Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**ЛАБОРАТОРНАЯ РАБОТА**

**по дисциплине**

**«**Программирование мобильных систем**»**

|  |
| --- |
| Тема лабораторной работы |
| **Интернет вещей, связывание ble устройств и мобильного приложения. Получение данных с Гироскопа** |

Студент

|  |  |  |
| --- | --- | --- |
| Группа | ФИО | Дата |
| 3821Б1ПР3 | Неделин Д.И. |  |

Преподаватель

|  |  |  |  |
| --- | --- | --- | --- |
| Должность | ФИО | Подпись | Дата |
| Старший преподаватель | Карчков Д.А. |  |  |

Нижний Новгород – 2024 г.

**Цель работы**

Реализовать передачу, соединение esp32 с MPU-6050 и передача данных с гироскопа на мобильное устройство

**Ход работы**

**Мобильное приложение**

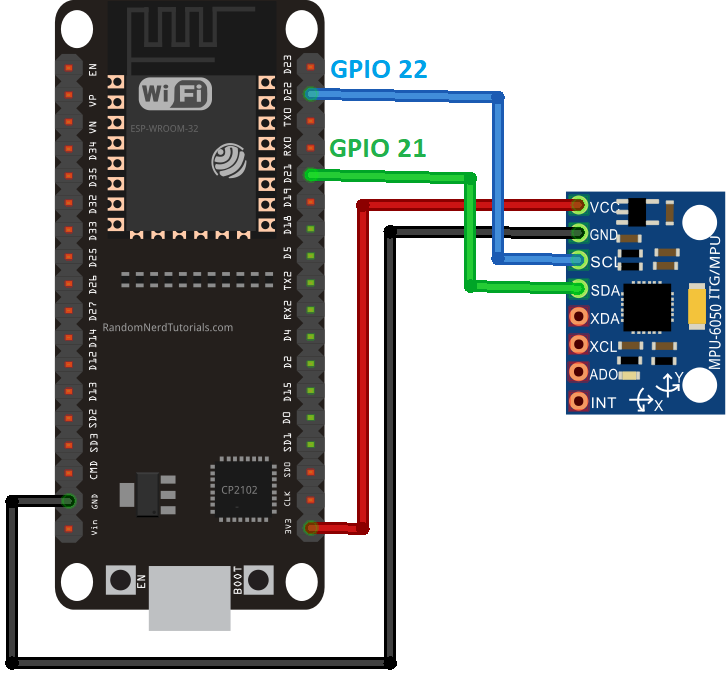
Было реализовано приложение на kotlin выполняющее следующие функции:

Хранение пользовательских UUID для быстрого переподключения к ранним устройствам, реализовано в **UserStore**

Поиск, подключение, отключение bluetooth low energy устройств реализован в классе **BluetoothViewModel**

**TestBluetoothScreen** и **MainActivity** содержат код отвечающий за визуальную составляющую  
Код указан в приложениях 1, 2, 3, 4

**Esp32 и гироскоп MPU-6050**



Схематичная диаграмма подключения esp32 c MPU-6050

MPU-6050 измеряет угловую скорость вращения вокруг оси и ускорение вдоль оси.

Esp32 питает MPU-6050, реализует подключение при помощи ble и передачу данных на мобильное устройство.

Код для Esp32 и MPU-6050 указан в приложении 5, 6, 7

**Результат лабораторной работы:**

Я реализовал мобильное приложение, способное выполнять поиск ble устройств, подключаться к ним и считывать данные с них.

Репозиторий с исходным кодом:

https://github.com/STELLSAN/Gyroscope\_IoT\_lab.git

# Приложения

## Приложение №1 UserStore

import kotlinx.coroutines.flow.first  
  
import android.content.Context  
import androidx.datastore.core.DataStore  
import androidx.datastore.preferences.core.Preferences  
import androidx.datastore.preferences.core.edit  
import androidx.datastore.preferences.core.stringPreferencesKey  
import androidx.datastore.preferences.preferencesDataStore  
  
// Хранение данных через dataStore  
class UserStore(private val context: Context) {  
 // preferences | userToken ; key-value  
 companion object {  
 private val Context.*dataStore*: DataStore<Preferences> by *preferencesDataStore*("userToken")  
 }  
// ассинхронн.  
 suspend fun getAccessToken(key: String): String {  
 return context.*dataStore*.data.first()[*stringPreferencesKey*(key)] ?: ""  
 }  
// ассинхронн.  
 suspend fun saveToken(key: String,token: String) {  
 context.*dataStore*.edit **{** preferences **->** preferences[*stringPreferencesKey*(key)] = token  
 **}** }  
}

## Приложение №2 MainActivity.kt

package com.example.bledproject  
  
import android.Manifest  
import android.app.Activity  
import android.bluetooth.BluetoothDevice  
import android.bluetooth.BluetoothManager  
import android.content.Intent  
import android.content.pm.PackageManager  
import android.os.Build  
import android.os.Bundle  
import android.widget.Toast  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.core.app.ActivityCompat  
import com.example.bledproject.bluetooth.BluetoothViewModel  
import com.example.bledproject.bluetooth.TestBluetoothScreen  
import com.example.bledproject.data.UserStore  
import com.example.bledproject.ui.theme.BLEDProjectTheme  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 // userStore for storing device address  
 val userStore = UserStore(*applicationContext*)  
  
 // setup bluetooth view model  
 val bluetoothManager = getSystemService(*BLUETOOTH\_SERVICE*) as BluetoothManager  
 val bluetoothAdapter = bluetoothManager.*adapter* val bluetoothViewModel = BluetoothViewModel(  
 *application*,  
 bluetoothAdapter,  
 userStore,  
 ::messageHandler  
 )  
  
 // auto connect to bluetooth device on start if it was connected before  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** if (userStore.getAccessToken(getString(R.string.*bluetoothDeviceAddress*)) != "") {  
 *println*("Access token bluetooth" + userStore.getAccessToken(getString(R.string.*bluetoothDeviceAddress*)))  
 // create BluetoothDevice from address  
  
 val bluetoothDevice: BluetoothDevice? =  
 bluetoothAdapter.getRemoteDevice(userStore.getAccessToken(getString(R.string.*bluetoothDeviceAddress*)))  
 if (bluetoothDevice != null) {  
 bluetoothViewModel.connectToDevice(bluetoothDevice)  
 } else {  
 *println*("Bluetooth device is null")  
 }  
 } else {  
 *println*("Access token bluetooth is empty" + userStore.getAccessToken(getString(R.string.*bluetoothDeviceAddress*)))  
 }  
 **}** *setContent* **{** BLEDProjectTheme **{** // чек, предоставлены ли разрешения Bluetooth  
 if (ActivityCompat.checkSelfPermission(  
 bluetoothViewModel.context,  
 Manifest.permission.*BLUETOOTH\_CONNECT* ) != PackageManager.*PERMISSION\_GRANTED* || ActivityCompat.checkSelfPermission(  
 bluetoothViewModel.context,  
 Manifest.permission.*BLUETOOTH\_SCAN* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 RequestPermission()  
 } else {  
 TestBluetoothScreen(bluetoothViewModel)  
 }  
  
  
 **}  
 }** }  
  
 @Composable  
 private fun RequestPermission() {  
 if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*S*) {  
 // request permissions диалог  
 ActivityCompat.requestPermissions(  
 this as Activity,  
 *arrayOf*(  
 Manifest.permission.*BLUETOOTH\_CONNECT*,  
 Manifest.permission.*BLUETOOTH\_SCAN* ),  
 1  
 )  
 } else {  
 Toast.makeText(  
 this,  
 "newer android version required",  
 Toast.*LENGTH\_SHORT* )  
 .show()  
 Text(text = "newer android version required")  
 }  
 }  
  
 private fun messageHandler(message: String) {  
 // MESSAGE HANDLER Заглушка  
 *println*("Message handler: " + message)  
 }  
 // обрабатываем результаты permission запросов  
 @Deprecated("Deprecated in Java")  
 override fun onRequestPermissionsResult(  
 requestCode: Int,  
 permissions: Array<String>,  
 grantResults: IntArray  
 ) {  
 super.onRequestPermissionsResult(  
 requestCode,  
 permissions,  
 grantResults  
 )  
 when (requestCode) {  
 1 -> {  
 if (grantResults.*isNotEmpty*() && grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
 // Разрешение получено  
 // перезапуск activity  
 val intent = Intent(  
 this,  
 MainActivity::class.*java* )  
 intent.addFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP* or Intent.*FLAG\_ACTIVITY\_NEW\_TASK*)  
 startActivity(intent)  
 finish()  
 } else {  
 // Разрешения отклонены  
 Toast.makeText(  
 this,  
 "Bluetooth Permission required",  
 Toast.*LENGTH\_SHORT* )  
 .show()  
 }  
 return  
 }  
 }  
 }  
}

## Приложение №3 BluetoothViewModel.kt

package com.example.bledproject.bluetooth  
  
  
import android.Manifest  
import android.bluetooth.BluetoothAdapter  
import android.bluetooth.BluetoothDevice  
import android.bluetooth.BluetoothGatt  
import android.bluetooth.BluetoothGattCallback  
import android.bluetooth.BluetoothGattCharacteristic  
import android.bluetooth.BluetoothGattDescriptor  
import android.bluetooth.BluetoothProfile  
import android.bluetooth.le.ScanCallback  
import android.bluetooth.le.ScanFilter  
import android.bluetooth.le.ScanResult  
import android.bluetooth.le.ScanSettings  
import android.content.Context  
import android.content.pm.PackageManager  
import android.util.Log  
import androidx.compose.runtime.mutableStateListOf  
import androidx.compose.runtime.mutableStateOf  
import androidx.core.app.ActivityCompat  
import com.example.bledproject.R  
import com.example.bledproject.data.UserStore  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import java.util.UUID  
  
  
class BluetoothViewModel(  
 myContext: Context,  
 bluetoothAdapter: BluetoothAdapter,  
 val userStore: UserStore,  
 val messageHandler: (String) -> Unit  
) {  
 // найденные устройства  
 val devices = *mutableStateListOf*<BluetoothDevice>()  
 // bool check  
 val scanning = *mutableStateOf*(false)  
 val context = myContext  
 // ручной дисконнект check  
 private val manuallyDisconnected = *mutableStateOf*(false)  
  
 // состояние connection, адрес  
 var connected = *mutableStateOf*(false)  
 var connectedDevice = *mutableStateOf*("")  
  
 // Заглушка *TODO: remove receivedData and use messageHandler instead* var receivedData = *mutableStateOf*("")  
  
 // thisGatt obj  
 var thisGatt: BluetoothGatt? = null  
 var writeCharacteristic: BluetoothGattCharacteristic? = null  
  
 // Интерфейс для скана & device add  
 private val bluetoothLeScanner = bluetoothAdapter.*bluetoothLeScanner* private val scanCallback = object : ScanCallback() {  
 override fun onScanResult(  
 callbackType: Int,  
 result: ScanResult  
 ) {  
 val device = result.*device* // only add if not already in list  
 if (!devices.contains(device)) devices.add(device)  
 }  
 }  
 //  
 val gattObject = object : BluetoothGattCallback() {  
 override fun onConnectionStateChange(  
 gatt: BluetoothGatt,  
 status: Int,  
 newState: Int  
 ) {  
 super.onConnectionStateChange(  
 gatt,  
 status,  
 newState  
 )  
 if (newState == BluetoothProfile.*STATE\_CONNECTED*) {  
 Log.d(  
 "GattCallback",  
 "Successfully connected to device"  
 )  
 connected.value = true  
 manuallyDisconnected.value = false  
 saveDevice(gatt.*device*)  
 if (ActivityCompat.checkSelfPermission(  
 context,  
 Manifest.permission.*BLUETOOTH\_CONNECT* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 // *TODO: Consider calling* // *ActivityCompat#requestPermissions* // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.  
 return  
 }  
 gatt.discoverServices()  
 } else if (newState == BluetoothProfile.*STATE\_DISCONNECTED*) {  
 Log.d(  
 "GattCallback",  
 "Successfully disconnected from device"  
 )  
 connected.value = false  
 connectedDevice.value = ""  
 if (!manuallyDisconnected.value) {  
 // reconnect  
 connectToDevice(gatt.*device*)  
 }  
 }  
 }  
 // Поиск устройств  
 override fun onServicesDiscovered(  
 gatt: BluetoothGatt?,  
 status: Int  
 ) {  
 super.onServicesDiscovered(  
 gatt,  
 status  
 )  
 if (status == BluetoothGatt.*GATT\_SUCCESS*) {  
 Log.d(  
 "GattCallback",  
 "Discovered Services"  
 )  
 thisGatt = gatt  
 gatt?.*services*?.*forEach* **{** service **->** Log.d(  
 "GattCallback",  
 "Service: ${service.*uuid*}"  
 )  
 service.*characteristics*.*forEach* **{** characteristic **->** Log.d(  
 "GattCallback",  
 "Characteristic: ${characteristic.*uuid*}"  
 )  
 if (characteristic.*uuid* == UUID.fromString("6E400003-B5A3-F393-E0A9-E50E24DCCA9E")) {  
 Log.d(  
 "GattCallback",  
 "Found read characteristic"  
 )  
 // Listen for changes on this characteristic  
 if (ActivityCompat.checkSelfPermission(  
 context,  
 Manifest.permission.*BLUETOOTH\_CONNECT* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 // *TODO: Consider calling* // *ActivityCompat#requestPermissions* // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.  
 return  
 }  
 gatt.setCharacteristicNotification(  
 characteristic,  
 true  
 )  
 // this is essential to be able to read the characteristic  
 val descriptor = characteristic.getDescriptor(  
 UUID.fromString("00002902-0000-1000-8000-00805f9b34fb")  
 )  
 descriptor.setValue(  
 BluetoothGattDescriptor.*ENABLE\_NOTIFICATION\_VALUE* )  
 gatt.writeDescriptor(descriptor)  
 }  
 if (characteristic.*uuid* == UUID.fromString("6E400002-B5A3-F393-E0A9-E50E24DCCA9E")) {  
 Log.d(  
 "GattCallback",  
 "Found write characteristic"  
 )  
 writeCharacteristic = characteristic  
 }  
 characteristic.*descriptors*.*forEach* **{** descriptor **->** Log.d(  
 "GattCallback",  
 "Descriptor: ${descriptor.*uuid*}"  
 )  
 **}  
 }  
 }** } else {  
 Log.d(  
 "GattCallback",  
 "Failed to discover services"  
 )  
 }  
 }  
  
 override fun onCharacteristicChanged(  
 gatt: BluetoothGatt,  
 characteristic: BluetoothGattCharacteristic,  
 value: ByteArray  
 ) {  
 super.onCharacteristicChanged(  
 gatt,  
 characteristic,  
 value  
 )  
 Log.d(  
 "GattCallback",  
 "Characteristic changed"  
 )  
 if (characteristic.*uuid* == UUID.fromString("6E400003-B5A3-F393-E0A9-E50E24DCCA9E")) {  
 // Characteristic changed  
 val readValue = *String*(characteristic.*value*)  
 Log.d(  
 "BluetoothScreen",  
 "Value: $readValue"  
 )  
 receivedData.value = readValue  
 messageHandler(readValue)  
 }  
 }  
 }  
  
 fun startScan() {  
 scanning.value = true  
 val scanSettings = ScanSettings.Builder()  
 .setScanMode(ScanSettings.*SCAN\_MODE\_LOW\_LATENCY*)  
 .build()  
 val scanFilter = ScanFilter.Builder()  
 .build()  
 val scanFilters = *listOf*(scanFilter)  
  
 if (ActivityCompat.checkSelfPermission(  
 context,  
 Manifest.permission.*BLUETOOTH\_SCAN* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 // *TODO: Consider calling* // *ActivityCompat#requestPermissions* // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.  
 return  
 }  
 bluetoothLeScanner.startScan(  
 scanFilters,  
 scanSettings,  
 scanCallback  
 )  
  
 }  
  
 fun writeCharacteristic(message: String) {  
 val characteristic = writeCharacteristic ?: return  
 characteristic.setValue(message.*toByteArray*(Charsets.UTF\_8))  
 if (ActivityCompat.checkSelfPermission(  
 context,  
 Manifest.permission.*BLUETOOTH\_CONNECT* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 // *TODO: Consider calling* // *ActivityCompat#requestPermissions* // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.  
 return  
 }  
 thisGatt?.writeCharacteristic(characteristic)  
 }  
  
 fun stopScan() {  
 scanning.value = false  
 if (ActivityCompat.checkSelfPermission(  
 context,  
 Manifest.permission.*BLUETOOTH\_SCAN* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 // *TODO: Consider calling* // *ActivityCompat#requestPermissions* // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.  
 return  
 }  
 bluetoothLeScanner.stopScan(scanCallback)  
 }  
  
 fun connectToDevice(device: BluetoothDevice) {  
 if (ActivityCompat.checkSelfPermission(  
 context,  
 Manifest.permission.*BLUETOOTH\_CONNECT* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 // *TODO: Consider calling* // *ActivityCompat#requestPermissions* // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.  
 return  
 }  
  
  
 device.connectGatt(  
 context,  
 false,  
 gattObject  
 )  
 connectedDevice.value = device.*address* }  
  
 fun disconnect() {  
 // отключение от устройства  
 if (ActivityCompat.checkSelfPermission(  
 context,  
 Manifest.permission.*BLUETOOTH\_CONNECT* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 // *TODO: Consider calling* // *ActivityCompat#requestPermissions* // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.  
 return  
 }  
 thisGatt?.disconnect()  
 manuallyDisconnected.value = true  
 }  
  
 fun saveDevice(device: BluetoothDevice) {  
 // сохраняем device address  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** device.*address*?.*let* **{** userStore.saveToken(  
 context.getString(R.string.*bluetoothDeviceAddress*),  
 **it** )  
 **}** Log.d(  
 "GattCallback",  
 "Saved bluetooth device address: " + userStore.getAccessToken(context.getString(R.string.*bluetoothDeviceAddress*))  
 )  
 **}** }  
  
  
}

## Приложение №4 TestBluetoothScreen.kt

package com.example.bledproject.bluetooth  
  
import android.Manifest  
import android.content.pm.PackageManager  
import android.widget.Toast  
import androidx.compose.animation.core.\*  
import androidx.compose.foundation.Canvas  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.lazy.LazyColumn  
import androidx.compose.material3.Button  
import androidx.compose.material3.Text  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.geometry.CornerRadius  
import androidx.compose.ui.geometry.Offset  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.graphics.graphicsLayer  
import androidx.compose.ui.unit.dp  
import androidx.core.app.ActivityCompat  
import kotlinx.coroutines.delay  
import kotlin.math.roundToInt  
import kotlin.random.Random  
  
@Composable  
fun TestBluetoothScreen(bluetoothViewModel: BluetoothViewModel) {  
  
 // уведомление при изменении connected  
 LaunchedEffect(  
 key1 = bluetoothViewModel.connected.value,  
 block = **{** if (bluetoothViewModel.connected.value) {  
 Toast.makeText(  
 bluetoothViewModel.context,  
 "Connected to ${bluetoothViewModel.connectedDevice.value}",  
 Toast.*LENGTH\_SHORT* ).show()  
 } else {  
 Toast.makeText(  
 bluetoothViewModel.context,  
 "Disconnected",  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}** )  
  
 Column **{** Row **{** Button(  
 modifier = Modifier,  
 onClick = **{** if (bluetoothViewModel.scanning.value) {  
 bluetoothViewModel.stopScan()  
 } else {  
 bluetoothViewModel.startScan()  
 }  
 **}**) **{** if (bluetoothViewModel.scanning.value) {  
 Text("Stop Scan")  
 } else {  
 Text("Start Scan")  
 }  
 **}** if (bluetoothViewModel.connected.value) {  
 Button(  
 modifier = Modifier.*fillMaxWidth*(),  
 onClick = **{** bluetoothViewModel.disconnect()  
 **}**) **{** Text("Disconnect")  
 **}** }  
 **}** LazyColumn(modifier = Modifier.*weight*(1f)) **{** bluetoothViewModel.devices.*forEach* **{** device **->** item **{** Row **{** if (ActivityCompat.checkSelfPermission(  
 bluetoothViewModel.context,  
 Manifest.permission.*BLUETOOTH\_CONNECT* ) != PackageManager.*PERMISSION\_GRANTED* ) {  
 return@item  
 }  
 Text(device.*name* ?: "Unnamed device")  
 Text(  
 modifier = Modifier.*weight*(1f),  
 text = device.*address* )  
 Button(onClick = **{** bluetoothViewModel.connectToDevice(device)  
 **}**) **{** Text("Connect")  
 **}  
 }  
 }  
 }  
 }** if (bluetoothViewModel.connected.value) {  
 Row **{** Text("Connected to: ")  
 Text(text = bluetoothViewModel.connectedDevice.value)  
 **}** Row **{** Text("Read Characteristic: ")  
 Text(text = bluetoothViewModel.receivedData.value)  
 **}** RotatingSquareScreen(bluetoothViewModel)  
 }  
 **}**}  
  
@Composable  
fun RotatingSquareScreen(bluetoothViewModel: BluetoothViewModel) {  
 var rotationX by remember **{** *mutableStateOf*(0f) **}** var rotationY by remember **{** *mutableStateOf*(0f) **}** LaunchedEffect(Unit) **{** while (true) {  
 var data = bluetoothViewModel.receivedData.value;  
 var (x, y) = *parseRotationData*(data)  
  
 rotationX = x  
 rotationY = y  
  
 delay(50)  
 }  
 **}** RotatingColoredSquare(rotationXValue = rotationX, rotationYValue = rotationY)  
}  
  
@Composable  
fun RotatingColoredSquare(rotationXValue: Float, rotationYValue: Float) {  
 var rotationX by remember **{** *mutableStateOf*(rotationXValue) **}** var rotationY by remember **{** *mutableStateOf*(rotationYValue) **}** LaunchedEffect(rotationXValue, rotationYValue) **{** rotationX = rotationXValue  
 rotationY = rotationYValue  
 **}** // 4 примыкающих квадрата  
 Canvas(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(600.*dp*)  
 .*graphicsLayer*(  
 rotationX = rotationX,  
 rotationY = rotationY  
 )  
 ) **{** val squareSize = 100f  
  
 val centerOffset = *Offset*(center.x - squareSize, center.y - squareSize)  
  
 drawRect(  
 color = Color.Red,  
 topLeft = centerOffset,  
 size = androidx.compose.ui.geometry.*Size*(squareSize, squareSize)  
 )  
  
 drawRect(  
 color = Color.Green,  
 topLeft = *Offset*(centerOffset.x + squareSize, centerOffset.y),  
 size = androidx.compose.ui.geometry.*Size*(squareSize, squareSize)  
 )  
  
 drawRect(  
 color = Color.Blue,  
 topLeft = *Offset*(centerOffset.x, centerOffset.y + squareSize),  
 size = androidx.compose.ui.geometry.*Size*(squareSize, squareSize)  
 )  
  
 drawRect(  
 color = Color.Yellow,  
 topLeft = *Offset*(centerOffset.x + squareSize, centerOffset.y + squareSize),  
 size = androidx.compose.ui.geometry.*Size*(squareSize, squareSize)  
 )  
 **}**}  
  
fun parseRotationData(data: String): Pair<Float, Float> {  
 return try {  
 val parts = data.*split*(" | ")  
 if (parts.size == 2) {  
 val rotationX = parts[0].*toFloatOrNull*()?.*coerceIn*(-180f, 180f) ?: 0f  
 val rotationY = parts[1].*toFloatOrNull*()?.*coerceIn*(-180f, 180f) ?: 0f  
  
 //val roundedRotationX = String.format("%.1f", rotationX).toFloat()  
 //val roundedRotationY = String.format("%.1f", rotationY).toFloat()  
  
 rotationX *to* rotationY  
 } else {  
 0f *to* 0f  
 }  
 } catch (e: Exception) {  
 0f *to* 0f  
 }  
}  
  
fun generateRandomData(): String {  
 val rotationX = Random.nextFloat() \* 360 - 180  
 val rotationY = Random.nextFloat() \* 360 - 180  
 return "$rotationX | $rotationY"  
}  
  
  
@Composable  
fun RotatingSquare(rotationXValue: Float, rotationYValue: Float) {  
 var rotationX by remember **{** *mutableStateOf*(rotationXValue) **}** var rotationY by remember **{** *mutableStateOf*(rotationYValue) **}** LaunchedEffect(rotationXValue, rotationYValue) **{** rotationX = rotationXValue  
 rotationY = rotationYValue  
 **}** Canvas(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(800.*dp*)  
 .*graphicsLayer*(  
 rotationX = rotationX,  
 rotationY = rotationY  
 )  
 ) **{** drawRoundRect(  
 color = Color.Blue,  
 topLeft = *Offset*(center.x - 50f, center.y - 50f),  
 size = androidx.compose.ui.geometry.*Size*(200f, 200f),  
 cornerRadius = *CornerRadius*(10f)  
 )  
 **}**}

## Приложение №5 bluetooth.h

#include <Arduino.h>

boolean setupBluetooth();

void loopBluetooth();

void sendMessage(String message);

void messageHandler(String message);

## Приложение №6 bluetooth.cpp

#include <Arduino.h>

#include <BLEDevice.h>

#include <BLEServer.h>

#include <BLEUtils.h>

#include <BLE2902.h>

#include "bluetooth.h"

#include <Wire.h>

#include <MPU6050\_light.h>

MPU6050 mpu(Wire);

unsigned long timer = 0;

BLEServer \*pServer = NULL;

BLECharacteristic \*pTxCharacteristic;

bool deviceConnected = false;

bool oldDeviceConnected = false;

uint8\_t txValue = 0;

// See the following for generating UUIDs:

// https://www.uuidgenerator.net/

#define SERVICE\_UUID "6E400001-B5A3-F393-E0A9-E50E24DCCA9E" // UART service UUID

#define CHARACTERISTIC\_UUID\_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"

#define CHARACTERISTIC\_UUID\_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

class MyServerCallbacks : public BLEServerCallbacks

{

    void onConnect(BLEServer \*pServer)

    {

        deviceConnected = true;

    };

    void onDisconnect(BLEServer \*pServer)

    {

        deviceConnected = false;

    }

};

class MyCallbacks : public BLECharacteristicCallbacks

{

    void onWrite(BLECharacteristic \*pCharacteristic)

    {

        std::string rxValue = pCharacteristic->getValue();

        char myArray[rxValue.size() + 1]; // as 1 char space for null is also required

        strcpy(myArray, rxValue.c\_str());

        if (rxValue.length() > 0)

        {

            messageHandler(rxValue.c\_str());

        }

    }

};

boolean setupBluetooth()

{

    Wire.begin();

    byte status = mpu.begin();

    Serial.println(F("Calculating offsets, do not move MPU6050"));

    delay(500);

    mpu.calcOffsets();

    Serial.println("Done!\n");

    // Create the BLE Device

    BLEDevice::init("UART Service");

    // Create the BLE Server

    pServer = BLEDevice::createServer();

    pServer->setCallbacks(new MyServerCallbacks());

    // Create the BLE Service

    BLEService \*pService = pServer->createService(SERVICE\_UUID);

    // Create a BLE Characteristic

    pTxCharacteristic = pService->createCharacteristic(

        CHARACTERISTIC\_UUID\_TX,

        BLECharacteristic::PROPERTY\_NOTIFY);

    pTxCharacteristic->addDescriptor(new BLE2902());

    BLECharacteristic \*pRxCharacteristic = pService->createCharacteristic(

        CHARACTERISTIC\_UUID\_RX,

        BLECharacteristic::PROPERTY\_WRITE);

    pRxCharacteristic->setCallbacks(new MyCallbacks());

    // Start the service

    pService->start();

    // Start advertising

    pServer->getAdvertising()->start();

    Serial.println("Waiting a client connection to notify...");

    return true;

}

void loopBluetooth()

{

    if (deviceConnected)

    {

        mpu.update();

        if((millis()-timer)>50){

            //Serial.print(mpu.getAngleX());

        // case: device connected

        //mpu.getAccAngleX()

            std::string value = std::to\_string(mpu.getAccAngleX());

            value += " | " + std::to\_string(mpu.getAccAngleY());

            Serial.println(value.c\_str());

            pTxCharacteristic->setValue(value.c\_str());

            pTxCharacteristic->notify();

            //txValue++;

        }

    }

    // disconnecting

    if (!deviceConnected && oldDeviceConnected)

    {

        // case: device disconnected

        delay(500);                  // give the bluetooth stack the chance to get things ready

        pServer->startAdvertising(); // restart advertising

        Serial.println("start advertising");

        oldDeviceConnected = deviceConnected;

    }

    // connecting

    if (deviceConnected && !oldDeviceConnected)

    {

        // do stuff here on first connect

        oldDeviceConnected = deviceConnected;

    }

    delay(1000);

}

void sendMessage(String message)

{

    pTxCharacteristic->setValue(message.c\_str());

    pTxCharacteristic->notify();

}

void messageHandler(String message)

{

    // ADD YOUR CODE HERE

    Serial.println(message);

}

## Приложение №7 main.cpp

#include <Arduino.h>

#include "connection/bluetooth.h"

void setup()

{

  Serial.begin(115200);

  setupBluetooth();

}

void loop()

{

  loopBluetooth();

}