Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №5

«Работа со списком и сетью на языке kotlin»

Выполнил:
студент группы РТ5-31Б:
Паншин М.В.

Проверил:
преподаватель кафедры ИУ5
Гапанюк Ю.Е.

Москва, 2024 г.

Постановка задачи

В качестве домашнего задания предлагается выполнить проект «Вопросы и Ответы». Этот сервис позволит пользователям Интернета задавать вопросы и получать на них ответы. Возможности комментирования и голосования формируют сообщество и позволяет пользователям.

Требования к проекту

1. Структура проекта должна быть понятна пользователям. Переходы по страницам осуществляются по ссылкам. Обработка форм должна осуществляться с редиректом.
2. Код проекта должен быть аккуратным и без дублирования.
3. Верстка проекта должна быть выполнена с помощью css фреймворка Twitter Bootstrap.
4. Страницы проекты не должны отдаваться более 1 секунды.

Текст программы

Файл MainActivity.kt

```
package com.example.panshin_homework


import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember


class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            val imageViewModel =
ImageViewModel(RetrofitController("https://api.unsplash.com/"))
            MainApp(imageViewModel)
```

```
        }

    }


}
```

Файл ImageView.kt

```
package com.example.panshin_homework
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope

import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch


import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.lazy.rememberLazyListState

import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator


import androidx.compose.material3.Text

import androidx.compose.runtime.*

import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.runtime.collectAsState

import androidx.compose.ui.Alignment
import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.platform.LocalContext

import coil.compose.AsyncImage
import coil.request.ImageRequest

import kotlinx.coroutines.flow.asStateFlow

import kotlinx.coroutines.flow.update




class ImageViewModel(
    private val requestController: RequestController
) : ViewModel() {
```

```
    private val _images = MutableStateFlow<List<ImageItem>>(emptyList())
    val images: StateFlow<List<ImageItem>> = _images.asStateFlow()

    private val _isLoading = MutableStateFlow(false)
    val isLoading: StateFlow<Boolean> = _isLoading.asStateFlow()

    private val _page = MutableStateFlow(1)
    val perPage = 5

    private val _error = MutableStateFlow<String?>(null)
    val error: StateFlow<String?> = _error
    init {
        loadNextPage()
    }
    fun loadNextPage() {
        if(_isLoading.value) return

        viewModelScope.launch {
            _isLoading.value = true
            try {
                //Запустить на IO потоке
                //val result =  requestController.requestImage(perPage)
                val result =
requestController.requestImage(_page.value,perPage)
                if(result is Result.Ok){
                    _images.update { currentImages ->
                        (currentImages + result.images).distinctBy {
it.urls.rawUrl } }
                    _page.value++
                }
                if (result is Result.Error){
                    _error.value = result.error

                }
            }
            catch (e: Exception){
                _error.value = e.message?: "Network Error"
            }
            finally {
                _isLoading.value = false
            }

        }
    }
}
@Composable
fun MainScreen(imageViewModel: ImageViewModel, onImageClick: (ImageItem) ->
Unit) {

//    var images = rememberSaveable(stateSaver = listSaver<ImageItem,
String>(
//        save = {list: List<ImageItem> -> list.map(it.urls.rawUrl)},
//        restore = {urlist -> urlist.map {ImageItem(Origin(it)) }}
//    )) {mutableStateListOf()}

    val images by imageViewModel.images.collectAsState()

    val isLoading by imageViewModel.isLoading.collectAsState()
    val error by imageViewModel.error.collectAsState()
    val lazyListState = rememberLazyListState()
```

```
    Box(modifier = Modifier.fillMaxSize(), contentAlignment =
Alignment.Center) {
        if (images.isEmpty()) {
            if (isLoading) {
                CircularProgressIndicator()
            } else if (error != null) {
                Column(horizontalAlignment = Alignment.CenterHorizontally) {
                    Text("Error:$error")
                    Button(onClick = { imageViewModel.loadNextPage() }) {
Text("Retry") }
                }

            } else {
                Text("No data")
            }

        } else {
            LazyColumn(
                state = lazyListState,
                modifier = Modifier.fillMaxSize()
            ) {
                items(items = images, key = { it.urls.rawUrl }) { image ->
                    ImageItemCard(image, {onImageClick(image)})
                }
                item {
                    if (!isLoading && error != null){
                        Column(modifier = Modifier
                            .fillMaxWidth()
                            .padding(16.dp),
                            horizontalAlignment =
Alignment.CenterHorizontally){
                            Text ("Error: $error")
                            Spacer(Modifier.height(8.dp))
                            Button(onClick = {imageViewModel.loadNextPage()})
{
                                Text("Retry")
                            }
                        }
                    }
                    else if (isLoading && error != null) {
                        Column(
                            modifier = Modifier
                                .fillMaxWidth()
                                .padding(16.dp),
                            horizontalAlignment =
Alignment.CenterHorizontally
                        ) { CircularProgressIndicator() }
                    }
                }
            }
            LaunchedEffect(lazyListState) {
                snapshotFlow {
lazyListState.layoutInfo.visibleItemsInfo.lastOrNull()?.index }
                    .collect { lastVisibleItemIndex ->
                        if (lastVisibleItemIndex != null &&
                            lastVisibleItemIndex >= images.size - 2 &&
                            !isLoading
                        ) {
                            imageViewModel.loadNextPage()
                        }
                    }
            }
        }
        LaunchedEffect(Unit) {
```

```
            imageViewModel.loadNextPage()
        }


    }
}



@Composable
fun ImageItemCard(image: ImageItem, onImageClick: (ImageItem) -> Unit) {
    Column(modifier = Modifier.padding(8.dp).clickable{onImageClick(image)})
{


        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(image.urls.rawUrl)
                .addHeader(
                    "Authorization",
                    "Client-ID KkEwtM9jsHnAgL2tukfUW0ywNj900soc99VGKes4MxE"
                )

                .crossfade(true)
                .build(),
            contentDescription = "Image",
            modifier = Modifier
                .fillMaxWidth()
                .aspectRatio(1.5f)
                .padding(4.dp),
            contentScale = ContentScale.Crop

        )
    }

}


@Composable
fun MainApp (viewModel: ImageViewModel){
    var selectedImage by remember { mutableStateOf<ImageItem?>(null)}

    if (selectedImage == null){
        MainScreen(viewModel){
                image -> selectedImage = image
        }
    }
    else{
        OneImageCard(selectedImage!!){
            selectedImage = null
        }
    }
}
@Composable
fun OneImageCard(image: ImageItem,
                 onImageClick: (ImageItem) -> Unit) {
    Box(modifier = Modifier.fillMaxSize().clickable(){onImageClick(image)}
        .background(Color.Black)) {
        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(image.urls.rawUrl)
                .addHeader(
                    "Authorization",
                    "Client-ID KkEwtM9jsHnAgL2tukfUW0ywNj900soc99VGKes4MxE"
```

```
                )
                .crossfade(true)
                .build(),
            contentDescription = "Image",
            modifier = Modifier
                .fillMaxWidth()
                .aspectRatio(1.5f)
                .padding(4.dp),
            contentScale = ContentScale.Crop

        )
    }
}
```

ФайлRetrofitController.kt

```
package com.example.panshin_homework

import
com.jakewharton.retrofit2.converter.kotlinx.serialization.asConverterFactory
import kotlinx.serialization.json.Json
import okhttp3.MediaType.Companion.toMediaType
import retrofit2.Retrofit

import retrofit2.Response
import retrofit2.http.GET
import retrofit2.http.Query


interface ImageApi {
    @GET("photos/random/?client_id=X79-
CUmynexhT219vf3CE69iYjKZZXStAX82PqIlsUM")

    suspend fun getImages(

        @Query("count") perPage: Int
    ): Response<List<ImageItem>>
}

class RetrofitController(api: String) : RequestController {

    private val retrofit = Retrofit.Builder()
        .baseUrl(api)
        .addConverterFactory(
            Json { ignoreUnknownKeys = true }
                .asConverterFactory(
                    "application/json; charset=UTF8".toMediaType()
                )
        )
        .build()

    private val imageApi = retrofit.create(ImageApi::class.java)

    override suspend fun requestImage(page: Int,perPage: Int): Result {
        //val response = imageApi.getImages(page,perPage)
        val response = imageApi.getImages(perPage)
        return if (response.isSuccessful) {
            response.body()?.let {images ->
                Result.Ok(images)
```

```
            } ?: Result.Error("Empty images")
        } else {
            Result.Error(response.code().toString())
        }
    }
}
```

Файл RequestController.kt

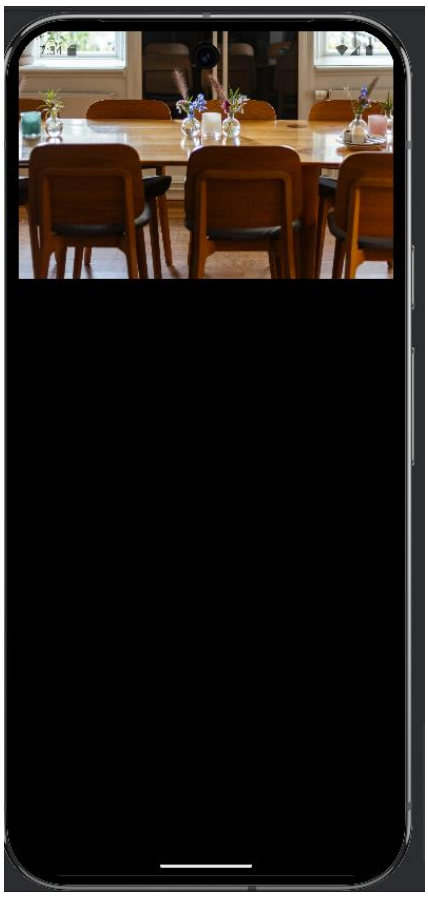
```
package com.example.panshin_homework


import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable


interface RequestController {

    //suspend fun requestImage(perPage: Int): Result
    suspend fun requestImage(page: Int, perPage: Int): Result
}

sealed interface Result {
    data class Ok(val images: List<ImageItem>) : Result
    data class Error(val error: String) : Result
}

@kotlinx.serialization.Serializable
data class Origin(
    @SerialName("raw") val rawUrl:String
)
@kotlinx.serialization.Serializable
data class ImageItem(
    @SerialName("urls") val urls: Origin
)
```

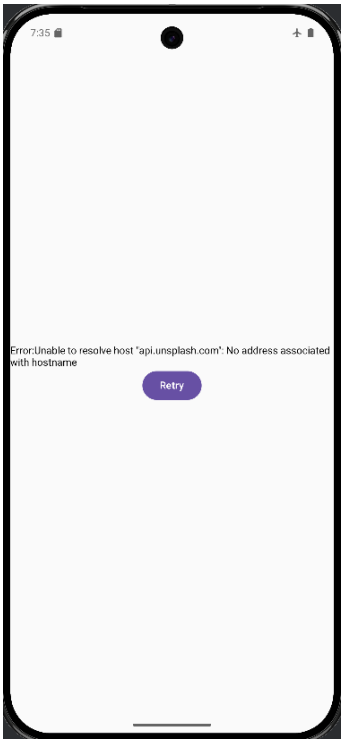
Результат работы программы

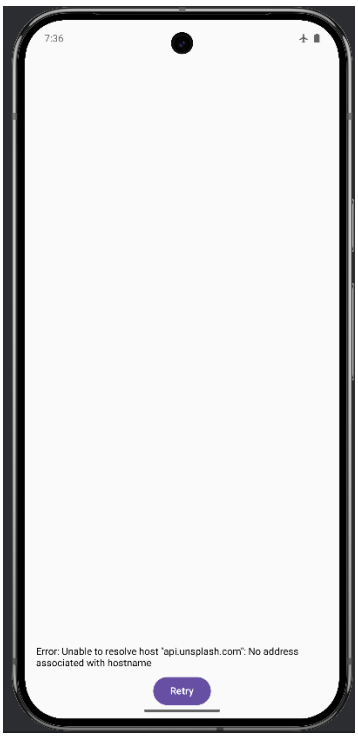
Приведены скриншоты в случаях:

1. Корректной работы
2. Вывода определенной картинки
3. Обработка отсутствия интернет подключения при запуске приложения

4. Обработка отсутствия интернет подключения во время работы приложения




7:35
Error:Unable to resolve host "api.unsplash.com": No address associated with hostname
Retry


7:36
Error: Unable to resolve host "api.unsplash.com": No address associated with hostname
Retry