МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №4
“Объекто-ориентированное программирование на Kotlin ”

Выполнил:
Студент группы ИУ5-34Б
Жеребенков А.Ю.
Преподаватель:
Нардид А.Н.

Москва 2025

Задание

- Требуется разработать небольшое приложение, которое отображает плашки с изображениями, полученными от API.
- Приложение должно уметь обрабатывать состояние загрузки данных. В примитивном варианте - выводить крутилку по центру экрана.
- Если есть догрузка данных - то отображать плашку с крутилкой.
- Приложение должно уметь обрабатывать ошибки загрузки - отображать заглушку, которая позволит повторить запрос.
- Необходимо также реализовать кеширование данных
- При клике на изображение необходимо создавать уведомление с информацией о картинке (порядковый номер в списке)

Требования:

- Состояния интерфейса:
- Стартовая загрузка: индикатор прогресса по центру
- Пагинация: индикатор внизу списка
- Отображение контента
- Ошибки: заглушка с кнопкой повтора
- Пагинация с бесконечным скроллом и индикатором прогресса внизу страницы при загрузке
- Техническая реализация:
- Сохранение данных при повороте экрана
- Динамическое масштабирование изображений (сохранение пропорций)
- Обязательно: Fragment/Compose
- Кэширование данных и изображений
- Ограничения:
- Обязательно: использование ресурсов, отсутствие хардкода

- Запрещены логи (Log.*) в финальной версии
-

Плюсы:

- Поддержка анимированных изображений (GIF)
- Pinterest-стиль (разные пропорции карточек)
-

Рекомендуемые API:

- Анимированные GIF: Giphy API
- Github of Public APIs
- Любое API, которое интересно использовать и к которому может получить доступ преподаватель

Код(MainActivity.kt)

```
package com.example.gifapi

import android.os.Build.VERSION.SDK_INT
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.staggeredgrid.LazyVerticalStaggeredGrid
import androidx.compose.foundation.lazy.staggeredgrid.StaggeredGridCells
import androidx.compose.foundation.lazy.staggeredgrid.StaggeredGridItemSpan
import androidx.compose.foundation.lazy.staggeredgrid.items
import androidx.compose.foundation.lazy.staggeredgrid.rememberLazyStaggeredGridState
import androidx.compose.material3.*
import androidx.compose.runtime.*
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.stringResource
import androidx.compose.ui.unit.dp
```

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.lifecycle.viewModelScope
import coil.compose.AsyncImage
import coil.request.CachePolicy
import coil.request.ImageRequest
import kotlinx.coroutines.launch
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.http.GET
import retrofit2.http.Query
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import androidx.compose.ui.text.style.TextOverflow
import coil.ImageLoader
import coil.decode.GifDecoder
import coil.decode.ImageDecoderDecoder
import com.example.gifapi.BuildConfig
import com.zherebenkoff.gifapi.ui.theme.SimpleGiphyTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SimpleGiphyTheme {
                val viewModel: MainViewModel = viewModel(factory = MainViewModelFactory())
                MainScreen(viewModel)
            }
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class, ExperimentalFoundationApi::class)
@Composable
fun MainScreen(viewModel: MainViewModel) {
    val state by viewModel.state.collectAsState()
    val context = LocalContext.current

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(stringResource(R.string.app_bar_title)) },
                colors = TopAppBarDefaults.topAppBarColors(
                    containerColor = MaterialTheme.colorScheme.primaryContainer,
                    titleContentColor = MaterialTheme.colorScheme.background
                )
            )
        },
        content = { padding ->
            Box(
                modifier = Modifier
```

```
            .fillMaxSize()
            .padding(padding)
            .background(MaterialTheme.colorScheme.background)
    ) {
        when {
            state.isLoading && state.images.isEmpty() -> {
                Box(
                    modifier = Modifier.fillMaxSize(),
                    contentAlignment = Alignment.Center
                ) {
                    Column(
                        horizontalAlignment = Alignment.CenterHorizontally,
                        verticalArrangement = Arrangement.spacedBy(16.dp)
                    ) {
                        CircularProgressIndicator(
                            modifier = Modifier.size(48.dp),
                            color = MaterialTheme.colorScheme.primary
                        )
                        Text(
                            stringResource(R.string.loading_images),
                            color = MaterialTheme.colorScheme.onBackground
                        )
                    }
                }
            }
            state.error != null && state.images.isEmpty() -> {
                Column(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(32.dp),
                    horizontalAlignment = Alignment.CenterHorizontally,
                    verticalArrangement = Arrangement.Center
                ) {
                    Text(
                        text = state.error ?: stringResource(R.string.error_unknown),
                        color = MaterialTheme.colorScheme.error,
                        style = MaterialTheme.typography.bodyLarge
                    )
                    Spacer(Modifier.height(16.dp))
                    Button(
                        onClick = { viewModel.retry() },
                        colors = ButtonDefaults.buttonColors(
                            containerColor = MaterialTheme.colorScheme.primary
                        )
                    ) {
                        Text(stringResource(R.string.retry_button))
                    }
                }
            }
            else -> {
                LazyVerticalStaggeredGrid(
                    columns = StaggeredGridCells.Adaptive(minSize = 150.dp),
                    modifier = Modifier.fillMaxSize(),
```

```
        verticalItemSpacing = 8.dp,
        horizontalArrangement = Arrangement.spacedBy(8.dp),
        contentPadding = PaddingValues(8.dp),
        state = rememberLazyStaggeredGridState()
    ) {
        items(state.images) { image ->
            PinterestImageCard(
                image = image,
                onClick = {
                    val index = state.images.indexOf(image) + 1
                    Toast.makeText(
                        context,
                        context.getString(
                            R.string.toast_image,
                            index,
                            image.title
                        ),
                        Toast.LENGTH_SHORT
                    ).show()
                }
            )
        }

        if (state.isLoadingMore) {
            item(span = StaggeredGridItemSpan.FullLine) {
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(16.dp)
                        .wrapContentWidth(Alignment.CenterHorizontally)
                ) {
                    CircularProgressIndicator(
                        color = MaterialTheme.colorScheme.primary
                    )
                }
            }
        }

        if (state.canLoadMore && !state.isLoadingMore) {
            item(span = StaggeredGridItemSpan.FullLine) {
                Button(
                    onClick = { viewModel.loadMore() },
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(16.dp),
                    colors = ButtonDefaults.buttonColors(
                        containerColor = MaterialTheme.colorScheme.primary
                    )
                ) {
                    Text(stringResource(R.string.load_more_button))
                }
            }
        }
```

```
                }
            }
        }
    }
  )
}

@Composable
fun PinterestImageCard(image: Image, onClick: () -> Unit) {
  Card(
    modifier = Modifier
      .fillMaxWidth()
      .clickable(onClick = onClick),
    shape = MaterialTheme.shapes.medium,
    elevation = CardDefaults.cardElevation(defaultElevation = 2.dp),
    colors = CardDefaults.cardColors(
      containerColor = MaterialTheme.colorScheme.surfaceVariant
    )
  ) {
    Column {
//        val gifEnabledLoader = ImageLoader.Builder(this)
//            . {
//                if ( SDK_INT >= 28 ) {
//                    add(ImageDecoderDecoder.Factory())
//                } else {
//                    add(GifDecoder.Factory())
//                }
//            }.build()
      AsyncImage(
        model = ImageRequest.Builder(LocalContext.current)
          .data(image.imageUrl)
          .crossfade(true)
          .memoryCachePolicy(CachePolicy.ENABLED)
          .diskCachePolicy(CachePolicy.ENABLED)
          .build(),
        contentDescription = image.title,
        contentScale = ContentScale.Crop,
        modifier = Modifier
          .fillMaxWidth()
          .aspectRatio(image.aspectRatio)

      )

      if (image.title.isNotBlank() && image.title != " ") {
        Text(
          text = image.title,
          modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth(),
          maxLines = 2,
          overflow = TextOverflow.Ellipsis,
          style = MaterialTheme.typography.bodySmall,
```

```
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        }
    }
}

data class Image(
    val id: String,
    val title: String,
    val imageUrl: String,
    val width: Int,
    val height: Int,
    val aspectRatio: Float = if (height > 0) width.toFloat() / height.toFloat() else 1.5f
)

data class GiphyResponse(val data: List<GiphyItem>, val pagination: Pagination)
data class GiphyItem(
    val id: String,
    val title: String,
    val images: Images
)
data class Images(
    val fixed_height: ImageData? = null,
    val original: ImageData
)
data class ImageData(val url: String, val width: String, val height: String)
data class Pagination(val total_count: Int, val count: Int, val offset: Int)

interface GiphyApi {
    @GET("v1/gifs/trending")
    suspend fun getTrendingGifs(
        @Query("api_key") apiKey: String,
        @Query("limit") limit: Int = 20,
        @Query("offset") offset: Int = 0,
        @Query("rating") rating: String = "g",
        @Query("mime_types") mimeTypes: String = "gif"
    ): GiphyResponse
}

object RetrofitClient {
    private const val BASE_URL = "https://api.giphy.com/"
    val giphyApi: GiphyApi by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(GiphyApi::class.java)
    }
}

data class MainViewState(
    val images: List<Image> = emptyList(),
```

```
    val isLoading: Boolean = false,
    val isLoadingMore: Boolean = false,
    val error: String? = null,
    val canLoadMore: Boolean = true,
    val totalCount: Int = 0,
    val loadedCount: Int = 0
)

class MainViewModel : ViewModel() {
    private var currentOffset = 0
    private val pageSize = 20
    private val _state = MutableStateFlow(MainViewState(isLoading = true))
    val state: StateFlow<MainViewState> = _state.asStateFlow()

    private val cachedImages = mutableListOf<Image>()

    init {
        loadFirstPage()
    }

    private fun loadFirstPage() {
        currentOffset = 0
        cachedImages.clear()
        loadImages(0)
    }

    fun loadMore() {
        if (!state.value.isLoadingMore && state.value.canLoadMore) {
            loadImages(currentOffset)
        }
    }

    private fun loadImages(offset: Int) {
        viewModelScope.launch {
            try {
                _state.value = _state.value.copy(
                    isLoading = offset == 0,
                    isLoadingMore = offset > 0,
                    error = null
                )

                val response = RetrofitClient.giphyApi.getTrendingGifs(
                    apiKey = BuildConfig.GIPHY_API_KEY,
                    limit = pageSize,
                    offset = offset,
                    rating = "g"
                )

                val newImages = response.data.map { item ->
                    val imageData = item.images.fixed_height ?: item.images.original
                    Image(
                        id = item.id,
                        title = item.title.ifBlank {
```

```
                        "Без названия"
                    },
                    imageUrl = imageData.url,
                    width = imageData.width.toIntOrNull() ?: 200,
                    height = imageData.height.toIntOrNull() ?: 200
                )
            }

            if (offset == 0) {
                cachedImages.clear()
                cachedImages.addAll(newImages)
            } else {
                cachedImages.addAll(newImages)
            }

            currentOffset = offset + pageSize
            val canLoadMore = newImages.isNotEmpty() &&
                    (cachedImages.size < response.pagination.total_count)

            _state.value = _state.value.copy(
                images = cachedImages.toList(),
                isLoading = false,
                isLoadingMore = false,
                canLoadMore = canLoadMore,
                totalCount = response.pagination.total_count,
                loadedCount = cachedImages.size
            )
        } catch (e: Exception) {
            _state.value = _state.value.copy(
                isLoading = false,
                isLoadingMore = false,
                error = when {
                    BuildConfig.GIPHY_API_KEY.isEmpty() ->
                        "API ключ не настроен. Проверьте local.properties"
                    e.message?.contains("Unable to resolve host") == true ->
                        "Нет подключения к интернету"
                    else -> "Ошибка загрузки"
                }
            )
        }
    }
}

fun retry() {
    if (state.value.error != null) {
        if (currentOffset == 0) {
            loadFirstPage()
        } else {
            loadMore()
        }
    }
}
}
```

```
class MainViewModelFactory : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(MainViewModel::class.java)) {
            @Suppress("UNCHECKED_CAST")
            return MainViewModel() as T
        }
        throw IllegalArgumentException("Unknown ViewModel")
    }
}
```

Код SimpleTheme.kt

```
package com.zherebenkoff.gifapi.ui.theme

import androidx.compose.ui.graphics.Color
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable

private val DarkColorScheme = darkColorScheme(
    primary = Color(0xFFD32F2F),
    secondary = Color(0xFFD32F2F),
    tertiary = Color(0xFFD32F2F),
    background = Color.Black,
    surface = Color.Black,
    onPrimary = Color.White,
    onSecondary = Color.White,
    onTertiary = Color.White,
    onBackground = Color.White,
    onSurface = Color.White,
    primaryContainer = Color(0xFFD32F2F),
    secondaryContainer = Color(0xFFD32F2F)
)

private val LightColorScheme = lightColorScheme(
    primary = Color(0xFFD32F2F),
    secondary = Color(0xFFD32F2F),
    tertiary = Color(0xFFD32F2F),
    background = Color.White,
    surface = Color.White,
    onPrimary = Color.White,
    onSecondary = Color.White,
    onTertiary = Color.White,
    onBackground = Color.Black,
    onSurface = Color.Black,
    primaryContainer = Color(0xFFD32F2F),
    secondaryContainer = Color(0xFFD32F2F)
)
```

```
@Composable
fun SimpleGiphyTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit
) {
    val colorScheme = if (darkTheme) DarkColorScheme else LightColorScheme

    MaterialTheme(
        colorScheme = colorScheme,
        content = content
    )
}
```

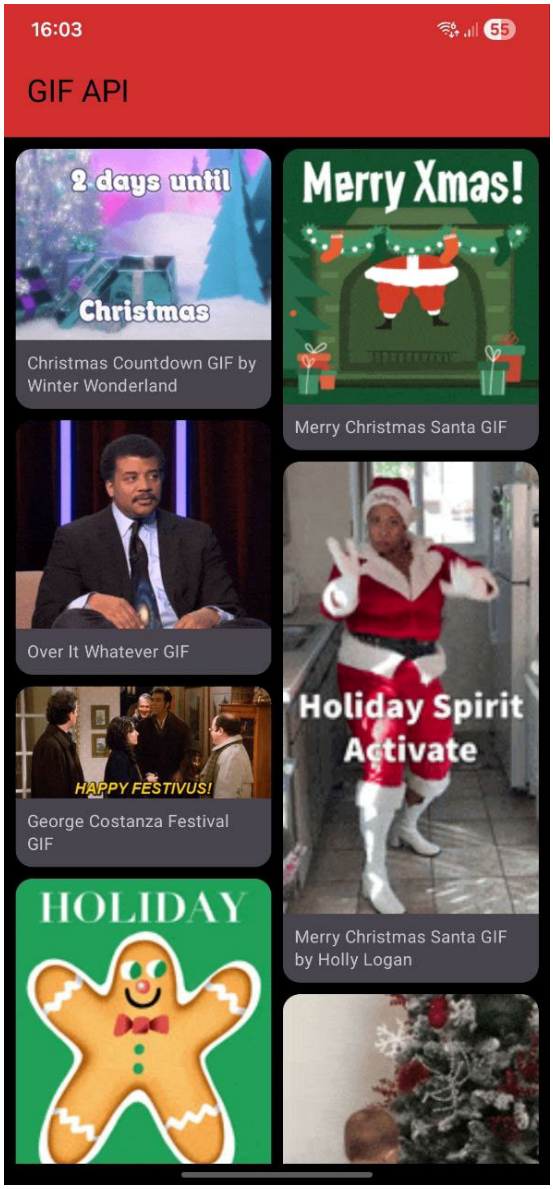
Код build.gradle.kts

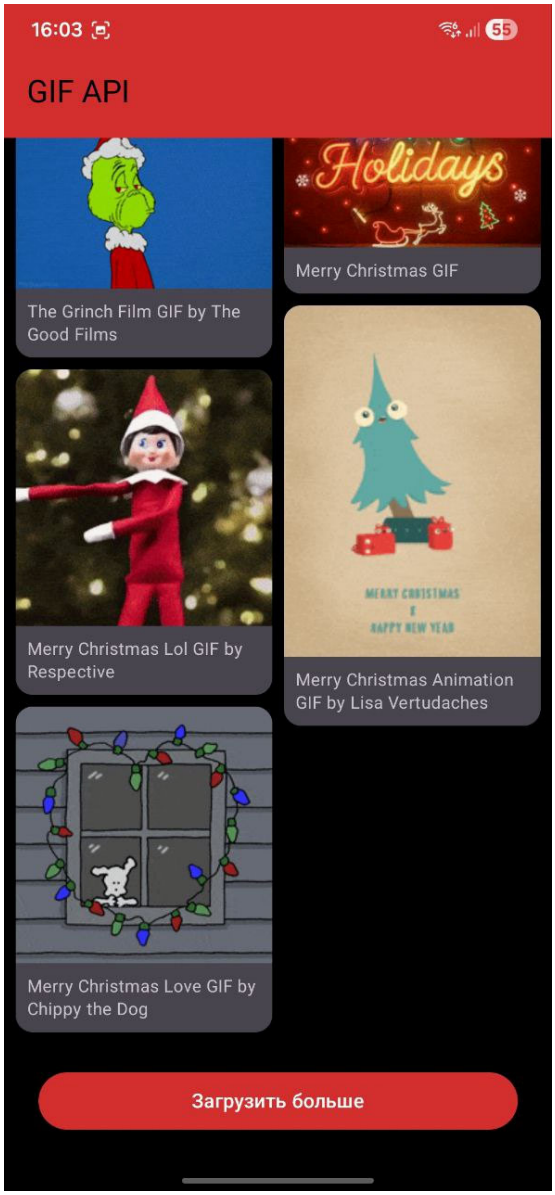
```
import java.util.Properties

plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("org.jetbrains.kotlin.plugin.compose") version "2.0.21"
    id("kotlin-kapt")
}

android {
    namespace = "com.example.gifapi"
    compileSdk = 35

    defaultConfig {
        applicationId = "com.example.gifapi"
        minSdk = 26
        targetSdk = 35
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"

        buildConfigField("String", "GIPHY_API_KEY", "\"${getApiKey(project)}\"")
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }

    compileOptions {
```

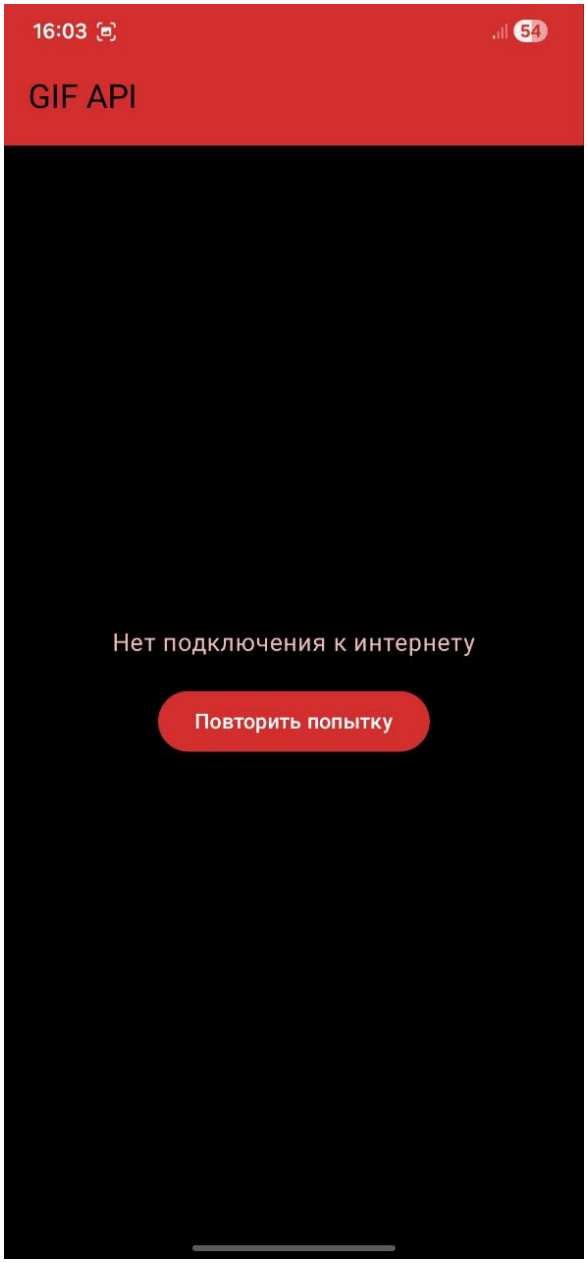
```
        sourceCompatibility = JavaVersion.VERSION_17
        targetCompatibility = JavaVersion.VERSION_17
    }

    kotlinOptions {
        jvmTarget = "17"
    }

    buildFeatures {
        compose = true
        buildConfig = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.5.14"
    }
}

fun getApiKey(project: Project): String {
    val propertiesFile = project.rootProject.file("local.properties")
    return if (propertiesFile.exists()) {
        Properties().apply {
            propertiesFile.inputStream().use { inputStream ->
                load(inputStream)
            }
        }.getProperty("giphy.api.key", "")
    } else {
        ""
    }
}

dependencies {
    implementation("androidx.core:core-ktx:1.13.1")
    implementation("androidx.appcompat:appcompat:1.7.0")
    implementation("com.google.android.material:material:1.12.0")
    implementation("androidx.activity:activity-compose:1.9.2")

    // Compose BOM
    implementation(platform("androidx.compose:compose-bom:2024.11.00"))
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-graphics")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.compose.material3:material3")

    // ViewModel + Compose
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.6")

    // Retrofit
    implementation("com.squareup.retrofit2:retrofit:2.11.0")
    implementation("com.squareup.retrofit2:converter-gson:2.11.0")

    // Coil (GIF + изображения)
    implementation("io.coil-kt:coil-compose:2.6.0")
```

```
    implementation("io.coil-kt:coil-gif:2.6.0")

    implementation("androidx.compose.animation:animation-graphics:1.6.7")

    implementation("androidx.compose.foundation:foundation-android:1.7.6")
    implementation("com.google.accompanist:accompanist-coil:0.15.0")
    implementation(libs.androidx.compose.ui.text) // Для GIF в Compose


    // Тесты
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.2.1")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.6.1")
    androidTestImplementation(platform("androidx.compose:compose-bom:2024.11.00"))
    androidTestImplementation("androidx.compose.ui:ui-test-junit4")
    debugImplementation("androidx.compose.ui:ui-tooling")
    debugImplementation("androidx.compose.ui:ui-test-manifest")
}
```

Работа приложения






16:03
54
GIF API
Нет подключения к интернету
Повторить попытку