kt: کامپوننت‌های پایه Compose که برای نمایش وضعیت‌های عمومی مانند بارگذاری، خطا، و داده‌ها استفاده می‌شوند.
* NetworkResultHandler.kt: کامپوننتی برای مدیریت نمایش وضعیت‌های شبکه در UI. می‌توانید وضعیت‌هایی مثل بارگذاری، موفقیت و خطا را در اینجا مدیریت کنید.
* ThemeUtils.kt: توابع و ابزارهایی برای تنظیمات تم و استایل‌های مشترک در کامپوز‌ها.
* CoilImageLoader.kt: پیکربندی برای بارگذاری تصاویر با استفاده از Coil.
* ExoPlayerManager.kt: پیکربندی برای پخش ویدیو با ExoPlayer.
* NavigationGraph.kt: گراف ناوبری برای مدیریت مسیرهای مختلف در اپلیکیشن Compose.

5. توابع و ابزارهای کمکی عمومی

* extension/: Extension functions برای افزودن قابلیت‌های جدید به کلاس‌ها و استفاده آسان‌تر از آن‌ها.
* constants/: ثابت‌های پروژه مانند URL‌ها، API keys و تنظیمات سراسری.
* network/: ابزارهای کمکی برای مدیریت وضعیت‌های شبکه، خطاها و مدیریت ارتباطات.
  + NetworkResultHandler.kt: تابعی برای تبدیل نتایج شبکه به UI مناسب.
* logger/: توابع کمکی برای مدیریت لاگ‌ها با استفاده از ابزارهایی مانند Timber.

6. پیکربندی‌های عمومی پروژه

* constants/: ثابت‌های سراسری مانند URL‌ها، کلیدهای API و مقادیر پیش‌فرض.
* di/: پیکربندی‌های مربوط به DI که به صورت سراسری در پروژه استفاده می‌شود.

نتیجه‌گیری

با این ساختار، شما قادر خواهید بود که به‌طور مؤثر نتایج شبکه را در پروژه خود مدیریت کنید، وضعیت‌های مختلف داده‌ها را به راحتی نشان دهید، و با استفاده از Koin وابستگی‌ها را به‌طور مرتب مدیریت کنید. همچنین با ایجاد Resource و NetworkResult می‌توانید تمامی وضعیت‌ها و خطاها را به شکل سازمان‌دهی‌شده مدیریت کنید.

ساختار اولیه utils/:

core/src/main/java/com/example/core/utils/

├── extension/

│ └── StringExtensions.kt

├── constants/

│ ├── NetworkConstants.kt

│ └── AppConstants.kt

├── logger/

│ └── LoggerHelper.kt

📄 فایل `ComposeExtension.kt`

package com.example.core.utils.extension

import androidx.compose.foundation.clickable

import androidx.compose.foundation.interaction.MutableInteractionSource

import androidx.compose.runtime.\*

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.alpha

import androidx.compose.ui.platform.LocalDensity

import androidx.compose.ui.unit.Dp

import kotlinx.coroutines.delay

/\*\*

\* توابع Extension برای Jetpack Compose

\* این فایل توابع کمکی برای مدیریت Modifier و رویدادهای UI در Compose را فراهم می‌کند.

\*/

/\*\*

\* کلیک همراه با جلوگیری از دوبار کلیک سریع (Debounce Click)

\* @param debounceTime زمان فاصله بین کلیک‌ها (به میلی‌ثانیه)

\* @param onClick عملی که هنگام کلیک باید انجام شود

\*/

fun Modifier.debouncedClickable(

debounceTime: Long = 600L,

onClick: () -> Unit

): Modifier = composed {

var lastClickTime by remember { mutableStateOf(0L) }

val interactionSource = remember { MutableInteractionSource() }

this.clickable(

interactionSource = interactionSource,

indication = null

) {

val currentTime = System.currentTimeMillis()

if (currentTime - lastClickTime >= debounceTime) {

lastClickTime = currentTime

onClick()

}

}

}

/\*\*

\* تغییر شفافیت (alpha) ویو بر اساس فعال یا غیرفعال بودن

\* @param isEnabled فعال یا غیرفعال بودن

\* @param disabledAlpha میزان شفافیت در حالت غیرفعال

\*/

fun Modifier.alphaIfDisabled(

isEnabled: Boolean,

disabledAlpha: Float = 0.5f

): Modifier = this.alpha(if (isEnabled) 1f else disabledAlpha)

/\*\*

\* تبدیل Dp به Px

\* @return مقدار معادل به پیکسل

\*/

@Composable

fun Dp.toPx(): Float {

val density = LocalDensity.current

return with(density) { this@toPx.toPx() }

}

/\*\*

\* اجرای یک تابع بعد از یک مدت زمان مشخص (Delay)

\* @param timeMillis زمان تاخیر بر حسب میلی‌ثانیه

\* @param onDelayFinish تابعی که بعد از پایان تاخیر اجرا می‌شود

\*/

@Composable

fun OnDelay(

timeMillis: Long,

onDelayFinish: () -> Unit

) {

LaunchedEffect(key1 = true) {

delay(timeMillis)

onDelayFinish()

}

}

### 📚 توضیحات کلی:

- `debouncedClickable`: جلوی کلیک‌های پشت سر هم رو می‌گیره (مثل حالت ClickDebounce توی View)

- `alphaIfDisabled`: وقتی دکمه یا ویو غیرفعال میشه، خودش شفاف‌تر میشه.

- `toPx()`: گاهی توی کامپوز نیاز داری Dp رو دستی به Pixel تبدیل کنی.

- `OnDelay()`: میخوای بعد چند میلی‌ثانیه یک کدی اجرا بشه؟ این آماده‌ست!

✅ مثال استفاده از این توابع در کامپوز:

Button(

onClick = { /\* عملیات \*/ },

modifier = Modifier

.debouncedClickable {

// فقط یک بار در بازه زمانی کلیک می‌کند

}

.alphaIfDisabled(isEnabled = false)

) {

Text("ثبت سفارش")

}

یا مثلاً:

OnDelay(timeMillis = 3000L) {

// بعد ۳ ثانیه اینجا اجرا میشه

}