**ПРИЛОЖЕНИЕ А**

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

# КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

СИСТЕМА ДЕТЕКЦИИ И РАСПОЗНАВАНИЯ АВТОМОБИЛЬНЫХ ЗНАКОВ

**ТЕКСТ ПРОГРАММЫ**

КП.ПО7.200149 – 05 12 00

Листов 10

|  |  |
| --- | --- |
| Руководитель | Е.В. Михно |
| Выполнил | А.Е. Комиссаров |
| Консультант |  |
| по ЕСПД | Е.В. Михно |
|  |  |

2023

**СОДЕРЖАНИЕ**

**ApplictaionRoot.kt –** первозапускаемый файл проекта, инициализирует остальные системы.

**CameraActivity.kt –** активность камеры.

**LoadActivity.kt –** активность загрузки изображения с файла.

**PermissionFragment.kt –** активность запроса разрешений.

**ApplicationRoot.kt**

package com.arpadfodor.stolenvehicledetector.android.app  
import android.Manifest  
import android.app.Application  
import android.os.Build  
import android.util.Log  
import androidx.preference.PreferenceManager  
import com.arpadfodor.stolenvehicledetector.android.app.model.\*  
import com.arpadfodor.stolenvehicledetector.android.app.model.ml.detector.ObjectDetectionService  
import com.arpadfodor.stolenvehicledetector.android.app.model.ml.VehicleIdentifierService  
import com.arpadfodor.stolenvehicledetector.android.app.model.api.ApiService  
import com.arpadfodor.stolenvehicledetector.android.app.model.ml.ocr.OCRService  
import com.arpadfodor.stolenvehicledetector.android.app.model.repository.GeneralRepository  
import com.arpadfodor.stolenvehicledetector.android.app.viewmodel.CameraViewModel  
import java.util.\*  
class ApplicationRoot : Application() {  
 companion object{  
 private const val TAG = "Application Root"  
 private const val NUM\_THREADS = 4  
 const val IMMERSIVE\_FLAG\_TIMEOUT = 100L  
 var requiredPermissions = if(Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*Q*){  
 *arrayOf*(  
 Manifest.permission.*CAMERA*,  
 Manifest.permission.*INTERNET*,  
 Manifest.permission.*READ\_EXTERNAL\_STORAGE*,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION*,  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*,  
 Manifest.permission.*ACCESS\_MEDIA\_LOCATION*)  
 }  
 else{  
 *arrayOf*(  
 Manifest.permission.*CAMERA*,  
 Manifest.permission.*INTERNET*,  
 Manifest.permission.*READ\_EXTERNAL\_STORAGE*,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION*,  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*)  
 }  
 var isAutoSyncEnabled = true  
 var keepScreenAlive = true  
 }  
 override fun onCreate() {  
 super.onCreate()  
 Log.i(TAG, "onCreate fired")  
 GeneralRepository.initialize(*applicationContext*)  
 ApiService.initialize(AccountService.getClient())  
 ObjectDetectionService.initialize(*assets*, NUM\_THREADS)  
 OCRService.initialize(*assets*, NUM\_THREADS)  
 val radius = *resources*.getDimension(R.dimen.*bounding\_box\_radius*)  
 val width = *resources*.getDimension(R.dimen.*bounding\_box\_line\_width*)  
 val textSize = *resources*.getDimension(R.dimen.*bounding\_box\_text\_size*)  
 val colors = *resources*.getStringArray(R.array.*bounding\_box\_colors*)  
 val alertColor = *resources*.getColor(R.color.*colorAlertBoundingBox*, null)  
 BoundingBoxDrawer.initialize(radius, width, textSize, colors, alertColor)  
 val appName = getString(R.string.*app\_name*)  
 MediaHandler.initialize(*applicationContext*, appName)  
 LocationService.initialize(*applicationContext*)  
 TextToSpeechService.init(*applicationContext*)  
 CameraViewModel.KEY\_EVENT\_ACTION = getString(R.string.*KEY\_EVENT\_ACTION*)  
 CameraViewModel.KEY\_EVENT\_EXTRA = getString(R.string.*KEY\_EVENT\_EXTRA*)  
 val preferences = PreferenceManager.getDefaultSharedPreferences(this)  
 isAutoSyncEnabled = preferences.getBoolean(getString(R.string.*SETTINGS\_AUTO\_SYNC*), *resources*.getBoolean(R.bool.*settings\_auto\_sync\_default*))  
 keepScreenAlive = preferences.getBoolean(getString(R.string.*SETTINGS\_KEEP\_SCREEN\_ALIVE*), *resources*.getBoolean(R.bool.*settings\_keep\_screen\_alive\_default*))  
 if(isAutoSyncEnabled){  
 GeneralRepository.updateAll**{** isVehiclesSuccess, isReportsSuccess **->** if(isVehiclesSuccess){  
 val currentTime = Calendar.getInstance().*time*.toString()  
 preferences.edit().putString(getString(R.string.*LAST\_SYNCED\_DB\_VEHICLES*), currentTime)  
 .apply()  
 }  
 VehicleIdentifierService.initialize()  
 if(isReportsSuccess){  
 val currentTime = Calendar.getInstance().*time*.toString()  
 preferences.edit().putString(getString(R.string.*LAST\_SYNCED\_DB\_REPORTS*), currentTime)  
 .apply()  
 }  
 **}** }  
 }  
}

**CameraActivity.kt**

package com.arpadfodor.stolenvehicledetector.android.app.view  
import android.content.Intent  
import android.hardware.SensorManager  
import android.os.Bundle  
import android.view.KeyEvent  
import android.view.OrientationEventListener  
import android.view.View.*SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN*import android.view.View.*SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE*import android.widget.FrameLayout  
import androidx.drawerlayout.widget.DrawerLayout  
import androidx.lifecycle.ViewModelProvider  
import androidx.localbroadcastmanager.content.LocalBroadcastManager  
import androidx.preference.PreferenceManager  
import com.arpadfodor.stolenvehicledetector.android.app.ApplicationRoot  
import com.arpadfodor.stolenvehicledetector.android.app.R  
import com.arpadfodor.stolenvehicledetector.android.app.view.utils.AppActivity  
import com.arpadfodor.stolenvehicledetector.android.app.view.utils.overshootAppearingAnimation  
import com.arpadfodor.stolenvehicledetector.android.app.viewmodel.CameraViewModel  
import com.google.android.material.navigation.NavigationView  
import kotlinx.android.synthetic.main.content\_camera.\*  
class CameraActivity : AppActivity() {  
 override lateinit var viewModel: CameraViewModel  
 private lateinit var container: FrameLayout  
 lateinit var deviceOrientationListener: OrientationEventListener  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_camera*)  
 container = findViewById(R.id.*camera\_container*)  
 val drawer = findViewById<DrawerLayout>(R.id.*cameraActivityDrawerLayout*)  
 val navigation = findViewById<NavigationView>(R.id.*camera\_navigation*)  
 initUi(drawer, navigation)  
 viewModel = ViewModelProvider(this).get(CameraViewModel::class.*java*)  
 showCameraFragment()  
 deviceOrientationListener = object : OrientationEventListener(this,  
 SensorManager.*SENSOR\_DELAY\_NORMAL*) {  
 override fun onOrientationChanged(orientation: Int) {  
 CameraViewModel.deviceOrientation = orientation  
 }  
 }  
 }  
 override fun subscribeToViewModel() {  
 val settings = PreferenceManager.getDefaultSharedPreferences(*applicationContext*)  
 val settingsNumRecognitionsKey = getString(R.string.*SETTINGS\_NUM\_RECOGNITIONS*)  
 val settingsMinimumPredictionCertaintyKey = getString(R.string.*SETTINGS\_MINIMUM\_PREDICTION\_CERTAINTY*)  
 val settingsShowReceptiveFieldKey = getString(R.string.*SETTINGS\_SHOW\_RECEPTIVE\_FIELD*)  
 val numRecognitionsToShow = settings.getInt(settingsNumRecognitionsKey, *resources*.getInteger(R.integer.*settings\_num\_recognitions\_default*))  
 val minimumPredictionCertaintyToShow = settings.getInt(settingsMinimumPredictionCertaintyKey, *resources*.getInteger(R.integer.*settings\_minimum\_prediction\_certainty\_default*))  
 val settingsShowReceptiveField = settings.getBoolean(settingsShowReceptiveFieldKey, *resources*.getBoolean(R.bool.*settings\_receptive\_field\_default*))  
 CameraViewModel.numRecognitionsToShow = numRecognitionsToShow  
 CameraViewModel.minimumPredictionCertaintyToShow = minimumPredictionCertaintyToShow.toFloat()  
 CameraViewModel.settingsShowReceptiveField = settingsShowReceptiveField  
 }  
 override fun subscribeListeners() {  
 deviceOrientationListener.enable()  
 container.postDelayed(**{** container.*systemUiVisibility* = (*SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN* and *SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE*)  
 appearingAnimations()  
 **}**, ApplicationRoot.IMMERSIVE\_FLAG\_TIMEOUT)  
 }  
 override fun unsubscribe() {  
 deviceOrientationListener.disable()  
 }  
 private fun showCameraFragment(){  
 *supportFragmentManager* .beginTransaction()  
 .replace(R.id.*camera\_container*, CameraFragment())  
 .commit()  
 }  
 override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean {  
 return when (keyCode){  
 KeyEvent.*KEYCODE\_VOLUME\_DOWN* -> {  
 val intent = Intent(CameraViewModel.KEY\_EVENT\_ACTION).*apply* **{** putExtra(CameraViewModel.KEY\_EVENT\_EXTRA, keyCode)  
 **}** LocalBroadcastManager.getInstance(this).sendBroadcast(intent)  
 true  
 }  
 else -> {  
 super.onKeyDown(keyCode, event)  
 }  
 }  
 }  
 override fun appearingAnimations() {  
 camera\_switch\_button?.*overshootAppearingAnimation*(this)  
 camera\_capture\_button?.*overshootAppearingAnimation*(this)  
 }  
}

**LoadActivity.kt**

package com.arpadfodor.stolenvehicledetector.android.app.view  
import android.content.Intent  
import android.graphics.Bitmap  
import android.net.Uri  
import android.os.Bundle  
import android.util.DisplayMetrics  
import android.view.View  
import androidx.constraintlayout.widget.ConstraintLayout  
import androidx.core.content.ContextCompat  
import androidx.drawerlayout.widget.DrawerLayout  
import androidx.lifecycle.Observer  
import androidx.lifecycle.ViewModelProvider  
import androidx.preference.PreferenceManager  
import com.arpadfodor.stolenvehicledetector.android.app.ApplicationRoot  
import com.arpadfodor.stolenvehicledetector.android.app.R  
import com.arpadfodor.stolenvehicledetector.android.app.viewmodel.LoadViewModel  
import com.arpadfodor.stolenvehicledetector.android.app.model.repository.dataclasses.UserRecognition  
import com.arpadfodor.stolenvehicledetector.android.app.view.utils.\*  
import com.bumptech.glide.Glide  
import com.google.android.material.navigation.NavigationView  
import kotlinx.android.synthetic.main.content\_load.\*  
class LoadActivity : AppActivity() {  
 override lateinit var viewModel: LoadViewModel  
 private lateinit var container: ConstraintLayou  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_load*)  
 container = findViewById(R.id.*loaded\_image\_container*)  
 val drawer = findViewById<DrawerLayout>(R.id.*loadActivityDrawerLayout*)  
 val navigation = findViewById<NavigationView>(R.id.*load\_navigation*)  
 viewModel = ViewModelProvider(this).get(LoadViewModel::class.*java*)  
 initUi(drawer, navigation)  
 ivLoadedImage.*clipToOutline* = true  
 Glide  
 .with(this)  
 .load(R.drawable.*icon\_photo\_library*)  
 .into(ivLoadedImage)  
 extendedFabLoadHelp.setOnClickListener **{** loadImage()  
 **}** extendedFabLoadHelp.*text* = getString(R.string.*load\_an\_image*)  
 extendedFabLoadHelp.*icon* = ContextCompat.getDrawable(this, R.drawable.*icon\_photo\_library*)  
 extendedFabLoadHelp.*iconTint* = ContextCompat.getColorStateList(this, R.color.*selector\_ic*)  
 extendedFabLoadHelp.*backgroundTintList* = ContextCompat.getColorStateList(this, R.color.*selector\_fab\_normal\_color*)  
 val myUri = *intent*.getParcelableExtra<Uri>("fromCameraUri")  
 if (myUri != null) {  
 viewModel.loadImage(myUri)**{**isSuccess **->** if(!isSuccess){  
 showErrorSnackBar(getString(R.string.*image\_load\_failed*))  
 }  
 **}** }  
 }  
 override fun onResume() {  
 super.onResume()  
 val metrics = DisplayMetrics()  
 *windowManager*.*defaultDisplay*.getMetrics(metrics)  
 viewModel.setScreenProperties(metrics.widthPixels, metrics.heightPixels)  
 container.postDelayed(**{** container.*systemUiVisibility* = (View.*SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN* and View.*SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE*)  
 **}**, ApplicationRoot.IMMERSIVE\_FLAG\_TIMEOUT)  
 val settings = PreferenceManager.getDefaultSharedPreferences(*applicationContext*)  
 val settingsNumRecognitionsKey = getString(R.string.*SETTINGS\_NUM\_RECOGNITIONS*)  
 val settingsMinimumPredictionCertaintyKey = getString(R.string.*SETTINGS\_MINIMUM\_PREDICTION\_CERTAINTY*)  
 val numRecognitionsToShow = settings.getInt(settingsNumRecognitionsKey, *resources*.getInteger(R.integer.*settings\_num\_recognitions\_default*))  
 val minimumPredictionCertaintyToShow = settings.getInt(settingsMinimumPredictionCertaintyKey, *resources*.getInteger(R.integer.*settings\_minimum\_prediction\_certainty\_default*))  
 LoadViewModel.numRecognitionsToShow = numRecognitionsToShow  
 LoadViewModel.minimumPredictionCertaintyToShow = minimumPredictionCertaintyToShow.toFloat()  
 }  
 override fun subscribeToViewModel() {  
 val imageObserver = *Observer*<Bitmap> **{** newImage **->** ivLoadedImage.*disappearingAnimation*(this)  
 Glide  
 .with(this)  
 .load(newImage)  
 .centerCrop()  
 .error(R.drawable.*icon\_photo\_library*)  
 .placeholder(R.drawable.*icon\_photo\_library*)  
 .into(ivLoadedImage)  
 ivLoadedImage.*appearingAnimation*(this)  
 **}** val boundingBoxImageObserver = *Observer*<Bitmap> **{** newImage **->** Glide  
 .with(this)  
 .load(newImage)  
 .centerCrop()  
 .into(ivLoadedImageBoundingBoxes)  
 **}** val recognitionsObserver = *Observer*<Array<UserRecognition>> **{** recognitions **->** if(recognitions.*isNotEmpty*()){  
 alertLoadedButton.setOnClickListener **{** viewModel.setAlertActivityParams()  
 val intent = Intent(this, AlertActivity::class.*java*)  
 startActivity(intent)  
 **}** extendedFabLoadHelp.setOnClickListener **{** viewModel.setAlertActivityParams()  
 val intent = Intent(this, AlertActivity::class.*java*)  
 startActivity(intent)  
 **}** if(alertLoadedButton.*visibility* == View.*GONE*){  
 alertLoadedButton.*appearingAnimation*(this)  
 }  
 extendedFabLoadHelp.*text* = getString(R.string.*view\_alert*)  
 extendedFabLoadHelp.*icon* = ContextCompat.getDrawable(this, android.R.drawable.*ic\_dialog\_alert*)  
 extendedFabLoadHelp.*iconTint* = ContextCompat.getColorStateList(this, R.color.*selector\_ic*)  
 extendedFabLoadHelp.*backgroundTintList* = ContextCompat.getColorStateList(this, R.color.*selector\_fab\_alert\_color*)  
 }  
 else{  
 if(alertLoadedButton.*visibility* == View.*VISIBLE*){  
 alertLoadedButton.*disappearingAnimation*(this)  
 }  
 if(viewModel.loadedImage.*value* != null){  
 extendedFabLoadHelp.setOnClickListener **{}** extendedFabLoadHelp.*text* = getString(R.string.*inspected*)  
 extendedFabLoadHelp.*icon* = ContextCompat.getDrawable(this, R.drawable.*icon\_done*)  
 }  
 else{  
 extendedFabLoadHelp.setOnClickListener **{** loadImage()  
 **}** extendedFabLoadHelp.*text* = getString(R.string.*load\_an\_image*)  
 extendedFabLoadHelp.*icon* = ContextCompat.getDrawable(this, R.drawable.*icon\_photo\_library*)  
 }  
 extendedFabLoadHelp.*iconTint* = ContextCompat.getColorStateList(this, R.color.*selector\_ic*)  
 extendedFabLoadHelp.*backgroundTintList* = ContextCompat.getColorStateList(this, R.color.*selector\_fab\_normal\_color*)  
 }  
 **}** viewModel.loadedImage.observe(this, imageObserver)  
 viewModel.boundingBoxImage.observe(this, boundingBoxImageObserver)  
 viewModel.recognitions.observe(this, recognitionsObserver)  
 }  
 override fun subscribeListeners() {  
 ivLoadedImage.setOnClickListener **{** if(viewModel.loadedImage.*value* == null){  
 loadImage()  
 }  
 **}** load\_image\_button.setOnClickListener **{** loadImage()  
 **}** loaded\_image\_rotate\_button.setOnClickListener **{** viewModel.rotateImage()  
 **}** }  
 override fun unsubscribe() {}  
 private fun loadImage(){  
 val intent = Intent(Intent.*ACTION\_PICK*)  
 intent.*type* = "image/\*"  
 intent.putExtra(Intent.*EXTRA\_MIME\_TYPES*, viewModel.imageMimeTypes)  
 startActivityForResult(intent, LoadViewModel.GALLERY\_REQUEST\_CODE)  
 }  
 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 super.onActivityResult(requestCode, resultCode, data)  
 if (resultCode == *RESULT\_OK*) {  
 when (requestCode) {  
 LoadViewModel.GALLERY\_REQUEST\_CODE -> {  
 val selectedImageUri = data?.*data* ?: return  
 viewModel.loadImage(selectedImageUri)**{**isSuccess **->** if(!isSuccess){  
 showErrorSnackBar(getString(R.string.*image\_load\_failed*))  
 }  
 **}** }  
 }  
 }  
 }  
 override fun appearingAnimations(){  
 load\_image\_button.*overshootAppearingAnimation*(this)  
 loaded\_image\_rotate\_button.*overshootAppearingAnimation*(this)  
 extendedFabLoadHelp.*overshootAppearingAnimation*(this)  
 }  
}

**PermissionFragment.kt**

package com.arpadfodor.stolenvehicledetector.android.app.view  
import android.content.Context  
import android.content.pm.PackageManager  
import android.os.Bundle  
import androidx.core.content.ContextCompat  
import com.arpadfodor.stolenvehicledetector.android.app.ApplicationRoot  
import com.arpadfodor.stolenvehicledetector.android.app.view.utils.AppFragment  
import java.lang.System.exit  
import kotlin.system.exitProcess  
private const val *PERMISSIONS\_REQUEST\_CODE* = 10  
private val *PERMISSIONS\_REQUIRED* = ApplicationRoot.requiredPermissions  
class PermissionsFragment(finished: () -> Unit = **{}**) : AppFragment() {  
 companion object {  
 fun hasPermissions(context: Context) = *PERMISSIONS\_REQUIRED*.*all* **{** ContextCompat.checkSelfPermission(context, **it**) == PackageManager.*PERMISSION\_GRANTED* **}** }  
 val actionFinished = finished  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 if (!hasPermissions(requireContext())) {  
 // Request permissions  
 requestPermissions(*PERMISSIONS\_REQUIRED*, *PERMISSIONS\_REQUEST\_CODE*)  
 }  
 else {  
 removeThisFragment()  
 }  
 }  
 override fun appearingAnimations(){}  
 override fun subscribeToViewModel(){}  
 override fun subscribeListeners(){}  
 override fun unsubscribe(){}  
 override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>, grantResults: IntArray) {  
 super.onRequestPermissionsResult(requestCode, permissions, grantResults)  
 if (requestCode == *PERMISSIONS\_REQUEST\_CODE*) {  
 if(grantResults.*isEmpty*()){  
 *exitProcess*(0)  
 }  
 if (grantResults.*contains*(PackageManager.*PERMISSION\_DENIED*)) {  
 *exitProcess*(0)  
 }  
 else {  
 //granted  
 }  
 removeThisFragment()  
 }  
 }  
 private fun removeThisFragment(){  
 requireActivity().*supportFragmentManager*.beginTransaction().remove(this).commit()  
 actionFinished()  
 }  
}

**RecognitionActivity.kt**

package com.arpadfodor.stolenvehicledetector.android.app.view  
  
import android.os.Bundle  
import androidx.lifecycle.ViewModelProvider  
import com.arpadfodor.stolenvehicledetector.android.app.R  
import com.arpadfodor.stolenvehicledetector.android.app.view.utils.MasterDetailActivity  
import com.arpadfodor.stolenvehicledetector.android.app.viewmodel.RecognitionViewModel  
  
class RecognitionActivity : MasterDetailActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 viewModel = ViewModelProvider(this).get(RecognitionViewModel::class.*java*)  
 (viewModel as RecognitionViewModel).updateDataFromDb()  
 super.onCreate(savedInstanceState)  
  
 listName = getString(R.string.*user\_recognition\_list*)  
 detailName = getString(R.string.*user\_recognition\_details*)  
  
 sendSucceed = getString(R.string.*recognition\_sent*)  
 sendFailed = getString(R.string.*recognition\_sending\_failed*)  
 deleted = getString(R.string.*deleted*)  
 deleteFailed = getString(R.string.*delete\_failed*)  
 alreadySent = getString(R.string.*recognition\_already\_sent*)  
 updateSucceed = getString(R.string.*updated*)  
 updateFailed = getString(R.string.*update\_failed*)  
  
 }  
  
 override fun onResume() {  
 super.onResume()  
 (viewModel as RecognitionViewModel).updateDataFromDb()  
 }  
  
}