Memuat dan Menampilkan Gambar dari Internet

Coil adalah library pengolahan gambar untuk Android yang dikembangkan oleh Kotlin. Coil merupakan singkatan dari **Co**routine **I**mage **L**oader. Library ini fokus pada kecepatan, efisiensi, dan kemudahan penggunaan, menjadikannya alternatif populer untuk library pengolahan gambar lainnya seperti Glide dan Picasso.

Fitur Utama Coil:

Modern dan Ringan: Coil dibangun dengan memanfaatkan coroutine Kotlin dan memanfaatkan fitur-fitur modern Android, menghasilkan library yang ringan dan efisien.

Fast: Coil mengoptimalkan proses pengolahan gambar, termasuk caching dan decoding, untuk memberikan waktu pemuatan gambar yang cepat.

Mudah Digunakan: Coil memiliki API yang sederhana dan intuitif, sehingga mudah diintegrasikan ke dalam aplikasi Android.

Fleksibel: Coil mendukung berbagai sumber gambar, termasuk URL, file lokal, resource drawable, dan lainnya.

Composable: Coil menyediakan dukungan untuk Jetpack Compose melalui library coil-compose.

Cara Menggunakan Coil:

Tambahkan dependency:

```
implementation("io.coil-kt.coil3:coil-compose:3.0.4")
implementation("io.coil-kt.coil3:coil-network-okhttp:3.0.4")
```

Muat gambar:

```
AsyncImage(
    model = "https://example.com/image.jpg",
    contentDescription = "My Image"
)
```

Contoh Proyek

Berikut adalah proyek untuk memuat dan menampilkan gambar dari internet menggunakan **Retrofit** dan **Coil** dengan **Jetpack Compose**:

Kita akan mengakses URL:

https://android-kotlin-fun-mars-server.appspot.com/photos

1. Tambahkan dependensi

Buka file `build.gradle (Module: app)` dan tambahkan dependensi berikut:

```
dependencies {

// Retrofit dan Gson
implementation ("com.squareup.retrofit2:retrofit:2.9.0")
implementation ("com.squareup.retrofit2:converter-gson:2.9.0")
// ViewModel
implementation ("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.7")
//coil
implementation("io.coil-kt.coil3:coil-compose:3.0.4")
implementation("io.coil-kt.coil3:coil-network-okhttp:3.0.4")

}
```

2. Buat model data

Misalnya, API kita memberikan data seperti ini:

```
{
  "id": "424905",
  "img_src": "https://mars.jpl.nasa.gov/msl-raw-
images/msss/01000/mcam/1000MR0044631300503690E01_DXXX.jpg"
}
```

Buat file data class `MarsPhoto.kt`:

```
data class MarsPhoto(
    val id: String,
    val img_src: String
)
```

3: Buat interface API dengan Retrofit

Buat file `ApiService.kt`:

```
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.http.GET

interface ApiService {
    @GET("photos")
    suspend fun getPhotos(): List<MarsPhoto>
}

object RetrofitClient {
    private const val BASE_URL = " https://android-kotlin-fun-mars-
server.appspot.com/"

    val apiService: ApiService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(ApiService::class.java)
```

```
    }
}
```

4: Repository

Buat file kelas `PhotosRepository.kt`:

```
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext

class PhotosRepository(private val apiService: ApiService) {
    suspend fun fetchPhotos(): Result<List<MarsPhoto>> {
        return try {
            val response = withContext(Dispatchers.IO) {
                apiService.getPhotos()
            }
            Result.success(response)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

5. Tambahkan ViewModel

Buat file `PhotoViewModel.kt`:

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

class PhotoViewModel : ViewModel() {
    private val repository = PhotoRepository(RetrofitClient.apiService)

    private val _photos = MutableStateFlow<List<MarsPhoto>>(emptyList())
    val photos: StateFlow<List<MarsPhoto>> get() = _photos

    private val _loading = MutableStateFlow(false)
    val loading: StateFlow<Boolean> get() = _loading

    private val _error = MutableStateFlow<String?>(null)
    val error: StateFlow<String?> get() = _error

    fun fetchPhotos() {
        viewModelScope.launch {
            _loading.value = true
            _error.value = null
            val result = repository.fetchPhotos()
            result.onSuccess { data ->
                _photos.value = data
            }.onFailure { exception ->
```

```
                _error.value = exception.message
            }
            _loading.value = false
        }
    }
}
```

6: Buat UI dengan Compose

Buat file PhotoScreen.kt:

```
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import coil3.compose.AsyncImage
import coil3.request.ImageRequest
import coil3.request.crossfade

@Composable
fun PhotoScreen(viewModel: PhotoViewModel = viewModel()) {
    val photos by viewModel.photos.collectAsState()
    val loading by viewModel.loading.collectAsState()
    val error by viewModel.error.collectAsState()

    Box(modifier = Modifier.fillMaxSize()) {
        when {
            loading -> {
```

```
                Box(
                    modifier = Modifier.fillMaxSize(),
                    contentAlignment = Alignment.Center
                ) {
                    CircularProgressIndicator()
                }
            }

            error != null -> {
                ErrorView(message = error.orEmpty(), onRetry = {
viewModel.fetchPhotos() })
            }

            photos.isNotEmpty() -> {
                PhotosList(photos = photos)
            }

            else -> {
                Column(
                    modifier = Modifier.fillMaxSize(),
                    verticalArrangement = Arrangement.Center,
                    horizontalAlignment = Alignment.CenterHorizontally
                ){
                    Text(
                        text = "No photos available",
                        style = MaterialTheme.typography.bodyLarge,
                    )
                    Button(onClick = { viewModel.fetchPhotos() }) {
                        Text(text = "Retry")
                    }
                }
            }
        }
    }
}

@Composable
fun PhotosList(photos: List<MarsPhoto>) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.padding(top = 40.dp)
    ) {
        Text(
            text = "Mars Photos",
            style = MaterialTheme.typography.bodyLarge
        )
        LazyColumn(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp)
        ) {
            items(photos) { photo ->
                PhotoItem(photo = photo)
            }
```

```
        }
    }
}


@Composable
fun ErrorView(message: String, onRetry: () -> Unit) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(text = "Error: $message", color =
MaterialTheme.colorScheme.error)
        Spacer(modifier = Modifier.height(8.dp))
        Button(onClick = onRetry) {
            Text(text = "Retry")
        }
    }
}

@Composable
fun PhotoItem(photo: MarsPhoto) {
        AsyncImage(
            model = photo.img_src,
            contentDescription = "Mars photo",
            modifier = Modifier
                .fillMaxWidth()
                .height(200.dp)
                .padding(8.dp),
            contentScale = ContentScale.Crop
        )
}
```

7: Integrasikan ke MainActivity

Modifikasi file MainActivity.kt:

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Surface
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import coil3.compose.AsyncImage
```

```
import com.example.internetimage.ui.theme.InternetImageTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            InternetImageTheme {
                Surface(modifier = Modifier.fillMaxSize()) {
                    PhotoScreen()
                }
            }
        }
    }
}
```

8. Tambahkan Izin Internet

Tambahkan izin internet di `AndroidManifest.xml` tepat di atas tag `<application>`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

9. Jalankan aplikasi

Ketika aplikasi dijalankan, data gambar akan dimuat dari API dan ditampilkan sebagai daftar menggunakan **LazyColumn** di Compose.

Coba ganti fungsi `PhotoItem()` dengan `PhotoItem2()` seperti berikut. `ImageRequest.Builder` secara manual digunakan untuk memberikan konfigurasi lebih lanjut.

```
@Composable
fun PhotoItem2(photo: MarsPhoto) {
    AsyncImage(
        model = ImageRequest.Builder(LocalContext.current)
            .data(photo.img_src) // URL gambar
            .crossfade(true) // Aktifkan crossfade
            .build(),
        contentDescription = "Mars photo",
        modifier = Modifier
            .fillMaxWidth()
            .height(200.dp)
            .padding(top=8.dp,bottom=8.dp),
        contentScale = ContentScale.Crop // Atur skala tampilan
    )
}
```

Penjelasan:

1. **Parameter `crossfade(true)`**: Mengaktifkan animasi crossfade. Anda juga bisa menentukan durasinya, misalnya `crossfade(500)` untuk 500ms.
2. **Parameter `contentScale`**: Mengatur bagaimana gambar disesuaikan dengan ruang yang tersedia. Contohnya, `ContentScale.Crop` memotong gambar agar sesuai dengan dimensi.
3. **Modifier**: Digunakan untuk mengatur ukuran dan tata letak gambar.

Jika Anda ingin menambahkan **placeholder** dan **error image**:

```
@Composable
fun PhotoItem2(photo: MarsPhoto) {
    AsyncImage(
        model = ImageRequest.Builder(LocalContext.current)
            .data(photo.img_src) // URL gambar
            .crossfade(true) // Aktifkan crossfade
            .build(),
        placeholder = painterResource(R.drawable.loading_img), // Gambar sementara
        error = painterResource(R.drawable.ic_broken_image), // Gambar saat error
        contentDescription = "Mars photo",
        modifier = Modifier
            .fillMaxWidth()
            .height(200.dp)
            .padding(top=8.dp,bottom=8.dp),
        contentScale = ContentScale.Crop // Atur skala tampilan
    )
}
```