Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование мобильных систем

Лабораторная работа №1

«Аналог Pinterest»

Выполнил:

студент группы 250502

Потейчук О.В.

Проверил:

Внук О.М.

МИНСК 2024

**1 ПОСТАНОВКА ЗАДАЧИ**

Разработать страницы и элементы пользовательского интерфейса. Описать реализуемый функционал для разрабатываемого приложения.

**2 СТРУКТУРА ПРОЕКТА**

В проекте используется сlean-архитектура, состоящая из app-модуля, содержащего в себе presentation-слой и управление dependency injection, domain-модуля, включающего use cases, models и интерфейсы для репозиториев, и data-модуля, состоящего из реализаций репозиториев, dtos и mappers.

A screenshot of a computer

Description automatically generated

Рисунок 2.1 – Структура app-модуля

A screenshot of a computer program

Description automatically generated

Рисунок 2.2 – Структура data и domain модулей

**3 ЛИСТИНГ КОДА**

package com.example.innowisepexelstestapp.di.module  
  
import com.github.terrakok.cicerone.Cicerone  
import com.github.terrakok.cicerone.NavigatorHolder  
import com.github.terrakok.cicerone.Router  
import dagger.Module  
import dagger.Provides  
import javax.inject.Singleton  
  
@Module  
class AppCiceroneModule {  
 private val cicerone: Cicerone<Router> = Cicerone.create()  
  
 @Singleton  
 @Provides  
 fun provideRouter(): Router = cicerone.router  
  
 @Singleton  
 @Provides  
 fun provideNavigatorHolder(): NavigatorHolder = cicerone.getNavigatorHolder()  
}

package com.example.innowisepexelstestapp.di.module  
  
import android.content.Context  
import com.example.innowisepexelstestapp.util.ResourceProvider  
import dagger.Module  
import dagger.Provides  
import javax.inject.Singleton  
  
@Module  
class AppModule {  
  
 @Singleton  
 @Provides  
 fun **provideResourceProvider**(mAppContext: Context): ResourceProvider {  
 return ResourceProvider(mAppContext)  
 }  
}

package com.example.innowisepexelstestapp.di.module  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.ViewModelProvider  
import com.example.innowisepexelstestapp.di.ViewModelFactory  
import com.example.innowisepexelstestapp.di.ViewModelKey  
import com.example.innowisepexelstestapp.presentation.viewmodel.DetailsViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.FavoriteViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.HomeViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.MainViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.SignInViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.SignUpViewModel  
import dagger.Binds  
import dagger.Module  
import dagger.multibindings.IntoMap  
  
@Module  
interface AppViewModelModule {  
  
 @Binds  
 @IntoMap  
 @ViewModelKey(HomeViewModel::class)  
 fun **homeViewModel**(viewModel: HomeViewModel): ViewModel  
  
 @Binds  
 @IntoMap  
 @ViewModelKey(DetailsViewModel::class)  
 fun **detailsViewModel**(viewModel: DetailsViewModel): ViewModel  
  
 @Binds  
 @IntoMap  
 @ViewModelKey(FavoriteViewModel::class)  
 fun **favoriteViewModel**(viewModel: FavoriteViewModel): ViewModel  
  
 @Binds  
 @IntoMap  
 @ViewModelKey(MainViewModel::class)  
 fun **mainViewModel**(viewModel: MainViewModel): ViewModel  
  
 @Binds  
 @IntoMap  
 @ViewModelKey(SignInViewModel::class)  
 fun **signInViewModel**(viewModel: SignInViewModel): ViewModel  
  
 @Binds  
 @IntoMap  
 @ViewModelKey(SignUpViewModel::class)  
 fun **signUpViewModel**(viewModel: SignUpViewModel): ViewModel  
  
 @Binds  
 fun **bindViewModelFactory**(factory: ViewModelFactory): ViewModelProvider.Factory  
}

package com.example.innowisepexelstestapp.di.module  
  
import com.example.innowisepexelstestapp.repository.NetworkManager  
import com.example.innowisepexelstestapp.repository.pexelsapi.NetworkManagerImpl  
import com.example.innowisepexelstestapp.repository.pexelsapi.PexelsNetworkClient  
import dagger.Module  
import dagger.Provides  
import okhttp3.OkHttpClient  
import javax.inject.Singleton  
  
@Module  
class DataNetworkModule {  
  
 @Singleton  
 @Provides  
 fun **provideOkHttpClient**(): OkHttpClient {  
 return OkHttpClient()  
 }  
  
 @Singleton  
 @Provides  
 fun **providePexelsNetworkClient**(okHttpClient: OkHttpClient): PexelsNetworkClient {  
 return PexelsNetworkClient(okHttpClient)  
 }  
  
}

package com.example.innowisepexelstestapp.di.module  
  
import android.content.Context  
import androidx.room.Room  
import com.example.innowisepexelstestapp.repository.room.PhotoPexelsDao  
import com.example.innowisepexelstestapp.repository.room.FavoritePhotoDataBase  
  
import dagger.Module  
import dagger.Provides  
import javax.inject.Singleton  
  
@Module  
class DataRoomModule {  
  
 @Singleton  
 @Provides  
 fun **providePhotoPexelsDao**(mAppContext: Context): PhotoPexelsDao {  
 return Room.databaseBuilder(  
 mAppContext, FavoritePhotoDataBase::class.*java*, "roomDBCategories"  
 ).allowMainThreadQueries().build().photoPexelsDao()  
 }  
  
}

package com.example.innowisepexelstestapp.di.module  
  
import android.content.Context  
import com.example.innowisepexelstestapp.repository.DownloadFilesManager  
import com.example.innowisepexelstestapp.repository.FavoritePhotoManager  
import com.example.innowisepexelstestapp.repository.NetworkManager  
import com.example.innowisepexelstestapp.repository.SignInSignUpManager  
import com.example.innowisepexelstestapp.repository.downloadmanager.DownloadFilesManagerImpl  
import com.example.innowisepexelstestapp.repository.firebase.FirebaseFavoritePhotoManagerImpl  
import com.example.innowisepexelstestapp.repository.firebase.SignInSignUpManagerImpl  
import com.example.innowisepexelstestapp.repository.pexelsapi.NetworkManagerImpl  
import com.example.innowisepexelstestapp.repository.pexelsapi.PexelsNetworkClient  
import com.example.innowisepexelstestapp.repository.room.RoomFavoritePhotoManagerImpl  
import com.example.innowisepexelstestapp.repository.room.PhotoPexelsDao  
import dagger.Module  
import dagger.Provides  
import javax.inject.Named  
import javax.inject.Singleton  
  
const val *ROOM\_DB* = "room"  
const val *FIREBASE\_DB* = "firebase"  
  
@Module  
class DomainRepositoryModule {  
  
 @Singleton  
 @Provides  
 fun **provideNetworkManagerImpl**(networkClient: PexelsNetworkClient): NetworkManager {  
 return NetworkManagerImpl(networkClient)  
 }  
  
 @Singleton  
 @Provides  
 @Named(*ROOM\_DB*)  
 fun **provideRoomFavoritePhotoManager**(photoPexelsDao: PhotoPexelsDao): FavoritePhotoManager {  
 return RoomFavoritePhotoManagerImpl(photoPexelsDao)  
 }  
  
 @Singleton  
 @Provides  
 @Named(*FIREBASE\_DB*)  
 fun **provideFirebaseFavoritePhotoManager**(): FavoritePhotoManager {  
 return FirebaseFavoritePhotoManagerImpl()  
 }  
  
 @Singleton  
 @Provides  
 fun **provideDownloadFilesManager**(mAppContext: Context): DownloadFilesManager {  
 return DownloadFilesManagerImpl(mAppContext)  
 }  
  
 @Singleton  
 @Provides  
 fun **provideSignInSignUpManager**(): SignInSignUpManager {  
 return SignInSignUpManagerImpl()  
 }  
  
}

package com.example.innowisepexelstestapp.di.module  
  
import com.example.innowisepexelstestapp.repository.DownloadFilesManager  
import com.example.innowisepexelstestapp.repository.FavoritePhotoManager  
import com.example.innowisepexelstestapp.repository.SignInSignUpManager  
import com.example.innowisepexelstestapp.usecase.DeleteImageFromBdUseCase  
import com.example.innowisepexelstestapp.usecase.DownloadImageUseCase  
import com.example.innowisepexelstestapp.usecase.GetImagesFromBdUseCase  
import com.example.innowisepexelstestapp.usecase.SaveImageIntoBdUseCase  
import com.example.innowisepexelstestapp.usecase.SignInUseCase  
import com.example.innowisepexelstestapp.usecase.SignUpUseCase  
import dagger.Module  
import dagger.Provides  
import javax.inject.Named  
  
@Module  
class DomainUseCaseModule {  
  
 @Provides  
 fun **provideSaveImageIntoBdUseCase**(@Named(*FIREBASE\_DB*) favoritePhotoManager: FavoritePhotoManager): SaveImageIntoBdUseCase {  
 return SaveImageIntoBdUseCase(favoritePhotoManager)  
 }  
  
 @Provides  
 fun **provideDeleteImageFromBdUseCase**(@Named(*FIREBASE\_DB*) favoritePhotoManager: FavoritePhotoManager): DeleteImageFromBdUseCase {  
 return DeleteImageFromBdUseCase(favoritePhotoManager)  
 }  
  
 @Provides  
 fun **provideGetImagesFromBdUseCase**(@Named(*FIREBASE\_DB*) favoritePhotoManager: FavoritePhotoManager): GetImagesFromBdUseCase {  
 return GetImagesFromBdUseCase(favoritePhotoManager)  
 }  
  
 @Provides  
 fun **provideDownloadImageUseCase**(downloadFilesManager: DownloadFilesManager): DownloadImageUseCase {  
 return DownloadImageUseCase(downloadFilesManager)  
 }  
  
 @Provides  
 fun **provideSignUpUseCase**(signInSignUpManager: SignInSignUpManager): SignUpUseCase {  
 return SignUpUseCase(signInSignUpManager)  
 }  
  
 @Provides  
 fun **provideSignInUseCase**(signInSignUpManager: SignInSignUpManager): SignInUseCase {  
 return SignInUseCase(signInSignUpManager)  
 }  
  
}

package com.example.innowisepexelstestapp.di  
  
import android.content.Context  
import androidx.lifecycle.ViewModelProvider  
import com.example.innowisepexelstestapp.di.module.AppCiceroneModule  
import com.example.innowisepexelstestapp.di.module.AppModule  
import com.example.innowisepexelstestapp.di.module.AppViewModelModule  
import com.example.innowisepexelstestapp.di.module.DataRoomModule  
import com.example.innowisepexelstestapp.di.module.DataNetworkModule  
import com.example.innowisepexelstestapp.di.module.DomainRepositoryModule  
import com.example.innowisepexelstestapp.di.module.DomainUseCaseModule  
import com.example.innowisepexelstestapp.presentation.rv.RvCategoryAdapter  
import com.example.innowisepexelstestapp.presentation.view.MainActivity  
import dagger.BindsInstance  
import dagger.Component  
import javax.inject.Singleton  
  
@Singleton  
@Component(modules =  
 [AppCiceroneModule::class,  
 AppViewModelModule::class,  
 AppModule::class,  
 DomainUseCaseModule::class,  
 DomainRepositoryModule::class,  
 DataNetworkModule::class,  
 DataRoomModule::class]  
)  
interface AppComponent {  
  
 @Component.Builder  
 interface Builder {  
  
 @BindsInstance  
 fun **applicationContext**(context: Context): Builder  
 fun **build**(): AppComponent  
 }  
  
 fun **provideFactory**(): ViewModelProvider.Factory  
  
 fun **inject**(mainActivity: MainActivity)  
  
 fun **inject**(categoryHolder: RvCategoryAdapter.CategoryHolder)  
  
}

package com.example.innowisepexelstestapp.di  
  
import androidx.appcompat.app.AppCompatActivity  
import androidx.fragment.app.Fragment  
import androidx.lifecycle.LifecycleOwner  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.ViewModelProvider  
import com.example.innowisepexelstestapp.App  
import javax.inject.Inject  
import javax.inject.Provider  
import kotlin.reflect.KClass  
  
class ViewModelFactory @Inject constructor(  
 private val models: MutableMap<Class<out ViewModel>, Provider<ViewModel>>  
) : ViewModelProvider.Factory {  
 override fun <T : ViewModel> **create**(modelClass: Class<T>): T {  
 val viewModelProvider = models[modelClass]  
 ?: throw IllegalArgumentException("$modelClass not found")  
  
 @Suppress("UNCHECKED\_CAST")  
 return viewModelProvider.get() as T  
 }  
}  
  
inline fun <T, reified VM : ViewModel> T.**injectViewModel**(  
 viewModelClass: KClass<VM> = VM::class  
) where T : AppCompatActivity, T : LifecycleOwner = *lazy*(this) {  
 ViewModelProvider(  
 this,  
 (this.*application* as App).appComponent.provideFactory()  
 )[viewModelClass.*java*]  
}  
  
inline fun <T, reified VM : ViewModel> T.**injectViewModel**(  
 viewModelClass: KClass<VM> = VM::class  
) where T : Fragment, T : LifecycleOwner = *lazy*(this) {  
 ViewModelProvider(  
 this,  
 (*activity*!!.*application* as App).appComponent.provideFactory()  
 )[viewModelClass.*java*]  
}

package com.example.innowisepexelstestapp.di  
  
import androidx.lifecycle.ViewModel  
import dagger.MapKey  
import kotlin.reflect.KClass  
  
@Target(  
 AnnotationTarget.*FUNCTION*,  
 AnnotationTarget.*PROPERTY\_GETTER*,  
 AnnotationTarget.*PROPERTY\_SETTER*)  
@Retention(AnnotationRetention.*RUNTIME*)  
@MapKey  
annotation class ViewModelKey(val value: KClass<out ViewModel>)

package com.example.innowisepexelstestapp.presentation.navigation  
  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.presentation.view.DetailsFragment  
import com.example.innowisepexelstestapp.presentation.view.FavoriteFragment  
import com.example.innowisepexelstestapp.presentation.view.HomeFragment  
import com.example.innowisepexelstestapp.presentation.view.SignInFragment  
import com.example.innowisepexelstestapp.presentation.view.SignUpFragment  
  
import com.example.innowisepexelstestapp.util.withArguments  
import com.github.terrakok.cicerone.androidx.FragmentScreen  
  
object Screens {  
 fun **homeFragment**() = FragmentScreen { HomeFragment() }  
 fun **favoriteFragment**() = FragmentScreen { FavoriteFragment() }  
 fun **detailsFragment**(photoPexels: PhotoPexels, isItLikedPhoto: Boolean) = FragmentScreen {  
 DetailsFragment().*withArguments*(  
 "photoPexels" *to* photoPexels,  
 "isItLikedPhoto" *to* isItLikedPhoto  
 )  
 }  
 fun **signInFragment**() = FragmentScreen { SignInFragment() }  
 fun **signUpFragment**() = FragmentScreen { SignUpFragment() }  
}

package com.example.innowisepexelstestapp.presentation.rv  
  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Button  
import androidx.recyclerview.widget.RecyclerView  
import com.example.innowisepexelstestapp.App  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.RvCategoryItemBinding  
import com.example.innowisepexelstestapp.model.Category  
import com.example.innowisepexelstestapp.util.ResourceProvider  
import javax.inject.Inject  
  
class RvCategoryAdapter(private val mListener: ClickListener) :  
 RecyclerView.Adapter<RvCategoryAdapter.CategoryHolder>() {  
  
 val categories = *mutableListOf*<Category>()  
  
 class CategoryHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 @Inject  
 lateinit var resourceProvider: ResourceProvider  
 private val binding = RvCategoryItemBinding.bind(view)  
 init {  
 App.instance.appComponent.inject(this)  
 }  
  
  
 fun **bind**(category: Category, position: Int, listener: ClickListener) = *with*(binding) {  
 btn.*text* = category.name  
  
 if(category.isActive) {  
 btn.*backgroundTintList* = resourceProvider.getAttrColor(  
 com.google.android.material.R.attr.*colorTertiary*, btn.*context* )  
 btn.*typeface* = resourceProvider.getFont(R.font.*mulish\_700*)  
 } else {  
 btn.*backgroundTintList* = resourceProvider.getAttrColor(  
 com.google.android.material.R.attr.*colorSurfaceContainer*, btn.*context* )  
 btn.*typeface* = resourceProvider.getFont(R.font.*mulish\_400*)  
 }  
 itemView.setOnClickListener {  
 listener.onClickCategory(category, position)  
 }  
 }  
 }  
  
 override fun **onCreateViewHolder**(parent: ViewGroup, viewType: Int): CategoryHolder {  
 val view = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*rv\_category\_item*, parent, false)  
 return CategoryHolder(view)  
 }  
  
 override fun **onBindViewHolder**(holder: CategoryHolder, position: Int) {  
 holder.bind(categories[position], position, mListener)  
 }  
  
 override fun **getItemCount**(): Int {  
 return categories.size  
 }  
  
 fun **addCategories**(categoriesToAdd: List<Category>) {  
 categories.clear() //*todo тут добавить не через задницу* categories.addAll(categoriesToAdd)  
 notifyDataSetChanged()  
 }  
  
 interface ClickListener {  
 fun **onClickCategory**(category: Category, position: Int)  
 }  
}

package com.example.innowisepexelstestapp.presentation.rv  
  
import android.annotation.SuppressLint  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.RvPhotoItemBinding  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.squareup.picasso.Picasso  
import io.reactivex.Observable  
import io.reactivex.android.schedulers.AndroidSchedulers  
import io.reactivex.schedulers.Schedulers  
  
class RvPhotoAdapter(private val mListener: ClickListener, private val showAuthorName: Boolean) :  
 RecyclerView.Adapter<RvPhotoAdapter.PhotoHolder>() {  
  
 private val photoPexelsArray = *mutableListOf*<PhotoPexels>()  
  
 class PhotoHolder(view: View, isItForFavoriteScreen: Boolean) : RecyclerView.ViewHolder(view) {  
 private val binding = RvPhotoItemBinding.bind(view)  
  
 init {  
 if (isItForFavoriteScreen) {  
 binding.authorName.*visibility* = View.*VISIBLE* }  
 }  
  
 fun **bind**(photoPexels: PhotoPexels, listener: ClickListener) = *with*(binding) {  
 Picasso.get()  
 .load(photoPexels.sources.medium)  
 .placeholder(R.drawable.*ic\_imagestub*)  
 .into(rvHomeItem)  
  
 authorName.*text* = photoPexels.photographer  
  
 itemView.setOnClickListener {  
 listener.onClickPhoto(photoPexels)  
 }  
 }  
 }  
  
 override fun **onCreateViewHolder**(parent: ViewGroup, viewType: Int): PhotoHolder {  
 val view = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*rv\_photo\_item*, parent, false)  
 return PhotoHolder(view, showAuthorName)  
 }  
  
 override fun **onBindViewHolder**(holder: PhotoHolder, position: Int) {  
 holder.bind(photoPexelsArray[position], mListener)  
 }  
  
 override fun **getItemCount**(): Int {  
 return photoPexelsArray.size  
 }  
  
 @SuppressLint("CheckResult")  
 fun **addPhotoList**(photoPexelsList: List<PhotoPexels>) {  
 val newPhotoList = *mutableListOf*<PhotoPexels>()  
 Observable.just(photoPexelsList)  
 .subscribeOn(Schedulers.computation())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe {  
 newPhotoList.clear()  
 newPhotoList.addAll(**it**)  
 val listSize = newPhotoList.size  
 photoPexelsArray.addAll(**it**)  
 notifyItemRangeInserted(photoPexelsArray.size - listSize, listSize)  
 }  
 }  
  
 @SuppressLint("CheckResult")  
 fun **createNewPhotoList**(photoPexelsList: List<PhotoPexels>) {  
 Observable.just(photoPexelsList)  
 .subscribeOn(Schedulers.computation())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe {  
 photoPexelsArray.clear()  
 photoPexelsArray.addAll(photoPexelsList)  
 notifyDataSetChanged()  
 }  
 }  
  
  
 interface ClickListener {  
 fun **onClickPhoto**(photoPexels: PhotoPexels)  
 }  
}

package com.example.innowisepexelstestapp.presentation.view  
  
import android.os.Bundle  
import android.view.View  
import android.widget.Toast  
import androidx.fragment.app.Fragment  
import by.kirich1409.viewbindingdelegate.viewBinding  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.FragmentDetailsBinding  
import com.example.innowisepexelstestapp.di.injectViewModel  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.presentation.viewmodel.DetailsViewModel  
import com.example.innowisepexelstestapp.util.findArgument  
import com.squareup.picasso.Picasso  
  
class DetailsFragment : Fragment(R.layout.*fragment\_details*) {  
  
 private val mViewBinding by *viewBinding*(FragmentDetailsBinding::bind)  
 private val mViewModel: DetailsViewModel by *injectViewModel*() //*todo провериь и без injectViewModel* private val mPhotoPexelsArg: PhotoPexels by *lazy* (LazyThreadSafetyMode.*NONE*) {  
 *findArgument*("photoPexels")!! }  
 private val mIsItLikedPhotoArg: Boolean by *lazy* (LazyThreadSafetyMode.*NONE*) {  
 *findArgument*("isItLikedPhoto")!! }  
  
 override fun **onViewCreated**(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 setupListeners()  
 setViewsPresets()  
 setupObservers()  
 }  
  
 private fun **setViewsPresets**(): Unit = *with*(mViewBinding) {  
 mViewModel.setFavoriteBtnImage(mIsItLikedPhotoArg, favoriteBtn)  
  
 Picasso.get()  
 .load(mPhotoPexelsArg.sources.original)  
 .placeholder(R.drawable.*ic\_imagestub*)  
 .into(image)  
 }  
  
 private fun **setupListeners**(): Unit = *with*(mViewBinding) {  
 var isItLikedPhoto = mIsItLikedPhotoArg  
  
 favoriteBtn.setOnClickListener {  
 mViewModel.onFavoriteBtn(isItLikedPhoto, favoriteBtn, mPhotoPexelsArg)  
 isItLikedPhoto = !isItLikedPhoto  
 }  
  
 image.setOnViewDragListener { dx, dy ->  
 if (dx != 0f || dy != 0f) {  
 image.setScale(1.0f, true)  
 }  
 }  
 authorName.*text* = mPhotoPexelsArg.photographer  
  
 downloadBtn.setOnClickListener {  
 mViewModel.onDownloadBtn(mPhotoPexelsArg.sources.original)  
 }  
 backBtn.setOnClickListener {  
 mViewModel.onBackBtn()  
 }  
 }  
  
 private fun **setupObservers**() {  
 mViewModel.ldShowToast.observe(*viewLifecycleOwner*) {  
 Toast.makeText(requireContext(), **it**, Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
}

package com.example.innowisepexelstestapp.presentation.view  
  
import android.os.Bundle  
import android.view.View  
import androidx.fragment.app.Fragment  
import androidx.recyclerview.widget.StaggeredGridLayoutManager  
import by.kirich1409.viewbindingdelegate.viewBinding  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.FragmentFavoriteBinding  
import com.example.innowisepexelstestapp.di.injectViewModel  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.presentation.rv.RvPhotoAdapter  
import com.example.innowisepexelstestapp.presentation.viewmodel.FavoriteViewModel  
  
class FavoriteFragment : Fragment(R.layout.*fragment\_favorite*), RvPhotoAdapter.ClickListener {  
  
 private val mViewBinding by *viewBinding*(FragmentFavoriteBinding::bind)  
 private val mViewModel: FavoriteViewModel by *injectViewModel*()  
 private val mRvPhotoAdapter: RvPhotoAdapter = RvPhotoAdapter(this, showAuthorName = true)  
  
 override fun **onViewCreated**(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 setViewsPresets()  
 mViewModel.setPhotos()  
 setupListeners()  
 setupObservers()  
 }  
  
 override fun **onClickPhoto**(photoPexels: PhotoPexels) {  
 mViewModel.onClickPhoto(photoPexels)  
 }  
  
 private fun **setViewsPresets**(): Unit = *with*(mViewBinding) {  
 favoriteRv.*adapter* = mRvPhotoAdapter  
 }  
  
 private fun **setupListeners**(): Unit = *with*(mViewBinding) {  
 bnvHome.setOnClickListener {  
 mViewModel.navigateToHome()  
 }  
 tvExplore.setOnClickListener {  
 mViewModel.navigateToHome()  
 }  
 ivLogout.setOnClickListener {  
 mViewModel.onLogout()  
 }  
 }  
  
 private fun **setupObservers**(): Unit = *with*(mViewBinding) {  
 mViewModel.ldAddPhotoList.observe(*viewLifecycleOwner*) {  
 mRvPhotoAdapter.createNewPhotoList(**it**)  
 }  
 mViewModel.ldShowAnim.observe(*viewLifecycleOwner*) {  
 favoriteRv.startAnimation(**it**)  
 }  
 mViewModel.ldTvNoFavoritesVisibility.observe(*viewLifecycleOwner*) {  
 tvNoFavorites.*visibility* = **it** }  
 mViewModel.ldTvExploreVisibility.observe(*viewLifecycleOwner*) {  
 tvExplore.*visibility* = **it** }  
 }  
  
}

package com.example.innowisepexelstestapp.presentation.view  
  
import android.os.Bundle  
import android.view.View  
import androidx.core.widget.doAfterTextChanged  
import androidx.fragment.app.Fragment  
import androidx.recyclerview.widget.LinearLayoutManager  
import androidx.recyclerview.widget.RecyclerView  
import androidx.recyclerview.widget.StaggeredGridLayoutManager  
import by.kirich1409.viewbindingdelegate.viewBinding  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.FragmentHomeBinding  
import com.example.innowisepexelstestapp.di.injectViewModel  
import com.example.innowisepexelstestapp.model.Category  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.presentation.rv.RvCategoryAdapter  
import com.example.innowisepexelstestapp.presentation.rv.RvPhotoAdapter  
import com.example.innowisepexelstestapp.presentation.viewmodel.HomeViewModel  
  
class HomeFragment : Fragment(R.layout.*fragment\_home*),  
 RvPhotoAdapter.ClickListener,  
 RvCategoryAdapter.ClickListener {  
  
 private val mViewBinding by *viewBinding*(FragmentHomeBinding::bind)  
 private val mViewModel: HomeViewModel by *injectViewModel*() //*todo попробовать by viewmodels* private val mRvPhotoAdapter: RvPhotoAdapter = RvPhotoAdapter(this,  
 showAuthorName = false)  
 private val mRvCategoryAdapter: RvCategoryAdapter = RvCategoryAdapter(this)  
  
 override fun **onViewCreated**(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 setViewsPresets()  
 setupListeners()  
 setupObservers()  
 }  
  
 override fun **onClickPhoto**(photoPexels: PhotoPexels) {  
 mViewModel.onClickPhoto(photoPexels)  
 }  
  
 override fun **onClickCategory**(category: Category, position: Int) {  
 mViewModel.onClickCategory(category, position)  
 }  
  
 private fun **setViewsPresets**(): Unit = *with*(mViewBinding) {  
 categoryRv.*layoutManager* = LinearLayoutManager(*context*,  
 LinearLayoutManager.*HORIZONTAL*, false)  
  
 homeRv.*adapter* = mRvPhotoAdapter  
 categoryRv.*adapter* = mRvCategoryAdapter  
 categoryRv.setItemViewCacheSize(7)  
 }  
  
 private fun **setupListeners**(): Unit = *with*(mViewBinding) {  
 var searchBarText = ""  
 bnvFavorite.setOnClickListener {  
 mViewModel.navigateToFavorite()  
 }  
 tvTryAgain.setOnClickListener {  
 mViewModel.addPhotos()  
 mViewModel.setCategories()  
 }  
 searchBarCloseIcon.setOnClickListener {  
 mViewModel.onSearchBarCloseIcon()  
 }  
 searchBarEditText.*doAfterTextChanged* {  
 searchBarText = searchBarEditText.*text*.toString().*trim*()  
 mViewModel.doAfterTextChanged(**it**!!)  
 }  
 searchBarSearchIcon.setOnClickListener {  
 mViewModel.addQueryPhotos(searchBarText)  
 }  
 homeRv.addOnScrollListener(object : RecyclerView.OnScrollListener() {  
 override fun **onScrolled**(recyclerView: RecyclerView, dx: Int, dy: Int) {  
 super.onScrolled(recyclerView, dx, dy)  
 mViewModel.onScrolledRv(recyclerView, searchBarText)  
 }  
 })  
 }  
  
 private fun **setupObservers**(): Unit = *with*(mViewBinding) {  
 mViewModel.ldOnCloseButton.observe(*viewLifecycleOwner*) {  
 searchBarEditText.*text*.clear()  
 searchBarEditText.clearFocus()  
 mRvCategoryAdapter.categories.*forEach* { category ->  
 category.isActive = false  
 }  
 mRvCategoryAdapter.notifyDataSetChanged()  
 }  
 mViewModel.ldSearchBarCloseIconVisibility.observe(*viewLifecycleOwner*) {  
 searchBarCloseIcon.*visibility* = **it** }  
 mViewModel.ldIvNoNetworkVisibility.observe(*viewLifecycleOwner*) {  
 ivNonetwork.*visibility* = **it** }  
 mViewModel.ldTvTryAgainVisibility.observe(*viewLifecycleOwner*) {  
 tvTryAgain.*visibility* = **it** }  
 mViewModel.ldAddPhotoList.observe(*viewLifecycleOwner*) {  
 mRvPhotoAdapter.addPhotoList(**it**)  
 }  
 mViewModel.ldCreateNewPhotoList.observe(*viewLifecycleOwner*) {  
 mRvPhotoAdapter.createNewPhotoList(**it**)  
 }  
 mViewModel.ldAddCategoryList.observe(*viewLifecycleOwner*) {  
 mRvCategoryAdapter.addCategories(**it**)  
 }  
 mViewModel.ldProgressBarVisibility.observe(*viewLifecycleOwner*) {  
 progressBar.*visibility* = **it** }  
 mViewModel.ldShowAnim.observe(*viewLifecycleOwner*) {  
 homeRv.startAnimation(**it**)  
 }  
 mViewModel.ldSetActiveCategory.observe(*viewLifecycleOwner*) {  
 mRvCategoryAdapter.categories.*forEach* { category ->  
 category.isActive = false  
 }  
 mRvCategoryAdapter.categories[**it**].isActive = true  
 mRvCategoryAdapter.notifyDataSetChanged()  
 }  
 mViewModel.ldSetEditTextCategoryName.observe(*viewLifecycleOwner*) {  
 searchBarEditText.*text* = **it** }  
 }  
}

package com.example.innowisepexelstestapp.presentation.view  
  
import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
import by.kirich1409.viewbindingdelegate.viewBinding  
import com.example.innowisepexelstestapp.App  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.ActivityMainBinding  
import com.example.innowisepexelstestapp.di.injectViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.MainViewModel  
import com.github.terrakok.cicerone.Command  
import com.github.terrakok.cicerone.Navigator  
import com.github.terrakok.cicerone.NavigatorHolder  
import com.github.terrakok.cicerone.Replace  
import com.github.terrakok.cicerone.androidx.AppNavigator  
import javax.inject.Inject  
//*todo disposableBag сделать для всего rx*//*todo поменять serialize в bundle на parcel*//*todo сделать свайпы влево и вправо на фото в списке*//*todo сделать экран множественного открытия с compose*//*todo extended fab*//*todo перевести picasso на glide*//*todo отъебывается программа при notifyItemRangeInserted(photoPexelsArray.size - listSize, listSize). исправить*//*todo закинуть нормально в viewmodel то, что при нажатии категории*//*todo пофиксить баг непрогрузки интерфейса при повороте экрана на запуске*//*todo мб поменять splashscreen на возможности сторонних библиотек*//*todo мб добавить эффект нажатия на изображение*//*todo мб добавить размер фото на кнопку скачивания*class MainActivity : AppCompatActivity() {  
  
 private val mViewBinding by *viewBinding*(ActivityMainBinding::bind)  
 private val mViewModel: MainViewModel by *injectViewModel*()  
  
 @Inject  
 lateinit var mNavigatorHolder: NavigatorHolder  
  
 override fun **onCreate**(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 App.instance.appComponent.inject(this)  
 setContentView(R.layout.*activity\_main*)  
  
 getSystemService(*DOWNLOAD\_SERVICE*)  
  
 setupObservers(savedInstanceState)  
 }  
  
 private val mNavigator: Navigator = object : AppNavigator(this, R.id.*container*) {  
 override fun **applyCommands**(commands: Array<out Command>) {  
 super.applyCommands(commands)  
 *supportFragmentManager*.executePendingTransactions()  
 }  
 }  
  
 override fun **onResumeFragments**() {  
 super.onResumeFragments()  
 mNavigatorHolder.setNavigator(mNavigator)  
 }  
  
 override fun **onPause**() {  
 mNavigatorHolder.removeNavigator()  
 super.onPause()  
 }  
  
 private fun **setupObservers**(savedInstanceState: Bundle?) = *with*(mViewBinding) {  
 mViewModel.ldLogoVisibility.observe(this@MainActivity) {  
 logo.*visibility* = **it** }  
 mViewModel.ldLogoBackgroundVisibility.observe(this@MainActivity) {  
 logoBackground.*visibility* = **it** }  
 mViewModel.ldLogoStartAnim.observe(this@MainActivity) {  
 logo.startAnimation(**it**)  
 }  
 if (savedInstanceState == null) {  
 mViewModel.ldSetStartFragment.observe(this@MainActivity) {  
 mNavigator.applyCommands(*arrayOf*<Command>(Replace(**it**)))  
 }  
 }  
 }  
  
}

package com.example.innowisepexelstestapp.presentation.view  
  
import android.os.Bundle  
import android.view.View  
import android.widget.Toast  
import androidx.fragment.app.Fragment  
import by.kirich1409.viewbindingdelegate.viewBinding  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.FragmentSignInBinding  
import com.example.innowisepexelstestapp.di.injectViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.SignInViewModel  
  
class SignInFragment : Fragment(R.layout.*fragment\_sign\_in*) {  
  
 private val mViewBinding by *viewBinding*(FragmentSignInBinding::bind)  
 private val mViewModel: SignInViewModel by *injectViewModel*()  
  
 override fun **onViewCreated**(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 setupListeners()  
 setViewsPresets()  
 setupObservers()  
 }  
  
 private fun **setViewsPresets**(): Unit = with(mViewBinding) {  
  
 }  
  
 private fun **setupListeners**(): Unit = *with*(mViewBinding) {  
 tvSignUp.setOnClickListener {  
 mViewModel.onSignUpBtn()  
 }  
 btnSignIn.setOnClickListener {  
 mViewModel.onSignInBtn(etEmail.*text*.toString(), etPassword.*text*.toString())  
 }  
 }  
  
 private fun **setupObservers**(): Unit = *with*(mViewModel) {  
 ldShowToastText.observe(*viewLifecycleOwner*) {  
 Toast.makeText(requireContext(), **it**, Toast.*LENGTH\_LONG*).show()  
 }  
 }  
}

package com.example.innowisepexelstestapp.presentation.view  
  
import android.os.Bundle  
import android.view.View  
import android.widget.Toast  
import androidx.fragment.app.Fragment  
import by.kirich1409.viewbindingdelegate.viewBinding  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.databinding.FragmentSignUpBinding  
import com.example.innowisepexelstestapp.di.injectViewModel  
import com.example.innowisepexelstestapp.presentation.viewmodel.SignUpViewModel  
  
class SignUpFragment : Fragment(R.layout.*fragment\_sign\_up*) {  
  
 private val mViewBinding by *viewBinding*(FragmentSignUpBinding::bind)  
 private val mViewModel: SignUpViewModel by *injectViewModel*()  
  
 override fun **onViewCreated**(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 setupListeners()  
 setViewsPresets()  
 setupObservers()  
 }  
  
 private fun **setViewsPresets**(): Unit = with(mViewBinding) {  
  
 }  
  
 private fun **setupListeners**(): Unit = *with*(mViewBinding) {  
 tvSignIn.setOnClickListener {  
 mViewModel.onSignInBtn()  
 }  
 btnSignUn.setOnClickListener {  
 mViewModel.onSignUpBtn(  
 etEmail.*text*.toString(),  
 etPassword.*text*.toString(),  
 etConfirmPassword.*text*.toString()  
 )  
 }  
 }  
  
 private fun **setupObservers**() : Unit = *with*(mViewModel) {  
 ldShowToastText.observe(*viewLifecycleOwner*) {  
 Toast.makeText(requireContext(), **it**, Toast.*LENGTH\_LONG*).show()  
 }  
 }  
}

package com.example.innowisepexelstestapp.presentation.viewmodel  
  
import android.widget.ImageView  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.usecase.DeleteImageFromBdUseCase  
import com.example.innowisepexelstestapp.usecase.DownloadImageUseCase  
import com.example.innowisepexelstestapp.usecase.SaveImageIntoBdUseCase  
import com.example.innowisepexelstestapp.util.ResourceProvider  
import com.github.terrakok.cicerone.Router  
import javax.inject.Inject  
  
class DetailsViewModel @Inject constructor(  
 private val mRouter: Router,  
 private val mResourceProvider: ResourceProvider,  
 private val saveImageIntoBdUseCase: SaveImageIntoBdUseCase,  
 private val deleteImageIntoRoomUseCase: DeleteImageFromBdUseCase,  
 private val downloadImageUseCase: DownloadImageUseCase  
) : ViewModel() {  
  
 private val \_ldShowToast: MutableLiveData<String> = MutableLiveData()  
  
 val ldShowToast: LiveData<String> = \_ldShowToast  
  
  
 fun **onDownloadBtn**(imageUrl: String) {  
 downloadImageUseCase.execute(imageUrl)  
 \_ldShowToast.*value* = mResourceProvider.getStringRes(R.string.*downloading*)  
 }  
  
 fun **onBackBtn**() {  
 mRouter.exit()  
 }  
  
 fun **onFavoriteBtn**(isItLikedPhoto: Boolean, view: ImageView, photoPexels: PhotoPexels) {  
 // I tried to set the image through the LiveData and using the ResourceProvider,  
 // but because the image is SVG, it is not set normally. So i set directly  
  
 if (isItLikedPhoto) {  
 view.setImageResource(R.drawable.*ic\_favorite\_inactive*)  
 deleteImageIntoRoomUseCase.execute(photoPexels)  
 \_ldShowToast.*value* = mResourceProvider.getStringRes(R.string.*deleted*)  
 } else {  
 view.setImageResource(R.drawable.*ic\_favorite\_active*)  
 saveImageIntoBdUseCase.execute(photoPexels)  
 \_ldShowToast.*value* = mResourceProvider.getStringRes(R.string.*saved*)  
 }  
  
 }  
  
 fun **setFavoriteBtnImage**(mIsItLikedPhotoArg: Boolean, favoriteBtn: ImageView) {  
 // I tried to set the image through the LiveData and using the ResourceProvider,  
 // but because the image is SVG, it is not set normally. So i set directly  
 if(mIsItLikedPhotoArg) {  
 favoriteBtn.setImageResource(R.drawable.*ic\_favorite\_active*)  
 }  
 }  
}

package com.example.innowisepexelstestapp.presentation.viewmodel  
  
import android.annotation.SuppressLint  
import android.view.View  
import android.view.animation.AlphaAnimation  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.presentation.navigation.Screens  
import com.example.innowisepexelstestapp.repository.SignInSignUpManager  
import com.example.innowisepexelstestapp.usecase.GetImagesFromBdUseCase  
import com.github.terrakok.cicerone.Router  
import io.reactivex.Single  
import io.reactivex.android.schedulers.AndroidSchedulers  
import io.reactivex.schedulers.Schedulers  
import javax.inject.Inject  
  
class FavoriteViewModel @Inject constructor(  
 private val mRouter: Router,  
 private val mGetImagesFromBdUseCase: GetImagesFromBdUseCase,  
 private val mSignInSignUpManager: SignInSignUpManager  
) : ViewModel() {  
  
 private val \_ldTvNoFavoritesVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldTvExploreVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldAddPhotoList: MutableLiveData<List<PhotoPexels>> = MutableLiveData()  
 private val \_ldShowAnim: MutableLiveData<AlphaAnimation> = MutableLiveData()  
  
 val ldTvNoFavoritesVisibility: LiveData<Int> = \_ldTvNoFavoritesVisibility  
 val ldTvExploreVisibility: LiveData<Int> = \_ldTvExploreVisibility  
 val ldAddPhotoList: LiveData<List<PhotoPexels>> = \_ldAddPhotoList  
 val ldShowAnim: LiveData<AlphaAnimation> = \_ldShowAnim  
  
 fun **onClickPhoto**(photoPexels: PhotoPexels) {  
 mRouter.navigateTo(Screens.detailsFragment(photoPexels, isItLikedPhoto = true))  
 }  
  
 fun **navigateToHome**() {  
 mRouter.exit()  
 }  
  
 @SuppressLint("CheckResult")  
 fun **setPhotos**() {  
 Single.fromCallable {  
 mGetImagesFromBdUseCase.execute()  
 }  
 .subscribeOn(Schedulers.computation())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe({ photos ->  
 \_ldAddPhotoList.*value* = photos  
 showRvAlphaAnimation()  
  
 if (photos.*isNotEmpty*()) {  
 \_ldTvNoFavoritesVisibility.*value* = View.*GONE* \_ldTvExploreVisibility.*value* = View.*GONE* } else {  
 \_ldTvNoFavoritesVisibility.*value* = View.*VISIBLE* \_ldTvExploreVisibility.*value* = View.*VISIBLE* }  
 }, { throwable ->  
 throwable.printStackTrace()  
 })  
 }  
  
 private fun **showRvAlphaAnimation**() {  
 val fadeInAnimation = AlphaAnimation(0f, 1f)  
 fadeInAnimation.*duration* = 300  
 \_ldShowAnim.*value* = fadeInAnimation  
 }  
  
 @SuppressLint("CheckResult")  
 fun **onLogout**() {  
 mSignInSignUpManager.logOutUser()  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe { isLogoutSuccessful ->  
 if(isLogoutSuccessful) {  
 mRouter.newRootScreen(Screens.signInFragment())  
 }  
 }  
 }  
}

package com.example.innowisepexelstestapp.presentation.viewmodel  
  
import android.annotation.SuppressLint  
import android.text.Editable  
import android.util.Log  
import android.view.View  
import android.view.animation.AlphaAnimation  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import androidx.recyclerview.widget.RecyclerView  
import androidx.recyclerview.widget.StaggeredGridLayoutManager  
import com.example.innowisepexelstestapp.model.Category  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.presentation.navigation.Screens  
import com.example.innowisepexelstestapp.repository.NetworkManager  
import com.github.terrakok.cicerone.Router  
import io.reactivex.Observable  
import io.reactivex.Single  
import io.reactivex.android.schedulers.AndroidSchedulers  
import io.reactivex.disposables.Disposable  
import io.reactivex.schedulers.Schedulers  
import java.util.concurrent.TimeUnit  
import javax.inject.Inject  
  
class HomeViewModel @Inject constructor(  
 private val mRouter: Router,  
 private val mNetworkManager: NetworkManager,  
) : ViewModel() {  
  
 private val \_ldOnCloseButton: MutableLiveData<Unit> = MutableLiveData()  
 private val \_ldSearchBarCloseIconVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldAddPhotoList: MutableLiveData<List<PhotoPexels>> = MutableLiveData()  
 private val \_ldCreateNewPhotoList: MutableLiveData<List<PhotoPexels>> = MutableLiveData()  
 private val \_ldAddCategoryList: MutableLiveData<List<Category>> = MutableLiveData()  
 private val \_ldIvNoNetworkVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldTvTryAgainVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldProgressBarVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldShowAnim: MutableLiveData<AlphaAnimation> = MutableLiveData()  
 private val \_ldSetActiveCategory: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldSetEditTextCategoryName: MutableLiveData<Editable> = MutableLiveData()  
  
 val ldOnCloseButton: LiveData<Unit> = \_ldOnCloseButton  
 val ldSearchBarCloseIconVisibility: LiveData<Int> = \_ldSearchBarCloseIconVisibility  
 val ldAddPhotoList: LiveData<List<PhotoPexels>> = \_ldAddPhotoList  
 val ldCreateNewPhotoList: LiveData<List<PhotoPexels>> = \_ldCreateNewPhotoList  
 val ldAddCategoryList: LiveData<List<Category>> = \_ldAddCategoryList  
 val ldIvNoNetworkVisibility: LiveData<Int> = \_ldIvNoNetworkVisibility  
 val ldTvTryAgainVisibility: LiveData<Int> = \_ldTvTryAgainVisibility  
 val ldProgressBarVisibility: LiveData<Int> = \_ldProgressBarVisibility  
 val ldShowAnim: LiveData<AlphaAnimation> = \_ldShowAnim  
 val ldSetActiveCategory: LiveData<Int> = \_ldSetActiveCategory  
 val ldSetEditTextCategoryName: LiveData<Editable> = \_ldSetEditTextCategoryName  
  
 private val queryNamesList = *mutableListOf*("")  
  
 private var disposable: Disposable? = null  
  
 init {  
 addPhotos()  
 setCategories()  
 }  
  
 @SuppressLint("CheckResult")  
 fun **doAfterTextChanged**(editable: Editable) {  
 val text = editable.toString().*trim*()  
 if (text.*isNotEmpty*()) {  
 \_ldSearchBarCloseIconVisibility.*value* = View.*VISIBLE* disposable?.dispose()  
 disposable = Observable.timer(1000, TimeUnit.*MILLISECONDS*)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe {  
 addQueryPhotos(text)  
 }  
  
 } else {  
 \_ldSearchBarCloseIconVisibility.*value* = View.*GONE* disposable?.dispose()  
 }  
 }  
  
 fun **addQueryPhotos**(query: String) {  
 queryNamesList.add(query)  
 observePhotos(mNetworkManager.getQueryPhotos(query))  
 }  
  
 fun **addPhotos**() {  
 queryNamesList.add("")  
 observePhotos(mNetworkManager.getCuratedPhotos())  
 }  
  
 @SuppressLint("CheckResult")  
 private fun **observePhotos**(singleList: Single<List<PhotoPexels>>) {  
 singleList.observeOn(AndroidSchedulers.mainThread())  
 .subscribe({ photos ->  
 \_ldIvNoNetworkVisibility.*value* = View.*GONE* \_ldTvTryAgainVisibility.*value* = View.*GONE* chooseSetPhotoType(photos)  
  
 \_ldProgressBarVisibility.*value* = View.*INVISIBLE* }, {  
 \_ldIvNoNetworkVisibility.*value* = View.*VISIBLE* \_ldTvTryAgainVisibility.*value* = View.*VISIBLE* })  
 showRvAlphaAnimation()  
 }  
  
 fun **onClickPhoto**(photoPexels: PhotoPexels) {  
 mRouter.navigateTo(Screens.detailsFragment(photoPexels, isItLikedPhoto = false))  
 }  
  
 fun **onClickCategory**(category: Category, position: Int) {  
 \_ldSetActiveCategory.*value* = position  
 \_ldSetEditTextCategoryName.*value* = Editable.Factory.getInstance()  
 .newEditable(category.name)  
 }  
  
 fun **onSearchBarCloseIcon**() {  
 \_ldOnCloseButton.*value* = Unit  
 addPhotos()  
 }  
  
 fun **navigateToFavorite**() {  
 mRouter.navigateTo(Screens.favoriteFragment())  
 }  
  
 @SuppressLint("CheckResult")  
 fun **setCategories**() {  
 mNetworkManager.getCategories()  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe({ categories ->  
 \_ldAddCategoryList.*value* = categories  
 }, {  
  
 })  
 }  
  
 private fun **showRvAlphaAnimation**() {  
 val fadeInAnimation = AlphaAnimation(0f, 1f)  
 fadeInAnimation.*duration* = 300  
  
 \_ldShowAnim.*value* = fadeInAnimation  
 }  
  
 private var isNewUploadAllowed = true  
 @SuppressLint("CheckResult")  
 fun **onScrolledRv**(recyclerView: RecyclerView, text: String) {  
 //delay on 0.2 sec to avoid too frequent handling  
 if (isNewUploadAllowed) {  
  
 isNewUploadAllowed = false  
  
 val layoutManager = recyclerView.*layoutManager* as StaggeredGridLayoutManager  
 val lastVisibleItemPositions = layoutManager  
 .findLastVisibleItemPositions(null)  
 val totalItemCount = layoutManager.*itemCount* val maxVisibleItemPosition = lastVisibleItemPositions.*maxOrNull*()  
  
 if (maxVisibleItemPosition == totalItemCount - 1) {  
 Log.w("customLog", ">SUCCESSFULLY NEW RESPONSE")  
 if (text.*isEmpty*()) {  
 addPhotos()  
 } else {  
 addQueryPhotos(text)  
 }  
 }  
  
 Observable.timer(200, TimeUnit.*MILLISECONDS*)  
 .subscribeOn(Schedulers.io())  
 .observeOn(Schedulers.io())  
 .subscribe {  
 isNewUploadAllowed = true  
 }  
  
 }  
 }  
  
 */\*\*  
 \* This method checks whether the last request matches the second to last one.  
 \* if it matches call livedata with adding photos.  
 \* if it does not match, then the old photos are deleting and new ones are adding  
 \*/* private fun **chooseSetPhotoType**(photos: List<PhotoPexels>) {  
 if (queryNamesList.*last*() == queryNamesList[queryNamesList.size - 2]) {  
 \_ldAddPhotoList.*value* = photos  
 } else {  
 \_ldCreateNewPhotoList.*value* = photos  
 }  
  
 }  
  
}

package com.example.innowisepexelstestapp.presentation.viewmodel  
  
import android.annotation.SuppressLint  
import android.view.View  
import android.view.animation.Animation  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.presentation.navigation.Screens  
import com.example.innowisepexelstestapp.repository.SignInSignUpManager  
import com.example.innowisepexelstestapp.util.ResourceProvider  
import com.github.terrakok.cicerone.androidx.FragmentScreen  
import io.reactivex.Observable  
import io.reactivex.android.schedulers.AndroidSchedulers  
import io.reactivex.schedulers.Schedulers  
import java.util.concurrent.TimeUnit  
import javax.inject.Inject  
  
class MainViewModel @Inject constructor(  
 private val mResourceProvider: ResourceProvider,  
 private val signInSignUpManager: SignInSignUpManager) : ViewModel() {  
  
 private val \_ldLogoVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldLogoStartAnim: MutableLiveData<Animation> = MutableLiveData()  
 private val \_ldLogoBackgroundVisibility: MutableLiveData<Int> = MutableLiveData()  
 private val \_ldSetStartFragment: MutableLiveData<FragmentScreen> = MutableLiveData()  
  
 val ldLogoVisibility: LiveData<Int> = \_ldLogoVisibility  
 val ldLogoStartAnim: LiveData<Animation> = \_ldLogoStartAnim  
 val ldLogoBackgroundVisibility: LiveData<Int> = \_ldLogoBackgroundVisibility  
 val ldSetStartFragment: LiveData<FragmentScreen> = \_ldSetStartFragment  
  
 init {  
 startSplashScreenAnim()  
 delayedHideSplashScreen()  
 redirectIfLoggedIn()  
 }  
  
 @SuppressLint("CheckResult")  
 private fun **redirectIfLoggedIn**() {  
 signInSignUpManager.isUserSignedUp()  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe { isUserSignedUp ->  
 \_ldSetStartFragment.*value* = if (isUserSignedUp) {  
 Screens.homeFragment()  
 } else {  
 Screens.signInFragment()  
 }  
 }  
 }  
  
 @SuppressLint("CheckResult")  
 private fun **delayedHideSplashScreen**() {  
 Observable.timer(1200, TimeUnit.*MILLISECONDS*)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe {  
 \_ldLogoVisibility.*value* = View.*GONE* \_ldLogoBackgroundVisibility.*value* = View.*GONE* }  
 }  
  
 private fun **startSplashScreenAnim**() {  
 val animation = mResourceProvider.getAnim(R.anim.*fall\_animation*)  
 \_ldLogoStartAnim.*value* = animation  
 }  
  
}

package com.example.innowisepexelstestapp.presentation.viewmodel  
  
import android.annotation.SuppressLint  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.presentation.navigation.Screens  
import com.example.innowisepexelstestapp.usecase.SignInState  
import com.example.innowisepexelstestapp.usecase.SignInUseCase  
import com.example.innowisepexelstestapp.util.ResourceProvider  
import com.github.terrakok.cicerone.Router  
import io.reactivex.android.schedulers.AndroidSchedulers  
import io.reactivex.schedulers.Schedulers  
import javax.inject.Inject  
  
class SignInViewModel @Inject constructor(  
 private val mRouter: Router,  
 private val signInUseCase: SignInUseCase,  
 private val resourceProvider: ResourceProvider  
) : ViewModel() {  
  
 private val \_ldShowToastText: MutableLiveData<String> = MutableLiveData()  
 val ldShowToastText: LiveData<String> = \_ldShowToastText  
  
 fun **onSignUpBtn**() {  
 mRouter.navigateTo(Screens.signUpFragment())  
 }  
  
 @SuppressLint("CheckResult")  
 fun **onSignInBtn**(email: String, password: String) {  
 signInUseCase.execute(email, password)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe { resultState ->  
 when (resultState!!) {  
 SignInState.*SIGN\_IN\_FAILED* -> {  
 \_ldShowToastText.*value* = resourceProvider  
 .getStringRes(R.string.*failed\_to\_sign\_in\_to\_the\_account*)  
 }  
 SignInState.*EMPTY\_EMAIL* -> {  
 \_ldShowToastText.*value* = resourceProvider  
 .getStringRes(R.string.*email\_is\_empty*)  
 }  
 SignInState.*SIGN\_IN\_SUCCESSFUL* -> {  
 mRouter.newRootScreen(Screens.homeFragment())  
 }  
 }  
 }  
 }  
  
}

package com.example.innowisepexelstestapp.presentation.viewmodel  
  
import android.annotation.SuppressLint  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.example.innowisepexelstestapp.R  
import com.example.innowisepexelstestapp.presentation.navigation.Screens  
import com.example.innowisepexelstestapp.usecase.SignUpState  
import com.example.innowisepexelstestapp.usecase.SignUpUseCase  
import com.example.innowisepexelstestapp.util.ResourceProvider  
import com.github.terrakok.cicerone.Router  
import io.reactivex.android.schedulers.AndroidSchedulers  
import io.reactivex.schedulers.Schedulers  
import javax.inject.Inject  
  
class SignUpViewModel @Inject constructor(  
 private val mRouter: Router,  
 private val signUpUseCase: SignUpUseCase,  
 private val resourceProvider: ResourceProvider  
) : ViewModel() {  
  
 private val \_ldShowToastText: MutableLiveData<String> = MutableLiveData()  
 val ldShowToastText: LiveData<String> = \_ldShowToastText  
  
 fun **onSignInBtn**() {  
 mRouter.exit()  
 }  
  
 @SuppressLint("CheckResult")  
 fun **onSignUpBtn**(email: String, password: String, confirmPassword: String) {  
 signUpUseCase.execute(email, password, confirmPassword)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe { resultState ->  
 when (resultState!!) {  
 SignUpState.*INVALID\_PASSWORD* -> {  
 \_ldShowToastText.*value* = resourceProvider  
 .getStringRes(R.string.*the\_password\_must\_contain\_at\_least*)  
 }  
 SignUpState.*PASSWORDS\_ARE\_NOT\_EQUAL* -> {  
 \_ldShowToastText.*value* = resourceProvider  
 .getStringRes(R.string.*passwords\_are\_not\_equal*)  
 }  
 SignUpState.*EMPTY\_EMAIL* -> {  
 \_ldShowToastText.*value* = resourceProvider  
 .getStringRes(R.string.*email\_is\_empty*)  
 }  
 SignUpState.*SIGN\_UP\_FAILED* -> {  
 \_ldShowToastText.*value* = resourceProvider  
 .getStringRes(R.string.*failed\_to\_create\_an\_account*)  
 }  
 SignUpState.*SIGN\_UP\_SUCCESSFUL* -> {  
 mRouter.newRootScreen(Screens.homeFragment())  
 }  
 }  
 }  
 }  
}

package com.example.innowisepexelstestapp.util  
  
import android.os.Bundle  
import android.os.Parcelable  
import androidx.core.os.bundleOf  
import androidx.fragment.app.Fragment  
import java.io.Serializable  
import kotlin.reflect.KClass  
  
inline fun <reified T : Any> Bundle?.**getArgument**(key: String): T = *findArgument*(key)  
 ?: throw NullPointerException("Argument with key = $key not found")  
//*todo мб переенести этот класс в другой модуль*@Suppress("IMPLICIT\_CAST\_TO\_ANY")  
inline fun <reified T : Any> Bundle?.**findArgument**(key: String): T? = *findArgument*(key) {  
 val type = T::class  
  
 val argument = when {  
 type == String::class -> getString(key)  
 type == Int::class -> getInt(key, -1)  
 type == Long::class -> getLong(key, -1L)  
 type == Boolean::class -> getBoolean(key, false)  
 type == Set::class -> getSerializable(key)  
 type == List::class -> getSerializable(key)  
 type == CharSequence::class -> getCharSequence(key)  
 type *isAssign* Parcelable::class -> getParcelable(key)  
 type *isAssign* Serializable::class -> getSerializable(key)  
 else -> throw IllegalArgumentException("Unsupported argument type = ${type.simpleName}")  
 } as? T  
  
 when (argument) {  
 -1 -> null  
 else -> argument  
 }  
}  
  
infix fun KClass<\*>.**isAssign**(type: KClass<\*>): Boolean = type.*javaObjectType* .isAssignableFrom(this.*javaObjectType*)  
  
inline fun <reified T : Any> Bundle?.**findArgument**(key: String, getArgument: Bundle.() -> T?): T? =  
 this?.*let* { if (**it**.containsKey(key)) **it**.getArgument() else null }  
  
fun <T : Fragment> T.**withArguments**(vararg args: Pair<String, Any?>): T = *apply* {  
 *arguments*?.putAll(*bundleOf*(\*args)) ?: *run* { *arguments* = *bundleOf*(\*args) }  
}  
  
inline fun <reified T : Any> Fragment.**getArgument**(key: String): T = *arguments*.*getArgument*(key)  
  
inline fun <reified T : Any> Fragment.**findArgument**(key: String): T? = *arguments*.*findArgument*(key)  
  
inline fun <reified T : Any> Fragment.**findArgumentSafe**(key: String, handler: (T) -> Unit) {  
 *arguments*.*findArgument*<T>(key)?.*let* { handler.invoke(**it**) }  
}

package com.example.innowisepexelstestapp.util  
  
import android.content.Context  
import android.content.res.ColorStateList  
import android.graphics.Typeface  
import android.util.TypedValue  
import android.view.animation.Animation  
import android.view.animation.AnimationUtils  
import androidx.annotation.AttrRes  
import androidx.core.content.res.ResourcesCompat  
  
class ResourceProvider(private val mAppContext: Context) {  
  
 fun **getStringRes**(id: Int): String {  
 return mAppContext.getString(id)  
 }  
  
 fun **getAnim**(id: Int): Animation {  
 return AnimationUtils.loadAnimation(mAppContext, id)  
 }  
  
 fun **getColor**(id: Int): Int {  
 return mAppContext.getColor(id)  
 }  
  
 fun **getFont**(id: Int): Typeface {  
 return ResourcesCompat.getFont(mAppContext, id)!!  
 }  
  
 fun **getAttrColor**(@AttrRes attributeId: Int, context: Context) : ColorStateList {  
 //i know that passing context as a parameter is bad practice, but with mAppContext it doesn't work  
 val typedValue = TypedValue()  
 context.*theme*.resolveAttribute(attributeId, typedValue, true)  
 return ColorStateList.valueOf(typedValue.data)  
 }  
}

package com.example.innowisepexelstestapp  
  
import android.app.Application  
import com.example.innowisepexelstestapp.di.AppComponent  
import com.example.innowisepexelstestapp.di.DaggerAppComponent  
  
class App : Application() {  
  
 val appComponent: AppComponent by *lazy*(  
 DaggerAppComponent  
 .builder()  
 .applicationContext(this)::build  
 )  
  
 override fun **onCreate**() {  
 super.onCreate()  
 instance = this  
 }  
  
 companion object {  
 internal lateinit var instance: App  
 private set  
 }  
  
}

package com.example.innowisepexelstestapp.dto  
  
import com.google.gson.annotations.SerializedName  
  
data class CategoryDto(  
 @SerializedName("title") val name: String  
)

package com.example.innowisepexelstestapp.dto  
  
import com.google.gson.annotations.SerializedName  
  
data class CollectionsResultDto(  
 @SerializedName("collections") val collections: List<CategoryDto>,  
)

package com.example.innowisepexelstestapp.dto  
  
import com.google.gson.annotations.SerializedName  
  
data class CuratedResultDto(  
 @SerializedName("page") val page: Int,  
 @SerializedName("per\_page") val itemsPerPage: String,  
 @SerializedName("photos") val photos: List<PhotoPexelsDto>,  
 @SerializedName("prev\_page") val previousPageURL: String?,  
 @SerializedName("next\_page") val nextPageURL: String?  
)

package com.example.innowisepexelstestapp.dto  
  
import androidx.room.Entity  
import androidx.room.PrimaryKey  
import com.google.gson.annotations.SerializedName  
  
@Entity  
data class PhotoPexelsDto(  
 @PrimaryKey  
 @SerializedName("id") val id: Int = 0,  
 @SerializedName("photographer") val photographer: String = "",  
 @SerializedName("src") val sources: ImageSourcesDto = ImageSourcesDto(),  
)  
data class ImageSourcesDto(  
 @SerializedName("original") val original: String = "",  
 @SerializedName("medium") val medium: String = "",  
 @SerializedName("small") val small: String = "",  
)

package com.example.innowisepexelstestapp.mapper  
  
import com.example.innowisepexelstestapp.dto.CategoryDto  
import com.example.innowisepexelstestapp.model.Category  
  
class CategoryMapper {  
  
 fun **toDto**(c: Category): CategoryDto {  
 return CategoryDto(c.name)  
 }  
  
 fun **toModel**(cDto: CategoryDto): Category {  
 return Category(cDto.name, false)  
 }  
  
 fun **toDtos**(cList: List<Category>): List<CategoryDto> {  
 val dtoList = *mutableListOf*<CategoryDto>()  
 for (c in cList) {  
 dtoList.add(toDto(c))  
 }  
 return dtoList  
 }  
  
 fun **toModels**(dtoList: List<CategoryDto>): List<Category> {  
 val cList = *mutableListOf*<Category>()  
 for (dto in dtoList) {  
 cList.add(toModel(dto))  
 }  
 return cList  
 }  
  
}

package com.example.innowisepexelstestapp.mapper  
  
import com.example.innowisepexelstestapp.dto.ImageSourcesDto  
import com.example.innowisepexelstestapp.dto.PhotoPexelsDto  
import com.example.innowisepexelstestapp.model.ImageSources  
import com.example.innowisepexelstestapp.model.PhotoPexels  
  
class PhotoPexelsMapper {  
  
 fun **toDto**(p: PhotoPexels) : PhotoPexelsDto {  
 val s = p.sources  
 val imageSourcesDto = ImageSourcesDto(  
 s.original,  
 s.medium,  
 s.small,  
 )  
 return PhotoPexelsDto(  
 p.id,  
 p.photographer,  
 imageSourcesDto,  
 )  
 }  
  
 fun **toModel**(p: PhotoPexelsDto) : PhotoPexels {  
 val s = p.sources  
 val imageSources = ImageSources(  
 s.original,  
 s.medium,  
 s.small,  
  
 )  
 return PhotoPexels(  
 p.id,  
 p.photographer,  
 imageSources,  
 )  
 }  
  
 fun **toDtos**(pList: List<PhotoPexels>): List<PhotoPexelsDto> {  
 val dtoList = *mutableListOf*<PhotoPexelsDto>()  
 for (p in pList) {  
 dtoList.add(toDto(p))  
 }  
 return dtoList  
 }  
  
 fun **toModels**(dtoList: List<PhotoPexelsDto>): List<PhotoPexels> {  
 val pList = *mutableListOf*<PhotoPexels>()  
 for (p in dtoList) {  
 pList.add(toModel(p))  
 }  
 return pList  
 }  
}

package com.example.innowisepexelstestapp.repository.downloadmanager  
  
import android.app.DownloadManager  
import android.content.Context  
import android.net.Uri  
import android.os.Environment  
import android.webkit.CookieManager  
import android.webkit.URLUtil  
import androidx.core.content.ContextCompat.getSystemService  
import com.example.innowisepexelstestapp.repository.DownloadFilesManager  
  
class DownloadFilesManagerImpl(private val mAppContext: Context) : DownloadFilesManager {  
  
 override fun **download**(url: String) {  
 val request = DownloadManager.Request(Uri.parse(url))  
 val title = URLUtil.guessFileName(url, null, null)  
 request.setTitle(title)  
 request.setDescription("Downloading file, please wait...")  
 val cookie = CookieManager.getInstance().getCookie(url)  
 request.addRequestHeader("cookie", cookie)  
 request.setNotificationVisibility(DownloadManager  
 .Request  
 .*VISIBILITY\_VISIBLE\_NOTIFY\_COMPLETED*)  
 request.setDestinationInExternalPublicDir(Environment.*DIRECTORY\_DOWNLOADS*, title)  
  
 val downloadManager = getSystemService(mAppContext, DownloadManager::class.*java*)  
 downloadManager!!.enqueue(request)  
 }  
}

package com.example.innowisepexelstestapp.repository.firebase  
  
import android.annotation.SuppressLint  
import android.util.Log  
import com.example.innowisepexelstestapp.dto.PhotoPexelsDto  
import com.example.innowisepexelstestapp.mapper.PhotoPexelsMapper  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.repository.FavoritePhotoManager  
import com.google.firebase.database.DataSnapshot  
import com.google.firebase.database.DatabaseError  
import com.google.firebase.database.DatabaseReference  
import com.google.firebase.database.FirebaseDatabase  
import com.google.firebase.database.ValueEventListener  
import java.util.concurrent.CountDownLatch  
  
private const val *FIREBASE\_PHOTOS\_DB* = "FIREBASE\_PHOTOS\_DB"  
private const val *FIREBASE\_INSTANCE\_NAME* = "https://pexels-app-9497d-default-rtdb.europe-west1.firebasedatabase.app"  
  
class FirebaseFavoritePhotoManagerImpl : FavoritePhotoManager {  
  
 private val mMapper = PhotoPexelsMapper()  
  
 private val photosDbReference: DatabaseReference = FirebaseDatabase  
 .getInstance(*FIREBASE\_INSTANCE\_NAME*).getReference(*FIREBASE\_PHOTOS\_DB*)  
  
 @SuppressLint("CheckResult")  
 override fun **getAllFavoritePhoto**(): List<PhotoPexels> {  
 val photoDtoList = *mutableListOf*<PhotoPexelsDto>()  
 val latch = CountDownLatch(1)  
  
 photosDbReference.addListenerForSingleValueEvent(object : ValueEventListener {  
  
 override fun **onDataChange**(snapshot: DataSnapshot) {  
 for (photoSnapshot in snapshot.*children*) {  
 val photo = photoSnapshot.getValue(PhotoPexelsDto::class.*java*)  
 photo?.*let* {  
 photoDtoList.add(**it**)  
 }  
 }  
 latch.countDown()  
 }  
  
 override fun **onCancelled**(error: DatabaseError) {  
 error.toException().printStackTrace()  
 latch.countDown()  
 }  
  
 })  
  
 try {  
 latch.await()  
 } catch (e: InterruptedException) {  
 e.printStackTrace()  
 }  
 return mMapper.toModels(photoDtoList)  
 }  
  
 override fun **insertFavoritePhoto**(photoPexels: PhotoPexels) {  
 val photoPexelsDto = mMapper.toDto(photoPexels)  
 photosDbReference.child(photoPexelsDto.id.toString()).setValue(photoPexelsDto)  
 .addOnFailureListener { exception ->  
 exception.printStackTrace()  
 }  
 }  
  
 override fun **deleteFavoritePhoto**(photoPexels: PhotoPexels) {  
 val photoPexelsDto = mMapper.toDto(photoPexels)  
 photosDbReference.child(photoPexelsDto.id.toString()).removeValue()  
 .addOnFailureListener { exception ->  
 exception.printStackTrace()  
 }  
 }  
}

package com.example.innowisepexelstestapp.repository.firebase  
  
import com.example.innowisepexelstestapp.repository.SignInSignUpManager  
import com.google.firebase.auth.FirebaseAuth  
import io.reactivex.Single  
  
class SignInSignUpManagerImpl: SignInSignUpManager {  
  
 override fun **isUserSignedUp**(): Single<Boolean> {  
 return Single.create { emitter ->  
 emitter.onSuccess(FirebaseAuth.getInstance().*currentUser* != null)  
 }  
 }  
  
 override fun **signInUser**(email: String, password: String): Single<Boolean> {  
 return Single.create { emitter ->  
 FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)  
 .addOnSuccessListener {  
 emitter.onSuccess(true)  
 }  
 .addOnFailureListener {  
 emitter.onSuccess(false)  
 }  
 }  
 }  
  
 override fun **signUpUser**(email: String, password: String): Single<Boolean> {  
 return Single.create { emitter ->  
 FirebaseAuth.getInstance().createUserWithEmailAndPassword(email, password)  
 .addOnSuccessListener {  
 emitter.onSuccess(true)  
 }  
 .addOnFailureListener {  
 emitter.onSuccess(false)  
 }  
 }  
 }  
  
 override fun **logOutUser**(): Single<Boolean> {  
 return Single.create { emitter ->  
 FirebaseAuth.getInstance().signOut()  
 emitter.onSuccess(FirebaseAuth.getInstance().*currentUser* == null)  
 }  
 }  
}

package com.example.innowisepexelstestapp.repository.pexelsapi  
  
import com.example.innowisepexelstestapp.dto.CollectionsResultDto  
import com.example.innowisepexelstestapp.dto.CuratedResultDto  
import com.example.innowisepexelstestapp.mapper.CategoryMapper  
import com.example.innowisepexelstestapp.mapper.PhotoPexelsMapper  
import com.example.innowisepexelstestapp.model.Category  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.repository.NetworkManager  
import com.google.gson.Gson  
import io.reactivex.Single  
import io.reactivex.schedulers.Schedulers  
  
class NetworkManagerImpl(private val mNetworkClient: PexelsNetworkClient) : NetworkManager {  
  
 private val mPhotoMapper = PhotoPexelsMapper()  
 private val mCategoryMapper = CategoryMapper()  
 override fun **getCuratedPhotos**(): Single<List<PhotoPexels>> {  
 return Single.fromCallable {  
 val response = mNetworkClient.getResponseWithCuratedPhotos()  
 if (!response.isSuccessful) {  
 //*todo мб иначе обработать* throw Exception("Failed to get curated photos")  
 }  
  
 val curatedResultDto: CuratedResultDto = Gson().fromJson(response.body?.string(), CuratedResultDto::class.*java*)  
 return@fromCallable mPhotoMapper.toModels(curatedResultDto.photos)  
 }.subscribeOn(Schedulers.computation())  
 }  
  
 override fun **getCategories**(): Single<List<Category>> {  
 return Single.fromCallable {  
 val response = mNetworkClient.getResponseWithFeaturedCategories()  
 if (!response.isSuccessful) {  
 //*todo мб иначе обработать* throw Exception("Failed to get curated photos")  
 }  
 val collectionsResultDto: CollectionsResultDto = Gson().fromJson(response.body?.string(), CollectionsResultDto::class.*java*)  
 return@fromCallable mCategoryMapper.toModels(collectionsResultDto.collections)  
 }.subscribeOn(Schedulers.computation())  
 }  
  
 override fun **getQueryPhotos**(query: String): Single<List<PhotoPexels>> {  
 return Single.fromCallable {  
 val response = mNetworkClient.getResponseWithQueryPhotos(query)  
 if (!response.isSuccessful) {  
 //*todo мб иначе обработать* throw Exception("Failed to get curated photos")  
 }  
  
 val curatedResultDto: CuratedResultDto = Gson().fromJson(response.body?.string(), CuratedResultDto::class.*java*)  
 return@fromCallable mPhotoMapper.toModels(curatedResultDto.photos)  
 }.subscribeOn(Schedulers.computation())  
 }  
}

package com.example.innowisepexelstestapp.repository.pexelsapi  
  
import android.util.Log  
import okhttp3.OkHttpClient  
import okhttp3.Request  
import okhttp3.Response  
  
const val *KEY* = "1yRhwLBuxcoFBCLNlvFi7d00HeHhnv7fOzvYXsYTrbVBUV0RFk157lqx"  
  
const val *URL\_GET\_CURATED* = "https://api.pexels.com/v1/curated"  
const val *URL\_GET\_FEATURED\_COLLECTIONS* = "https://api.pexels.com/v1/collections/featured"  
const val *URL\_GET\_SEARCH* = "https://api.pexels.com/v1/search"  
  
const val *PER\_PAGE\_FEATURED\_COLLECTIONS* = "per\_page=7"  
const val *PER\_PAGE\_30* = "per\_page=30"  
const val *PAGE* = "page="  
const val *QUERY* = "query="  
  
class PexelsNetworkClient(private val mClient: OkHttpClient) {  
  
 private var pageCurated: Int = 1  
 private var pageQuery: Int = 1  
 fun **getResponseWithCuratedPhotos**(): Response {  
 Log.e("customLog", "CURATED")  
 val request = Request.Builder()  
 .url("$*URL\_GET\_CURATED*?$*PAGE*${pageCurated++}&$*PER\_PAGE\_30*")  
 .header("Authorization", *KEY*)  
 .get()  
 .build()  
 return mClient.newCall(request).execute()  
 }  
  
 fun **getResponseWithFeaturedCategories**(): Response {  
 Log.e("customLog", "CATEGORIES")  
 val request = Request.Builder()  
 .url("$*URL\_GET\_FEATURED\_COLLECTIONS*?$*PER\_PAGE\_FEATURED\_COLLECTIONS*")  
 .header("Authorization", *KEY*)  
 .get()  
 .build()  
 return mClient.newCall(request).execute()  
 }  
  
 fun **getResponseWithQueryPhotos**(query: String): Response {  
 Log.e("customLog", "QUERY")  
 val request = Request.Builder()  
 .url("$*URL\_GET\_SEARCH*?$*QUERY*$query&$*PAGE*${pageQuery++}&$*PER\_PAGE\_30*")  
 .header("Authorization", *KEY*)  
 .get()  
 .build()  
 return mClient.newCall(request).execute()  
 }  
  
}

package com.example.innowisepexelstestapp.repository.room  
  
import androidx.room.Database  
import androidx.room.RoomDatabase  
import androidx.room.TypeConverters  
import com.example.innowisepexelstestapp.dto.PhotoPexelsDto  
  
@Database(entities = [PhotoPexelsDto::class], version = 1)  
@TypeConverters(ImageSourcesConverter::class)  
abstract class FavoritePhotoDataBase : RoomDatabase() {  
 abstract fun **photoPexelsDao**(): PhotoPexelsDao  
}

package com.example.innowisepexelstestapp.repository.room  
  
import androidx.room.TypeConverter  
import com.example.innowisepexelstestapp.dto.ImageSourcesDto  
import com.google.gson.Gson  
  
object ImageSourcesConverter {  
  
 @TypeConverter  
 @JvmStatic  
 fun **fromImageSourcesEntity**(imageSources: ImageSourcesDto): String {  
 return Gson().toJson(imageSources)  
 }  
  
 @TypeConverter  
 @JvmStatic  
 fun **toImageSourcesEntity**(imageSourcesString: String): ImageSourcesDto {  
 return Gson().fromJson(imageSourcesString, ImageSourcesDto::class.*java*)  
 }  
}

package com.example.innowisepexelstestapp.repository.room  
  
import androidx.room.Dao  
import androidx.room.Delete  
import androidx.room.Insert  
import androidx.room.OnConflictStrategy  
import androidx.room.Query  
import com.example.innowisepexelstestapp.dto.PhotoPexelsDto  
  
@Dao  
interface PhotoPexelsDao {  
 @Query("SELECT \* FROM PhotoPexelsDto")  
 fun **getAllPhotoPexels**(): List<PhotoPexelsDto> //*todo хз может ли автоматически создаться оболочка Single* @Insert(onConflict = OnConflictStrategy.IGNORE)  
 fun **insertPhotoPexels**(photoPexelsDto: PhotoPexelsDto)  
  
 @Delete  
 fun **deletePhotoPexels**(photoPexelsDto: PhotoPexelsDto)  
}

package com.example.innowisepexelstestapp.repository.room  
  
import com.example.innowisepexelstestapp.mapper.PhotoPexelsMapper  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.repository.FavoritePhotoManager  
  
class RoomFavoritePhotoManagerImpl(private val photoPexelsDao: PhotoPexelsDao) : FavoritePhotoManager {  
  
 private val mMapper = PhotoPexelsMapper()  
 override fun **getAllFavoritePhoto**(): List<PhotoPexels> { //*todo добавить single* return mMapper.toModels(photoPexelsDao.getAllPhotoPexels())  
 }  
  
 override fun **insertFavoritePhoto**(photoPexels: PhotoPexels) {  
 photoPexelsDao.insertPhotoPexels(mMapper.toDto(photoPexels))  
 }  
  
 override fun **deleteFavoritePhoto**(photoPexels: PhotoPexels) {  
 photoPexelsDao.deletePhotoPexels(mMapper.toDto(photoPexels))  
 }  
}

package com.example.innowisepexelstestapp.model  
  
import java.io.Serializable  
  
data class Category(  
 val name: String,  
 var isActive: Boolean  
) : Serializable

package com.example.innowisepexelstestapp.model  
  
import com.google.gson.annotations.SerializedName  
  
data class CuratedResult(  
 val page: Int,  
 val itemsPerPage: String,  
 val photos: List<PhotoPexels>,  
 val previousPageURL: String?,  
 val nextPageURL: String?  
)

package com.example.innowisepexelstestapp.model  
  
import java.io.Serializable  
  
data class PhotoPexels(  
 val id: Int,  
 val photographer: String,  
 val sources: ImageSources,  
) : Serializable  
  
data class ImageSources(  
 val original: String,  
 val medium: String,  
 val small: String,  
) : Serializable

package com.example.innowisepexelstestapp.repository  
  
interface DownloadFilesManager {  
 fun **download**(url: String)  
}

package com.example.innowisepexelstestapp.repository  
  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import io.reactivex.Single  
  
interface FavoritePhotoManager {  
  
 fun **getAllFavoritePhoto**(): List<PhotoPexels>  
  
 fun **insertFavoritePhoto**(photoPexels: PhotoPexels)  
  
 fun **deleteFavoritePhoto**(photoPexels: PhotoPexels)  
}

package com.example.innowisepexelstestapp.repository  
  
import com.example.innowisepexelstestapp.model.Category  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import io.reactivex.Single  
  
interface NetworkManager {  
 fun **getCuratedPhotos**(): Single<List<PhotoPexels>>  
  
 fun **getQueryPhotos**(query: String): Single<List<PhotoPexels>>  
  
 fun **getCategories**(): Single<List<Category>>  
  
}

package com.example.innowisepexelstestapp.repository  
  
import io.reactivex.Single  
  
interface SignInSignUpManager {  
  
 fun **isUserSignedUp**(): Single<Boolean>  
  
 fun **signInUser**(email: String, password: String): Single<Boolean>  
  
 fun **signUpUser**(email: String, password: String): Single<Boolean>  
  
 fun **logOutUser**(): Single<Boolean>  
}

package com.example.innowisepexelstestapp.usecase  
  
import com.example.innowisepexelstestapp.model.PhotoPexels  
import com.example.innowisepexelstestapp.repository.FavoritePhotoManager  
  
class DeleteImageFromBdUseCase(private val mFavoritePhotoManager: FavoritePhotoManager) {  
 fun **execute**(photoPexels: PhotoPexels) {  
 mFavoritePhotoManager.deleteFavoritePhoto(photoPexels)  
 }  
}

package com.example.innowisepexelstestapp.usecase  
  
import com.example.innowisepexelstestapp.repository.DownloadFilesManager  
  
class DownloadImageUseCase(private val downloadFilesManager: DownloadFilesManager) {  
 fun **execute**(url: String) {  
 downloadFilesManager.download(url)  
 }  
}

package com.example.innowisepexelstestapp.usecase  
  
import com.example.innowisepexelstestapp.repository.SignInSignUpManager  
import io.reactivex.Single  
  
enum class SignInState {  
 *SIGN\_IN\_FAILED*,  
 *SIGN\_IN\_SUCCESSFUL*,  
 *EMPTY\_EMAIL*,  
}  
  
class SignInUseCase(private val signInSignUpManager: SignInSignUpManager) {  
  
 @Suppress("CheckResult")  
 fun **execute**(email: String, password: String): Single<SignInState> {  
 return Single.create<SignInState> { emitter ->  
 if (email.*isEmpty*()) {  
 emitter.onSuccess(SignInState.*EMPTY\_EMAIL*)  
 } else {  
 signInSignUpManager.signInUser(email, password).subscribe { isUserSignedUp ->  
 if (isUserSignedUp) {  
 emitter.onSuccess(SignInState.*SIGN\_IN\_SUCCESSFUL*)  
 } else {  
 emitter.onSuccess(SignInState.*SIGN\_IN\_FAILED*)  
 }  
 }  
 }  
 }  
 }  
}

package com.example.innowisepexelstestapp.usecase  
  
import com.example.innowisepexelstestapp.repository.SignInSignUpManager  
import io.reactivex.Single  
  
enum class SignUpState {  
 *INVALID\_PASSWORD*,  
 *PASSWORDS\_ARE\_NOT\_EQUAL*,  
 *SIGN\_UP\_FAILED*,  
 *SIGN\_UP\_SUCCESSFUL*,  
 *EMPTY\_EMAIL*}  
  
class SignUpUseCase(private val signInSignUpManager: SignInSignUpManager) {  
  
 @Suppress("CheckResult")  
 fun **execute**(email: String, password: String, confirmPassword: String): Single<SignUpState> {  
 //password contains at least one uppercase letter, one lowercase letter, one number and >5 symbols  
 val regex = Regex("^(?=.\*[A-Z])(?=.\*[a-z])(?=.\*\\d).{6,}$")  
  
 return Single.create<SignUpState> { emitter ->  
 if (email.*isEmpty*()) {  
 emitter.onSuccess(SignUpState.*EMPTY\_EMAIL*)  
 }  
 else if (password != confirmPassword) {  
 emitter.onSuccess(SignUpState.*PASSWORDS\_ARE\_NOT\_EQUAL*)  
 } else if (!regex.matches(password)) {  
 emitter.onSuccess(SignUpState.*INVALID\_PASSWORD*)  
 } else {  
 signInSignUpManager.signUpUser(email, password).subscribe { isUserSignedUp ->  
 if (isUserSignedUp) {  
 emitter.onSuccess(SignUpState.*SIGN\_UP\_SUCCESSFUL*)  
 } else {  
 emitter.onSuccess(SignUpState.*SIGN\_UP\_FAILED*)  
 }  
 }  
 }  
 }  
 }  
}

**4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

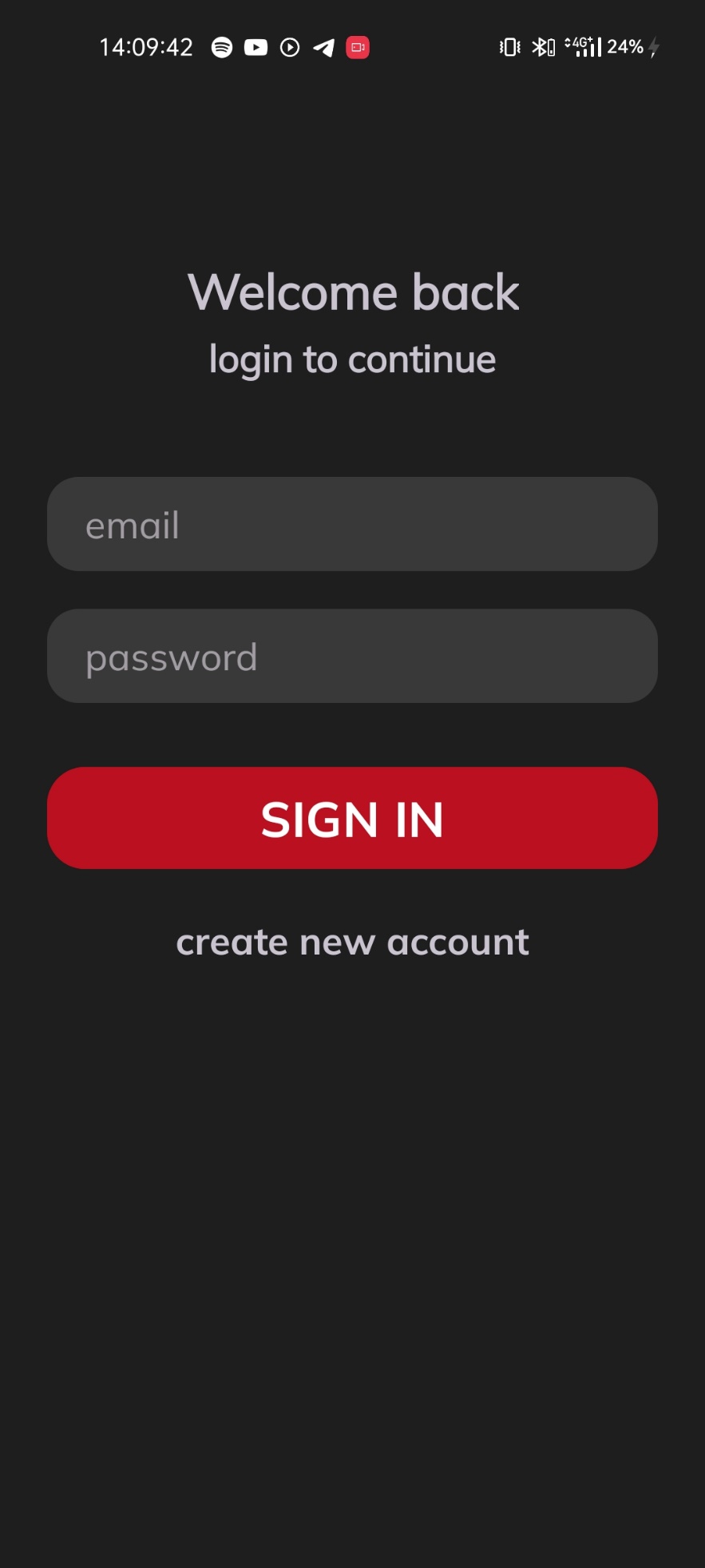


Рисунок 4.1 – Экран входа в аккаунт

A screenshot of a login form

Description automatically generated

Рисунок 4.2 – Экран регистрации



Рисунок 4.3 – Экран просмотра фотографии

A black background with white text

Description automatically generated

Рисунок 4.4 – Экран списка сохраненных фотографий

A screenshot of a cellphone

Description automatically generated

Рисунок 4.5 – Экран списка предлагаемых фотографий

**5 ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы был разработан проект с 5 фрагментами отображения интерфейса и возможностью взаимодействовать между ними. Была выстроена clean-архитектура для возможности дальнейшей расширяемости проекта

Окружение: Android SDK, Kotlin, XML, RxKotlin, Cicerone, Dagger 2, OkHttp, Room, MVVM, Firebase auth, Firebase Realtime Database