**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Доцент департамента  программной инженерии  факультета компьютерных наук,  канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Л. Макаров  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **АНДРОИД-ПРИЛОЖЕНИЕ "ЗЕЛЕНАЯ ВОЛНА ДЛЯ ПЕШЕХОДОВ"**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.16-01 12** **01-1-ЛУ** | | |
|  |  | |
| Исполнитель  студент группы БПИ162  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Н.О. Скудняков /  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. | |
|  | | |
|  | |  |

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.16-01 12 01-1-ЛУ |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **АНДРОИД-ПРИЛОЖЕНИЕ "ЗЕЛЕНАЯ ВОЛНА ДЛЯ ПЕШЕХОДОВ"**  **Текст программы**  **RU.17701729.04.16-01 12 01-1**  **Листов 36** | | |
|  |  | |
|  | |

**АННОТАЦИЯ**

В данном программном документе приведет текст «Андроид-приложение "Зеленая волна для пешеходов"». Текст программы реализован в виде символической записи на исходном языке. Исходным языком данной разработки является Kotlin 1.2.31. Среда разработки – Android Studio 3.1

Основная функция приложения – расчет времени до прибытия к светофору и подсказка пользователю, с какой скоростью следует двигаться, чтобы горел зеленый свет ко времени прибытия. Программой может пользоваться любой пешеход со смартфоном на операционной системе Android версии выше 4.2. Пользователь вводит расписание светофоров и периодичность смены сигналов. Далее при приближении к светофору с уже установленным расписанием пользователь сможет плавно скорректировать свою скорость, чтобы она совпадала с рекомендуемой скоростью движения. Таким образом, будет осуществляться безостановочное движение пользователя по регулируемым перекресткам.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов [1];

2) ГОСТ 19.102-77 Стадии разработки [2];

3) ГОСТ 19.103-77 Обозначения программ и программных документов [3];

4) ГОСТ 19.104-78 Основные надписи [4];

5) ГОСТ 19.105-78 Общие требования к программным документам [5];

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

[6];

7) ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению [7].

Изменения к данному документу оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-

78 [9].

**СОДЕРЖАНИЕ**

[1. ТЕКСТ ПРОГРАММЫ 3](#_Toc513046460)

[1.1 Класс LightSettings 3](#_Toc513046461)

[1.2 Класс Storage 5](#_Toc513046462)

[1.3 Класс TrafficLight 6](#_Toc513046463)

[1.4 Класс SettingsActivity 6](#_Toc513046464)

[1.5 Класс GreenwaveActivity 9](#_Toc513046465)

[1.6 Класс GreenwaveModel 17](#_Toc513046466)

[1.7 Интерфейс GreenwaveModelApi 22](#_Toc513046467)

[1.8 Класс GreenwavePresenter 22](#_Toc513046468)

[1.9 Интерфейс GreenwavePresenterApi 28](#_Toc513046469)

[1.10 Интерфейс GreenwaveView 29](#_Toc513046470)

[1.11 Класс CameraMovementLogicHelper 30](#_Toc513046471)

[1.12 Класс FirebaseDatabaseSingletone 30](#_Toc513046472)

[1.13 Класс GreenwaveConstants 31](#_Toc513046473)

[1.14 Класс MeanSpeed 31](#_Toc513046474)

[1.15 Класс OsmQueryResult 32](#_Toc513046475)

[1.16 Интерфейс OsmService 34](#_Toc513046476)

[ПРИЛОЖЕНИЕ 1 36](#_Toc513046477)

# ТЕКСТ ПРОГРАММЫ

## Класс LightSettings

package nick.greenwave.data.dto

import android.os.Parcel

import android.os.Parcelable

import android.support.annotation.Keep

import android.util.Log

import nick.greenwave.DEBUG

val TAG = "LightSettings"

@Keep

data class LightSetting(var greenCycle: Int = 0,

var redCycle: Int = 0,

var startOfMeasurement: Long = 0L,

var identifier: String = "1") : Parcelable {

constructor(parcel: Parcel) : this(

parcel.readInt(),

parcel.readInt(),

parcel.readLong(),

parcel.readString())

override fun describeContents(): Int {

return 0

}

override fun writeToParcel(p0: Parcel?, p1: Int) {

p0?.writeInt(greenCycle)

p0?.writeInt(redCycle)

p0?.writeLong(startOfMeasurement)

p0?.writeString(identifier)

}

companion object CREATOR : Parcelable.Creator<LightSettings> {

override fun createFromParcel(parcel: Parcel): LightSettings {

return LightSettings(parcel)

}

override fun newArray(size: Int): Array<LightSettings?> {

return arrayOfNulls(size)

}

fun parseFromString(str: String): LightSettings {

val result = LightSettings()

if (DEBUG) Log.d(TAG, "(40, LightSettings.kt) parseFromString: $str")

val tokens = str.split("-")

result.greenCycle = tokens[0].toInt()

result.redCycle = tokens[1].toInt()

result.startOfMeasurement = tokens[2].toLong()

return result

}

}

fun isSet() :Boolean {

return startOfMeasurement != 0L

}

override fun toString(): String {

return "$greenCycle-$redCycle-$startOfMeasurement"

}

}

## Класс Storage

package nick.greenwave.data

import android.content.Context

import android.content.SharedPreferences

import android.util.Log

import nick.greenwave.DEBUG

import nick.greenwave.data.dto.LightSettings

const val TAG = "Storage"

class Storage(private val context: Context) {

private val preferences: SharedPreferences

by lazy { context.getSharedPreferences("lights", Context.MODE\_PRIVATE) }

fun saveToPreferences(light: TrafficLight) {

preferences.edit().putString(createKey(light), createValue(light)).apply()

}

private fun createKey(light: TrafficLight): String {

return "${light.lat}-${light.lng}".replace('.',';')

}

private fun createValue(light: TrafficLight): String {

return "${light.settings.greenCycle}-" +

"${light.settings.redCycle}-" +

"${light.settings.startOfMeasurement}"

}

private fun parseValue(key: String, value: String): TrafficLight {

if (DEBUG) Log.d(TAG, "(29, Storage.kt) parseValue: $value key $key")

val pair = parseKey(key)

val result = TrafficLight(pair.first, pair.second)

result.settings = LightSettings.parseFromString(value)

result.settings.identifier = key

if (DEBUG) Log.d(TAG, "(29, Storage.kt) parsed: $result")

return result

}

companion object {

fun parseKey(key: String): Pair<Double, Double> {

val pair = key.split("-")

return Pair(pair[0].replace(';', '.').toDouble(),

pair[1].replace(';', '.').toDouble())

}

}

fun getAllLights() : List<TrafficLight> {

val userLights = ArrayList<TrafficLight> ()

try {

preferences.all.forEach({ userLights.add(parseValue(it.key, it.value as String)) })

} catch (e: Exception) {

Log.w(TAG, "(53, Storage.kt) getAllLights: $e")

}

return userLights

}

}

## Класс TrafficLight

package nick.greenwave.data

import android.location.Location

import nick.greenwave.data.dto.LightSettings

class TrafficLight(

val lat: Double,

val lng: Double,

var settings: LightSettings =

LightSettings("$lat-$lng".replace('.', ';'))) {

override fun toString(): String = "($lat, $lng), settings=$settings"

val location = Location("")

init {

location.latitude = lat

location.longitude = lng

}

companion object {

fun createFromSettings(settings: LightSettings): TrafficLight {

val latLng = Storage.parseKey(settings.identifier)

return TrafficLight(latLng.first, latLng.second, settings)

}

}

}

## Класс SettingsActivity

package nick.greenwave.settings

import android.app.Activity

import android.content.DialogInterface

import android.content.Intent

import android.os.Bundle

import android.support.v7.app.AlertDialog

import android.support.v7.app.AppCompatActivity

import android.util.Log

import android.view.Menu

import android.view.MenuItem

import android.widget.EditText

import android.widget.ToggleButton

import nick.greenwave.DEBUG

import nick.greenwave.R

import nick.greenwave.data.dto.LightSettings

import utils.EXTRAS\_LIGHT\_INFO

import utils.SECOND\_IN\_MILLIS

import java.util.\*

class SettingsActivity : AppCompatActivity() {

private val TAG = "SettingsActivity"

private var lightSettingsInfo: LightSettings? = null

private val greenCycle: EditText by lazy { findViewById<EditText>(R.id.input\_green\_cycle) }

private val redCycle: EditText by lazy { findViewById<EditText>(R.id.input\_red\_cycle) }

private val currentLight: ToggleButton by lazy { findViewById<ToggleButton>(R.id.current\_light) }

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.settings)

intent ?: finish()

val extras = intent.extras

if (DEBUG) Log.d(TAG, "(14, SettingsActivity.kt) onCreate extras: $extras")

extras ?: showErrorDialog()

if (!extras.containsKey(EXTRAS\_LIGHT\_INFO)) {

showErrorDialog()

return

}

lightSettingsInfo = extras.getParcelable(EXTRAS\_LIGHT\_INFO)

lightSettingsInfo ?: return

if (lightSettingsInfo!!.isSet()) {

setGreenCycle(lightSettingsInfo!!.greenCycle)

setRedCycle(lightSettingsInfo!!.redCycle)

}

}

override fun onCreateOptionsMenu(menu: Menu?): Boolean {

menuInflater.inflate(R.menu.settings\_menu, menu)

return true

}

override fun onOptionsItemSelected(item: MenuItem?): Boolean {

when (item?.itemId) {

R.id.done -> saveNewSettings()

else -> return false

}

return false

}

private fun saveNewSettings() {

if (DEBUG) Log.d(TAG, "(54, SettingsActivity.kt) saveNewSettings")

val result = Intent()

setNewSettings()

result.putExtra(EXTRAS\_LIGHT\_INFO, lightSettingsInfo)

setResult(Activity.RESULT\_OK, result)

finish()

}

private fun setNewSettings() {

lightSettingsInfo?.let {

if (currentLight.isActivated) {

// red just started, so start of measurement was 1 green cycle ago

it.startOfMeasurement = Date().time - it.greenCycle\* SECOND\_IN\_MILLIS

}

else {

it.startOfMeasurement = Date().time

}

if (greenCycle.text.isEmpty()) {

it.greenCycle = 0

} else {

it.greenCycle = greenCycle.text.toString().toInt()

}

if (redCycle.text.isEmpty()) {

it.redCycle = 0

} else {

it.redCycle = redCycle.text.toString().toInt()

}

}

}

private fun showErrorDialog() {

AlertDialog.Builder(this)

.setTitle("No lighter information provided!")

.setMessage("Settings will be closed")

.setPositiveButton("OK", { \_: DialogInterface, \_: Int -> finish() })

.show()

}

private fun setGreenCycle(v: Int) {

greenCycle.setText(v.toString())

}

private fun setRedCycle(v: Int) {

redCycle.setText(v.toString())

}

}

## Класс GreenwaveActivity

package nick.greenwave

import android.Manifest

import android.annotation.SuppressLint

import android.app.Activity

import android.content.Intent

import android.content.pm.PackageManager

import android.graphics.Color

import android.location.Location

import android.os.Bundle

import android.support.v4.app.ActivityCompat

import android.support.v4.content.ContextCompat

import android.support.v7.app.AppCompatActivity

import android.util.Log

import android.view.View

import android.widget.Button

import android.widget.LinearLayout

import android.widget.TextView

import com.google.android.gms.location.LocationCallback

import com.google.android.gms.location.LocationRequest

import com.google.android.gms.location.LocationResult

import com.google.android.gms.location.LocationServices

import com.google.android.gms.maps.CameraUpdateFactory

import com.google.android.gms.maps.GoogleMap

import com.google.android.gms.maps.MapFragment

import com.google.android.gms.maps.OnMapReadyCallback

import com.google.android.gms.maps.model.\*

import com.google.android.gms.tasks.Task

import io.reactivex.Observable

import io.reactivex.android.schedulers.AndroidSchedulers

import io.reactivex.disposables.Disposable

import io.reactivex.schedulers.Schedulers

import nick.greenwave.data.dto.LightSettings

import nick.greenwave.settings.SettingsActivity

import utils.\*

import java.util.\*

import java.util.concurrent.TimeUnit

const val DEBUG = true

private val SETTINGS\_ACTIVITY\_REQUEST\_CODE = 23

class GreenwaveActivity : AppCompatActivity(), OnMapReadyCallback, GreenwaveView {

private val presenter: GreenwavePresenterApi = GreenwavePresenter(this)

private val TAG = "GreenwaveActivity"

private var locationCallback: LocationCallback? = null

private val fusedLocationClient by lazy { LocationServices.getFusedLocationProviderClient(this) }

private val requestNearestLights by lazy { findViewById<Button>(R.id.request\_nearest) }

private val requestClosestLight by lazy { findViewById<Button>(R.id.get\_closest) }

private val speedView by lazy { findViewById<TextView>(R.id.current\_speed) }

private val recommendedSpeed by lazy { findViewById<TextView>(R.id.recommended\_speed) }

private val recommendedLabel by lazy { findViewById<TextView>(R.id.recommended\_speed\_label) }

private val recommendedUnits by lazy { findViewById<TextView>(R.id.recommended\_speed\_units) }

private val remainingDistance by lazy { findViewById<TextView>(R.id.distance\_remaining) }

private val remainingDistanceLabel by lazy { findViewById<TextView>(R.id.distance\_label) }

private val remainingDistanceUnits by lazy { findViewById<TextView>(R.id.distance\_units) }

private val lightInfoView by lazy { findViewById<LinearLayout>(R.id.light\_settings\_view) }

private val redCycle by lazy { lightInfoView.findViewById<TextView>(R.id.red\_cycle) }

private val greenCycle by lazy { lightInfoView.findViewById<TextView>(R.id.green\_cycle) }

private val current by lazy { lightInfoView.findViewById<TextView>(R.id.current) }

private val select by lazy { lightInfoView.findViewById<Button>(R.id.select\_current) }

private val settings by lazy { lightInfoView.findViewById<Button>(R.id.open\_settings) }

private var lastMarkerClicked: Marker? = null

private var followUser = true

private var stopSelectedMarkerTimer = false

private val timeToGreen by lazy { findViewById<TextView>(R.id.time) }

private val timeToGreenLabel by lazy { findViewById<TextView>(R.id.time\_label) }

private val timeToGreenUnits by lazy { findViewById<TextView>(R.id.time\_units) }

private var map: GoogleMap? = null

private var mCameraPosition: CameraPosition? = null

private val markers = ArrayList<Marker?>()

private var selectedMarkerCurrentLight: Disposable? = null

override var cameraPosition: CameraPosition?

get() = mCameraPosition

set(value) {

mCameraPosition = value

}

private val locationUpdateRequest by lazy {

LocationRequest.create()

.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY)

.setInterval(LOCATION\_UPDATE\_INTERVAL)

.setFastestInterval(SECOND\_IN\_MILLIS)

}

override fun canMoveCamera(): Boolean {

return followUser

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.greenwave)

if (DEBUG) Log.d(TAG, "onCreate: ")

requestNearestLights.setOnClickListener {

getDeviceLocation()?.addOnSuccessListener { presenter.requestNearestLights(it) }

}

requestClosestLight.setOnClickListener({ presenter.forceChooseNewClosestLight() })

val mapFragment = fragmentManager.findFragmentById(R.id.map) as MapFragment

mapFragment.getMapAsync(this)

hideSuggestions()

}

override fun onMapReady(map: GoogleMap?) {

if (DEBUG) Log.d(TAG, "(53, GreenwaveActivity.kt) onMapReady $map")

this.map = map

map?.setOnMapLongClickListener { presenter.addMapMark(it) }

map?.setOnCameraMoveStartedListener {

presenter.onCameraMoved()

}

map?.setOnMarkerClickListener { showLightInfoView(it) }

map?.setOnInfoWindowClickListener {

Log.d(TAG, "choose light")

presenter.chooseNewLight(it.position)

}

map?.setOnInfoWindowLongClickListener { presenter.openLightSettings(it) }

map?.setOnMapClickListener {

hideLightInfoView()

followUser = false // ux feature

}

presenter.onMapReady(map)

getDeviceLocation()?.addOnSuccessListener { presenter.requestNearestLights(it) }

}

override fun setDistance(distance: Double) {

runOnUiThread({remainingDistance.text = distance.toInt().toString()})

}

override fun setRecommendedSpeed(speed: Double) {

runOnUiThread({recommendedSpeed.text = String.format("%.2f", speed)})

}

override fun setActiveColorMarker(latLng: LatLng) {

runOnUiThread({

if (DEBUG) Log.d(TAG, "(125, GreenwaveActivity.kt) setActiveColorMarker: markers $markers")

if (!markers.isEmpty()) {

var current: Marker? = null

for (i in markers) {

current = i

if (current?.position?.equals(latLng)!!)

break

}

current?.setIcon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE\_GREEN))

}

})

}

override fun setTimeToGreen(time: Int) {

runOnUiThread({timeToGreen.text = time.toString()})

}

override fun resetMarkersColors() {

if (DEBUG) Log.d(TAG, "(142, GreenwaveActivity.kt) resetMarkersColors")

runOnUiThread({ markers.forEach { it?.setIcon(BitmapDescriptorFactory.defaultMarker()) } })

}

override fun setEmptyRecommendedFields() {

runOnUiThread({

recommendedSpeed.text = ""

timeToGreen.text = ""

})

}

override fun removeAllMarks() {

runOnUiThread({

markers.forEach { it?.remove() }

markers.clear()

})

}

override fun requestLocationPermissions() {

if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS\_FINE\_LOCATION)

== PackageManager.PERMISSION\_GRANTED) {

presenter.onPermissionsGranted()

} else {

if (DEBUG) Log.d(TAG, "(62, GreenwaveActivity.kt) onMapReady: request location permission")

ActivityCompat.requestPermissions(this, Array(1) { LOCATION\_PERMISSION }, LOCATION\_PERMISSION\_REQUEST)

}

}

override fun defaultCameraSettings(): CameraPosition? {

return CameraPosition.builder()

.target(STANDARD\_LOCATION)

.zoom(STANDARD\_ZOOM)

.tilt(STANDARD\_TILT)

.bearing(0f)

.build()

}

override fun moveCameraTo(position: CameraPosition) {

this.cameraPosition = position

map?.animateCamera(CameraUpdateFactory.newCameraPosition(position))

}

@SuppressLint("MissingPermission")

override fun enableMyLocationButton() {

map?.isMyLocationEnabled = true

map?.setOnMyLocationButtonClickListener {

mapToDeviceLocation()

followUser = true

true

}

}

@SuppressLint("MissingPermission")

override fun registerLocationUpdate() {

if (DEBUG) Log.d(TAG, "(89, GreenwaveActivity.kt) registerLocationUpdate ")

locationCallback ?: createLocationCallback()

val request = locationUpdateRequest

fusedLocationClient.requestLocationUpdates(request, locationCallback, null)

}

override fun unregisterLocationUpdate() {

if (DEBUG) Log.d(TAG, "(96, GreenwaveActivity.kt) unregisterLocationUpdate")

locationCallback ?: return

fusedLocationClient.removeLocationUpdates(locationCallback)

}

override fun addMark(latLng: LatLng, openSettings: Boolean) {

val markOptions = MarkerOptions()

.position(latLng)

.title("LightSettings")

.snippet("${latLng.latitude} ${latLng.longitude}")

val marker = map?.addMarker(markOptions)

markers.add(marker)

if (openSettings) {

marker?.let { presenter.openLightSettings(marker) }

}

}

override fun onReceiveSettings(light: LightSettings) {

runOnUiThread({

lightInfoView.visibility = View.VISIBLE

greenCycle.text = light.greenCycle.toString()

redCycle.text = light.redCycle.toString()

current.text = ""

lastMarkerClicked?.let { settings.setOnClickListener { presenter.openLightSettings(lastMarkerClicked!!) } }

lastMarkerClicked?.let { select.setOnClickListener { presenter.chooseNewLight(lastMarkerClicked!!.position) } }

stopSelectedMarkerTimer = false

if (light.isSet()) {

selectedMarkerCurrentLight = Observable.timer(SECOND\_IN\_MILLIS, TimeUnit.MILLISECONDS)

.observeOn(AndroidSchedulers.mainThread())

.subscribeOn(Schedulers.io())

.repeatUntil { stopSelectedMarkerTimer }

.subscribe({

val currentTime = Date().time

val diff = ((currentTime - light.startOfMeasurement) / 1000) % (light.redCycle + light.greenCycle)

if (diff < light.greenCycle) {

val greenText = "Green ${light.greenCycle - diff}"

current.text = greenText

current.setTextColor(Color.GREEN)

} else {

val redText = "Red ${light.redCycle - (diff - light.greenCycle)}"

current.text = redText

current.setTextColor(Color.RED)

}

})

}

})

}

private fun hideLightInfoView() {

stopSelectedMarkerTimer = true

selectedMarkerCurrentLight?.dispose()

lightInfoView.visibility = View.GONE

}

override fun hideSuggestions() {

remainingDistance.visibility = View.GONE

remainingDistanceLabel.visibility = View.GONE

remainingDistanceUnits.visibility = View.GONE

timeToGreen.visibility = View.GONE

timeToGreenLabel.visibility = View.GONE

timeToGreenUnits.visibility = View.GONE

recommendedSpeed.visibility = View.GONE

recommendedLabel.visibility = View.GONE

recommendedUnits.visibility = View.GONE

}

override fun showSuggestions() {

remainingDistance.visibility = View.VISIBLE

remainingDistanceLabel.visibility = View.VISIBLE

remainingDistanceUnits.visibility = View.VISIBLE

timeToGreen.visibility = View.VISIBLE

timeToGreenLabel.visibility = View.VISIBLE

timeToGreenUnits.visibility = View.VISIBLE

recommendedSpeed.visibility = View.VISIBLE

recommendedLabel.visibility = View.VISIBLE

recommendedUnits.visibility = View.VISIBLE

}

private fun showLightInfoView(marker: Marker): Boolean {

lastMarkerClicked = marker

marker.showInfoWindow()

hideLightInfoView()

presenter.requestSettingsFor(marker.position)

return true

}

override fun onResume() {

super.onResume()

presenter.onResume()

}

override fun onPause() {

super.onPause()

presenter.onPause()

}

override fun mapToDeviceLocation() {

getDeviceLocation()?.addOnSuccessListener {

if (DEBUG) Log.d(TAG, "(158, GreenwaveActivity.kt) mapToDeviceLocation: location = $it")

it?.apply {

map?.animateCamera(CameraUpdateFactory.newLatLngZoom(

LatLng(it.latitude, it.longitude),

STANDARD\_ZOOM))

}

}

}

@SuppressLint("MissingPermission")

private fun getDeviceLocation(): Task<Location>? {

return fusedLocationClient.lastLocation

}

override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {

if (requestCode == LOCATION\_PERMISSION\_REQUEST && permissions[0] == LOCATION\_PERMISSION

&& grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

presenter.onPermissionsGranted()

} else {

super.onRequestPermissionsResult(requestCode, permissions, grantResults)

}

}

private fun createLocationCallback() {

locationCallback = object : LocationCallback() {

override fun onLocationResult(loc: LocationResult?) {

if (DEBUG) Log.d(TAG, "(101, GreenwaveActivity.kt) onLocationResult: setting new location $loc")

if (DEBUG) Log.d(TAG, "locations: ${loc?.locations}")

if (DEBUG) Log.d(TAG, "lastLocation: ${loc?.lastLocation}")

loc?.let { presenter.onLocationUpdate(loc) }

}

}

}

override fun setCurrentSpeed(speed: Double) {

speedView.text = String.format("%.2f", speed)

}

override fun startSettingsActivy(lightSettingsInfo: LightSettings) {

val intent = Intent(this, SettingsActivity::class.java)

intent.putExtra(EXTRAS\_LIGHT\_INFO, lightSettingsInfo)

startActivityForResult(intent, SETTINGS\_ACTIVITY\_REQUEST\_CODE)

}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

if (requestCode == SETTINGS\_ACTIVITY\_REQUEST\_CODE && resultCode == Activity.RESULT\_OK) {

runOnUiThread({presenter.updateLightSettings(data?.getParcelableExtra(EXTRAS\_LIGHT\_INFO))})

return

}

super.onActivityResult(requestCode, resultCode, data)

}

}

## Класс GreenwaveModel

package nick.greenwave

import android.location.Location

import android.util.Log

import com.google.android.gms.maps.model.LatLng

import com.google.firebase.database.DataSnapshot

import com.google.firebase.database.DatabaseError

import com.google.firebase.database.ValueEventListener

import io.reactivex.android.schedulers.AndroidSchedulers

import io.reactivex.schedulers.Schedulers

import nick.greenwave.data.Storage

import nick.greenwave.data.TrafficLight

import nick.greenwave.data.dto.LightSettings

import utils.\*

import java.util.\*

import kotlin.concurrent.timer

class GreenwaveModel(private val presenter: GreenwavePresenterApi) : GreenwaveModelApi {

private val TAG = "GreenwaveModel"

private var nearestLights: List<TrafficLight>? = null

private val osmService by lazy { OsmService.create() }

private var lastQueryLightLocation: Location? = null

private var closestLight: TrafficLight? = null

private val userLightsStorage by lazy { Storage(presenter.getApplicationContext()) }

override fun createIdentifierFromLatlng(latLng: LatLng): String {

val result = "${latLng.latitude}-${latLng.longitude}"

return result.replace('.', ';')

}

override fun requestSettingsForLight(identifier: String, request: Int) {

val lightsRef = FirebaseDatabaseSingletone.getFirebaseInstance().getReference(LIGHTS\_REFERENCE\_FIREBASE)

if (DEBUG) Log.d(TAG, "(35, GreenwaveModel.kt) requestSettingsForLight $lightsRef identifier=$identifier")

val noSettingsTimer = timer("no\_settings", false, SECOND\_IN\_MILLIS, SECOND\_IN\_MILLIS,

{

replyToRequest(request, LightSettings(identifier))

this.cancel()

})

lightsRef.addListenerForSingleValueEvent(object : ValueEventListener {

override fun onDataChange(snapshot: DataSnapshot) {

var lightSettings = LightSettings(identifier)

if (DEBUG) Log.d(TAG, "(38, GreenwaveModel.kt) getting settings from firebase, check if key exists in $snapshot")

if (snapshot.hasChild(identifier)) {

val data = snapshot.child(identifier).getValue(LightSettings::class.java)

lightSettings = data!!

}

if (DEBUG) Log.d(TAG, "(43, GreenwaveModel.kt) firebase settings: $lightSettings ")

replyToRequest(request, lightSettings)

noSettingsTimer.cancel()

}

override fun onCancelled(p0: DatabaseError?) {

if (DEBUG) Log.d(TAG, "(44, GreenwaveModel.kt) onCancelled firebase: $p0")

}

})

}

/\*\*

\* Firebase callback

\*/

private fun replyToRequest(request: Int, settings: LightSettings) {

if (DEBUG) Log.d(TAG, "(70, GreenwaveModel.kt) replyToRequest: $settings")

when (request) {

REQUEST\_FIREBASE\_CLOSEST\_LIGHT\_SETTINGS ->

(presenter as FirebaseLightSettingsCallback).getSettings(settings)

REQUEST\_FIREBASE\_SETTINGS\_TO\_OPEN\_SETTINGS ->

(presenter as FirebaseLightSettingsCallback).openSettings(settings)

REQUEST\_FIREBASE\_JUST\_SETTINGS ->

(presenter as FirebaseLightSettingsCallback).onReceiveJustSettings(settings)

}

}

override fun updateLightSettingsInRemoteDb(light: LightSettings) {

val lightsRef = FirebaseDatabaseSingletone.getFirebaseInstance().getReference(LIGHTS\_REFERENCE\_FIREBASE)

lightsRef.child(light.identifier).setValue(light)

}

private fun createQuery(lat: Float, lng: Float): String {

val bounds = "(${lat - NEAREST\_LIGHTS\_MARGIN},${lng - NEAREST\_LIGHTS\_MARGIN}," +

"${lat + NEAREST\_LIGHTS\_MARGIN},${lng + NEAREST\_LIGHTS\_MARGIN})"

val result = StringBuilder()

OVERPASS\_QUERY.split(OVERPASS\_QUERY\_DELIMETER).joinTo(result, bounds)

if (DEBUG) Log.d(TAG, "(88, GreenwaveModel.kt) createdQuery: $result")

return result.toString()

}

var requestMade = false

override fun requestNearestLights(lat: Float, lng: Float) {

requestMade = true

requestNearestLightsFromApi(lat, lng)

}

private fun requestNearestLightsFromApi(lat: Float, lng: Float) {

osmService.fetchChunkData(createQuery(lat, lng))

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(

{ result ->

Log.d(TAG, "result = ${result.elements}")

onReceiveNearestLights(result)

},

{ error ->

Log.d(TAG, "error: $error")

}

)

}

private fun onReceiveNearestLights(result: OsmQueryResult) {

nearestLights = null

val lights = ArrayList<TrafficLight>()

for (element in result.elements) {

val identifier = "${element.lat}-${element.lon}".replace('.', ';')

val light = TrafficLight(element.lat, element.lon, LightSettings(identifier))

if (DEBUG) Log.d(TAG, "(124, GreenwaveModel.kt) create TrafficLight $light")

lights.add(light)

}

userLightsStorage.getAllLights().forEach({ lights.add(it) })

nearestLights = lights

presenter.onReceiveNearestLights(lights)

}

override fun setNewClosestLight(light: TrafficLight) {

closestLight = light

}

override fun getNearestLight(currentLocation: Location): TrafficLight? {

nearestLights ?: return null

lastQueryLightLocation = currentLocation

val latLng = LatLng(currentLocation.latitude, currentLocation.longitude)

val movementVector = getMovementVector(currentLocation)

val closestLights = nearestLights!!

.filter { isLightCloserThan(it, NEAREST\_LIGHT\_DISTANCE, currentLocation) }

.filter { isLightInFront(it, latLng, movementVector) }

.sortedBy { getDistance(latLng, LatLng(it.lat, it.lng)) }

if (DEBUG) Log.d(TAG, "(79, GreenwaveModel.kt) nearest lights: $closestLights")

if (closestLights.isNotEmpty()) {

closestLight = closestLights.first()

return closestLight

}

return null

}

private fun getMovementVector(currentLocation: Location): Pair<Double, Double> {

return Pair(

Math.cos(degreeToRadian(currentLocation.bearing.toDouble())),

Math.sin(degreeToRadian(currentLocation.bearing.toDouble())))

}

override fun getDefinedClosestFrontLight(location: Location): TrafficLight? {

closestLight ?: return null

val movementVector = getMovementVector(location)

// Check if user has passed the light. If not so, return

if (isLightInFront(closestLight!!, LatLng(location.latitude, location.longitude), movementVector))

return closestLight

return null

}

override fun detectNotableDistanceFromLastQueryLight(currentLocation: Location): Boolean {

lastQueryLightLocation ?: return !requestMade

return lastQueryLightLocation?.distanceTo(currentLocation)!! > NOTABLE\_DISTANCE

}

private fun degreeToRadian(deg: Double): Double {

return deg / 180.0 \* Math.PI

}

private fun getDistance(from: LatLng, to: LatLng): Double {

return Math.sqrt(

Math.pow(from.latitude - to.latitude, 2.0) +

Math.pow(from.longitude - to.longitude, 2.0))

}

private fun isLightInFront(light: TrafficLight, position: LatLng, movementVector: Pair<Double, Double>): Boolean {

val lightVector = Pair(light.lng - position.longitude, light.lat - position.latitude)

val cos = movementVector.scalar(lightVector)

return cos > 0

}

private fun isLightCloserThan(light: TrafficLight, distance: Double, location: Location): Boolean {

return location.distanceTo(light.location) < distance

}

}

private fun Pair<Double, Double>.scalar(vector: Pair<Double, Double>): Double {

return this.first \* vector.first + this.second \* vector.second

}

## Интерфейс GreenwaveModelApi

package nick.greenwave

import android.location.Location

import com.google.android.gms.maps.model.LatLng

import nick.greenwave.data.TrafficLight

import nick.greenwave.data.dto.LightSettings

interface GreenwaveModelApi {

fun requestNearestLights(lat: Float, lng: Float)

fun getNearestLight(currentLocation: Location): TrafficLight?

fun detectNotableDistanceFromLastQueryLight(currentLocation: Location): Boolean

fun getDefinedClosestFrontLight(location: Location): TrafficLight?

fun updateLightSettingsInRemoteDb(light: LightSettings)

fun createIdentifierFromLatlng(latLng: LatLng) : String

fun requestSettingsForLight(identifier: String, request: Int)

fun setNewClosestLight(light: TrafficLight)

}

## Класс GreenwavePresenter

package nick.greenwave

import android.content.Context

import android.location.Location

import android.util.Log

import com.google.android.gms.location.LocationResult

import com.google.android.gms.maps.GoogleMap

import com.google.android.gms.maps.model.CameraPosition

import com.google.android.gms.maps.model.LatLng

import com.google.android.gms.maps.model.Marker

import io.reactivex.Observable

import io.reactivex.android.schedulers.AndroidSchedulers

import io.reactivex.disposables.Disposable

import nick.greenwave.data.Storage

import nick.greenwave.data.TrafficLight

import nick.greenwave.data.dto.LightSettings

import utils.\*

import java.util.\*

import java.util.concurrent.TimeUnit

import kotlin.math.abs

interface FirebaseLightSettingsCallback {

fun openSettings(light: LightSettings)

fun getSettings(light: LightSettings)

fun onReceiveJustSettings(light: LightSettings)

}

class GreenwavePresenter(val view: GreenwaveView) : GreenwavePresenterApi, FirebaseLightSettingsCallback {

private var lastLoc: Location = Location("")

private var lastSpeed: Double = 0.0

private val model: GreenwaveModelApi = GreenwaveModel(this)

private val movementHelper = CameraMovementLogicHelper()

private var needUpdateNearestLight = false

private var meanSpeedHelper = MeanSpeed()

private var subscription: Disposable? = null

private val TAG = "GreenwavePresenter"

override fun onMapReady(map: GoogleMap?) {

view.requestLocationPermissions()

}

override fun updateLightSettings(light: LightSettings?) {

try {

light?.let { model.updateLightSettingsInRemoteDb(it) }

} catch (ignored: Exception) {}

}

override fun onPermissionsGranted() {

view.enableMyLocationButton()

view.defaultCameraSettings()

view.mapToDeviceLocation()

view.registerLocationUpdate()

}

override fun onPause() {

view.unregisterLocationUpdate()

}

override fun onResume() {

view.registerLocationUpdate()

}

override fun onLocationUpdate(location: LocationResult) {

lastLoc = location.lastLocation

onSpeedChanged(lastLoc.speed.toDouble())

if (view.canMoveCamera()) {

moveCameraToLastLocation()

}

updateDistanceToClosestLight()

detectTimeToUpdateClosestLight()

updateClosestLightIfNeeded()

}

private fun moveCameraToLastLocation() {

val position = CameraPosition.builder(view.cameraPosition

?: view.defaultCameraSettings())

.target(LatLng(lastLoc.latitude, lastLoc.longitude))

.bearing(lastLoc.bearing)

.build()

view.moveCameraTo(position)

}

private fun detectTimeToUpdateClosestLight() {

if (model.detectNotableDistanceFromLastQueryLight(lastLoc)) {

if (DEBUG) Log.d(TAG, "(63, GreenwavePresenter.ktt) time to update nearest lights")

requestNearestLights(lastLoc)

}

}

override fun forceChooseNewClosestLight() {

needUpdateNearestLight = true

}

private fun updateClosestLightIfNeeded() {

if (needUpdateNearestLight) {

needUpdateNearestLight = false

val closest = model.getNearestLight(lastLoc)

closest?.let { model.requestSettingsForLight(closest.settings.identifier, REQUEST\_FIREBASE\_CLOSEST\_LIGHT\_SETTINGS) }

}

}

private fun updateDistanceToClosestLight() {

val closestLight = model.getDefinedClosestFrontLight(lastLoc)

closestLight?.let { setDistanceTo(closestLight) }

}

private fun onReceiveOneClosestLight(light: TrafficLight) {

view.resetMarkersColors()

view.setActiveColorMarker(LatLng(light.lat, light.lng))

setDistanceTo(light)

subscription?.dispose()

view.setEmptyRecommendedFields()

if (!light.settings.isSet()) {

return

}

val distance = light.location.distanceTo(lastLoc)

val cycle = light.settings.greenCycle + light.settings.redCycle

val diff = (((Date().time - light.settings.startOfMeasurement) / 1000) % cycle).toInt()

var timeToGreen = findOptimalTimeToGreen(light, diff, distance, cycle)

if (DEBUG) Log.d(TAG, "(114, GreenwavePresenter.kt) light has settings, time to green $timeToGreen")

view.showSuggestions()

subscription = Observable.timer(SECOND\_IN\_MILLIS, TimeUnit.MILLISECONDS)

.observeOn(AndroidSchedulers.mainThread())

.repeat(timeToGreen.toLong())

.subscribe({

Log.v(TAG, "time to green ${timeToGreen - 1}")

view.setTimeToGreen(--timeToGreen)

view.setRecommendedSpeed(calculateRecommendedSpeed(light.location.distanceTo(lastLoc), timeToGreen))

if (timeToGreen == 1) {

view.hideSuggestions()

}

})

}

private fun findOptimalTimeToGreen(light: TrafficLight, diff: Int, distance: Float, cycle: Int): Int {

var closestStartOfGreen = light.settings.redCycle - diff // may be < 0

var minimalCorrectionOfSpeed = 10000F

var optimalMultiplier = 1

if (closestStartOfGreen > 0) {

optimalMultiplier = 0

}

var lastCorrection = minimalCorrectionOfSpeed

for (i in optimalMultiplier..100) {

val currentCorrection = abs(5 - distance / (closestStartOfGreen + i \* cycle))

if (currentCorrection > lastCorrection) {

break

}

lastCorrection = currentCorrection

if (currentCorrection < minimalCorrectionOfSpeed) {

minimalCorrectionOfSpeed = currentCorrection

optimalMultiplier = i

}

}

var timeToGreen = optimalMultiplier \* cycle + closestStartOfGreen

return timeToGreen

}

private fun setDistanceTo(light: TrafficLight) {

view.setDistance(light.location.distanceTo(lastLoc).toDouble())

}

private fun calculateRecommendedSpeed(distance: Float, timeToGreen: Int): Double {

return distance / timeToGreen.toDouble()

}

override fun onCameraMoved() {

movementHelper.startMovement()

}

override fun getApplicationContext(): Context {

val applicationContext = view.getApplicationContext()

if (DEBUG) Log.d(TAG, "(56, GreenwavePresenterr.kt) getApplicationContext: $applicationContext")

return applicationContext

}

override fun onSpeedChanged(newSpeed: Double) {

meanSpeedHelper.addSpeed(newSpeed)

lastSpeed = meanSpeedHelper.getMeanSpeed() \* 3.6

view.setCurrentSpeed(lastSpeed) // convert from m/s to km/h

}

private val userLightsStorage by lazy { Storage(view.getApplicationContext()) }

override fun addMapMark(latLng: LatLng) {

userLightsStorage.saveToPreferences(TrafficLight(latLng.latitude, latLng.longitude))

view.addMark(latLng, true)

}

override fun chooseNewLight(position: LatLng) {

model.setNewClosestLight(TrafficLight(position.latitude, position.longitude))

model.requestSettingsForLight(model.createIdentifierFromLatlng(position), REQUEST\_FIREBASE\_CLOSEST\_LIGHT\_SETTINGS)

}

override fun openLightSettings(marker: Marker) {

if (DEBUG) Log.d(TAG, "(80, GreenwavePresenterr.kt) openLightSettings for ${marker.snippet}")

val identifier = model.createIdentifierFromLatlng(marker.position)

model.requestSettingsForLight(identifier, REQUEST\_FIREBASE\_SETTINGS\_TO\_OPEN\_SETTINGS)

}

override fun openLightSettings(position: LatLng) {

val identifier = model.createIdentifierFromLatlng(position)

model.requestSettingsForLight(identifier, REQUEST\_FIREBASE\_SETTINGS\_TO\_OPEN\_SETTINGS)

}

override fun getSettings(light: LightSettings) {

if (DEBUG) Log.d(TAG, "(176, GreenwavePresenter.kt) getSettings: $light")

onReceiveOneClosestLight(TrafficLight.createFromSettings(light))

}

override fun openSettings(light: LightSettings) {

onReceiveLightSettingsOpenSettings(light)

}

override fun onReceiveLightSettingsOpenSettings(light: LightSettings) {

view.startSettingsActivy(light)

}

override fun onReceiveNearestLights(lights: List<TrafficLight>) {

view.removeAllMarks()

if (DEBUG) Log.d(TAG, "(172, GreenwavePresenter.kt) onReceiveNearestLights: ")

for (i in lights) {

view.addMark(LatLng(i.lat, i.lng), false)

}

}

override fun requestNearestLights(location: Location) {

if (DEBUG) Log.d(TAG, "(99, GreenwavePresenterr.kt) requestNearestLights for $location")

model.requestNearestLights(location.latitude.toFloat(), location.longitude.toFloat())

forceChooseNewClosestLight()

}

override fun requestSettingsFor(latLng: LatLng) {

model.requestSettingsForLight(model.createIdentifierFromLatlng(latLng), REQUEST\_FIREBASE\_JUST\_SETTINGS)

}

override fun onReceiveJustSettings(light: LightSettings) {

view.onReceiveSettings(light)

}

}

## Интерфейс GreenwavePresenterApi

package nick.greenwave

import android.content.Context

import android.location.Location

import com.google.android.gms.location.LocationResult

import com.google.android.gms.maps.GoogleMap

import com.google.android.gms.maps.model.LatLng

import com.google.android.gms.maps.model.Marker

import nick.greenwave.data.TrafficLight

import nick.greenwave.data.dto.LightSettings

interface GreenwavePresenterApi {

fun onMapReady(map: GoogleMap?)

fun onPermissionsGranted()

fun onPause()

fun onResume()

fun onLocationUpdate(location: LocationResult)

fun onSpeedChanged(newSpeed: Double)

fun getApplicationContext(): Context

fun addMapMark(latLng: LatLng)

fun openLightSettings(marker: Marker)

fun onReceiveNearestLights(lights: List<TrafficLight>)

fun requestNearestLights(location: Location)

fun onCameraMoved()

fun updateLightSettings(light: LightSettings?)

fun onReceiveLightSettingsOpenSettings(light: LightSettings)

fun forceChooseNewClosestLight()

fun chooseNewLight(position: LatLng)

fun requestSettingsFor(latLng: LatLng)

fun openLightSettings(position: LatLng)

}

## Интерфейс GreenwaveView

package nick.greenwave

import android.content.Context

import com.google.android.gms.maps.model.CameraPosition

import com.google.android.gms.maps.model.LatLng

import nick.greenwave.data.dto.LightSettings

interface GreenwaveView {

var cameraPosition: CameraPosition?

fun requestLocationPermissions()

fun enableMyLocationButton()

fun defaultCameraSettings(): CameraPosition?

fun mapToDeviceLocation()

fun registerLocationUpdate()

fun unregisterLocationUpdate()

fun moveCameraTo(position: CameraPosition)

fun getApplicationContext(): Context

fun addMark(latLng: LatLng, openSettings: Boolean)

fun startSettingsActivy(lightSettingsInfo: LightSettings)

fun setCurrentSpeed(speed: Double)

fun setActiveColorMarker(latLng: LatLng)

fun resetMarkersColors()

fun removeAllMarks()

fun setDistance(distance: Double)

fun setRecommendedSpeed(speed: Double)

fun setTimeToGreen(time: Int)

fun setEmptyRecommendedFields()

fun onReceiveSettings(light: LightSettings)

fun canMoveCamera(): Boolean

fun hideSuggestions()

fun showSuggestions()

}

## Класс CameraMovementLogicHelper

package utils

class CameraMovementLogicHelper {

val TAG = "MovementHelper"

private var isMoving = false

private var lastMove = -1L

fun startMovement() {

lastMove = System.currentTimeMillis()

}

fun canMoveCamera(): Boolean {

return (System.currentTimeMillis() - lastMove) > 500

}

}

## Класс FirebaseDatabaseSingletone

package utils

import com.google.firebase.database.FirebaseDatabase

class FirebaseDatabaseSingletone {

companion object {

@Volatile private var instance: FirebaseDatabase? = null

fun getFirebaseInstance(): FirebaseDatabase {

instance ?: synchronized(this) {

instance = FirebaseDatabase.getInstance()

instance?.setPersistenceEnabled(true)

}

return instance!!

}

}

}

## Класс GreenwaveConstants

package utils

import android.Manifest

import com.google.android.gms.maps.model.LatLng

const val SECOND\_IN\_MILLIS = 1000L

const val MINUNE\_IN\_MILLIS = SECOND\_IN\_MILLIS \* 60

const val STANDARD\_ZOOM = 16f

const val MEDIUM\_ZOOM = 10f

const val STANDARD\_TILT = 50f

val STANDARD\_LOCATION = LatLng(55.7604523, 37.5772471)

const val LOCATION\_UPDATE\_INTERVAL = 5000L

val LOCATION\_PERMISSION = Manifest.permission.ACCESS\_FINE\_LOCATION

val LOCATION\_PERMISSION\_REQUEST = 22

val SPEED\_UPDATE\_INTERVAL = 5 \* 1000L

val MAX\_LOCATIONS\_IN\_HISTORY = 3

val LAST\_2\_SPEED\_MEASURMENT = 11

val ALL\_MEAN\_SPEED\_MEASURMENT = 23

val EXTRAS\_LIGHT\_INFO = "extras\_light\_info"

val OVERPASS\_QUERY\_DELIMETER = "%bounds%"

val OVERPASS\_QUERY = "[out:json];(node[crossing=traffic\_signals]$OVERPASS\_QUERY\_DELIMETER);out body center qt 100;"

val NEAREST\_LIGHTS\_MARGIN = 0.01f // ~1.5 km

val NEAREST\_LIGHT\_DISTANCE = 800.0 // m

val MARKER\_EPSILON = 0.00003f

val NOTABLE\_DISTANCE = 100.0 //m

val LIGHTS\_REFERENCE\_FIREBASE = "lights"

val TIMER\_NAME\_GREEN = "green\_timer"

val REQUEST\_FIREBASE\_CLOSEST\_LIGHT\_SETTINGS = 123

val REQUEST\_FIREBASE\_SETTINGS\_TO\_OPEN\_SETTINGS = 124

val REQUEST\_FIREBASE\_JUST\_SETTINGS = 125

## Класс MeanSpeed

package utils

import java.util.\*

val COUNT\_OF\_MEASUREMENTS = 3

class MeanSpeed {

private var speeds = ArrayList<Double>()

fun addSpeed(speed: Double) {

if (speeds.size == COUNT\_OF\_MEASUREMENTS) {

speeds.removeAt(0)

}

speeds.add(speed)

}

fun getMeanSpeed() : Double {

return speeds.sum() / speeds.size

}

}

## Класс OsmQueryResult

package utils

import com.google.gson.annotations.SerializedName

import java.util.\*

class OsmQueryResult {

@SerializedName("elements")

var elements: List<Element> = ArrayList()

class Element {

@SerializedName("type")

var type: String? = null

@SerializedName("id")

var id: Long = 0

@SerializedName("lat")

var lat: Double = 0.toDouble()

@SerializedName("lon")

var lon: Double = 0.toDouble()

@SerializedName("tags")

var tags = Tags()

class Tags {

@SerializedName("type")

var type: String? = null

@SerializedName("amenity")

var amenity: String? = null

@SerializedName("name")

var name: String? = null

@SerializedName("phone")

var phone: String? = null

@SerializedName("contact:email")

var contactEmail: String? = null

@SerializedName("website")

var website: String? = null

@SerializedName("addr:city")

var addressCity: String? = null

@SerializedName("addr:postcode")

var addressPostCode: String? = null

@SerializedName("addr:street")

var addressStreet: String? = null

@SerializedName("addr:housenumber")

var addressHouseNumber: String? = null

@SerializedName("wheelchair")

var wheelchair: String? = null

@SerializedName("wheelchair:description")

var wheelchairDescription: String? = null

@SerializedName("opening\_hours")

var openingHours: String? = null

@SerializedName("internet\_access")

var internetAccess: String? = null

@SerializedName("fee")

var fee: String? = null

@SerializedName("operator")

var operator: String? = null

}

}

}

## Интерфейс OsmService

package utils

import retrofit2.Retrofit

import retrofit2.adapter.rxjava2.RxJava2CallAdapterFactory

import retrofit2.converter.gson.GsonConverterFactory

import retrofit2.http.GET

import retrofit2.http.Query

interface OsmService {

@GET("/cgi/interpreter")

fun fetchChunkData(@Query("data") query: String): io.reactivex.Observable<OsmQueryResult>

companion object {

fun create(): OsmService {

val retrofit = Retrofit.Builder()

.addCallAdapterFactory(RxJava2CallAdapterFactory.create())

.addConverterFactory(GsonConverterFactory.create())

.baseUrl("http://overpass.openstreetmap.ru")

.build()

return retrofit.create(OsmService::class.java)

}

}

}

# ПРИЛОЖЕНИЕ 1

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |