ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ЗА ДИГИТАЛНИ НАУКИ „СОФТУНИ БУДИТЕЛ“, гр. София

**ДИПЛОМЕН ПРОЕКТ**

на Алекс-Валентино Георгиев Георгиев

ученик/ученичка от XII А клас

професия- код: **481030**, **“Приложен програмист”**

специалност- код: **4810301**, **“Приложно програмиране”**

**Тема: Домашна Автоматизация (Smart Home)**

**Ръководител-консултант: Венцислав Ангелов**

Сесия: май-юни 2024г.

Дата:.........................

1. **Увод**
   1. **Мотивация**

Като студент в 12-ти клас, работещ върху дипломен проект за иновативно приложение за управление на RGB осветлението чрез мобилно приложение, моята мотивация играе жизненоважна роля в гарантирането на неговия успех. Като човек, който винаги е бил привлечен от технологиите и автоматизацията, намирам интелектуалното предизвикателство на този проект за изключително стимулиращо. Освен това съм вътрешно мотивиран да допринеса за подобряването на енергийната ефективност и удобството на дома чрез създаването на тази иновативна система за интелигентно осветление.

* 1. **Цели на проекта**

При разработването на иновативното приложение за управление на RGB осветлението чрез мобилно устройство, основните ми цели за проекта са: **а)** Създаването на удобен за потребителя, ефективен и визуално привлекателен интерфейс, който позволява безпроблемно управление на различни системи за осветление.

**б)** Осигуряване на евтин и лесен за инсталация физически модул за оправление на осветителни тела

Тези цели са предназначени да осигурят висококачествен продукт, който отговаря на нуждите на потребителите и демонстрира моите умения като софтуерен инженер и дизайнер.

* 1. **Очаквани резултати**

Очакваните резултати от този проект са както следва:

1. Потребителски интерфейс (UI): Приложението ще разполага с интуитивен и естетически приятен UI, който позволява на потребителите лесно да навигират и контролират своите RGB осветителни системи. Ще бъдат включени различни функции, като персонализирани цветови палитри, предварително зададени режими и възможности за дистанционно управление.
2. Ефективност: Приложението ще работи гладко и надеждно на различни устройства и операционни системи, осигурявайки оптимално потребителско изживяване. Това ще бъде постигнато чрез щателни тестове за съвместимост, отстраняване на грешки и внедряване на функции за самодиагностика, които позволяват ранното откриване и разрешаване на проблеми.
3. Енергийна ефективност: Приложението ще бъде проектирано да бъде енергийно ефективно, като свежда до минимум консумацията на ресурси и удължава живота на свързаните RGB осветителни системи. Това ще бъде постигнато чрез внедряване на усъвършенствани алгоритми за управление на мощността и осигуряване на опции за потребителите да персонализират нивата на яркост и честотата на опресняване на осветлението.
4. Документация: Ще бъде изготвена изчерпателна техническа документация, описваща дизайна, архитектурата и функционалността на приложението. Това ще включва диаграми на потока от данни, UML диаграми и подробни описания на кода, което ще улесни бъдещите разработки и поддръжка на приложението.
5. Изпълнение: Приложението ще демонстрира способността ми да проектирам и разработвам софтуерни решения, които отговарят на специфични изисквания и предоставят стойност на потребителите. Това ще бъде важен крайъгълен камък в моето академично и професионално развитие като софтуерен инженер и дизайнер.
6. **Изложение**
   1. **Описание на хардуера и софтуера**
      1. **Хадруер**

Хардуерът на този дипломен проект се състои от микроконтролер ESP32, който служи като централна точка за комуникация между мобилното приложение и системата за RGB осветление. ESP32 е оборудван с Wi-Fi и Bluetooth възможности, което позволява безжична комуникация между устройствата. Релето за включване и изключване на осветлението е свързано към ESP32, заедно с необходимите транзистори и резистори за правилно функциониране. Други малки компоненти, като бутони, индикатори и защитни капачки, допълват хардуерната конфигурация.

* + 1. **Софтуер**

Софтуерът на проекта е разработен с помощта на Java в Android Studio за мобилното приложение. Приложението комуникира с ESP32 чрез Wi-Fi връзка, използвайки протокол за команден обмен. ESP32, от друга страна, изпълнява собствен софтуер, базиран на C++, който обработва входящите команди от приложението, контролира RGB осветлението и изпраща обратна връзка към приложението, ако е необходимо.

Комуникацията между мобилното приложение и ESP32 е криптирана за сигурност, като се използват индустриални стандарти за защита на предаваните данни. Софтуерът също така включва функции за самодиагностика, които позволяват ранното откриване и разрешаване на проблеми, като гарантират, че системата работи гладко и надеждно.

Като цяло, комбинацията от ESP32 микроконтролер и персонализиран софтуер както за мобилни устройства, така и за микроконтролери позволява безпроблемна комуникация и контрол на RGB осветителните системи, осигурявайки удобство, персонализиране и енергийна ефективност за потребителите.

* 1. **Процес на сглобяване на устройството**
     1. **Захранване**

Модулът може да се захранва от 10V-12V 0.5A-1A DC захранващ блок. RGB лентата е свързана директно към захранването, докато във ел. веригата на микроконтролера има преобразовател на напрежение, чието изходно напрежение е 3.3V - точно толкова, колкото са нужни за да се захрани ESP32 през пиновете.

* + 1. **Микроконтролер**

Микроконтролерът (ESP32) във всеки модул отговаря за остановяване на връзка с мобилно приложение, както и за приемане на команди и изпращане на информация за състоянието на периферният нахдруер на модула.

* + 1. **Реле (10V)**

Изпълнява простата задача - влкючване/изключване на светлините. Получава входен сигнал от потребителя през микроконтролера и според него затваря или отваря ел. веригата на RGB лентата свързана към модула.

* + 1. **Транзистори (NPN)**

Сигналът за всеки цвят на RGB лентата подаден от микроконтролера, минава през транзистор, чиято роля е да регулира потока на електричество към еликтрическата веригата на съответния цвят, спрямо сигналът от контролера.

* + 1. **RGB лента**

Проста, 12V RGB лента, която приема 3 входящи сигнала (по един за всеки от основните цветове - червено, зелено, синьо), както и линия с 12V

* 1. **Принцип на работа**
     1. **Мобилно приложение**

Приложението е написано на основата на Java и е предназначено за Android мобилни устройства.

Пази информация за потребителя и устройствата му в дистанционна база данни (Firebase), както и локално, на мобилното устройство.

Установява връзка с множество контролни модула чрез Bluetooth Low Energy (BLE), като всеки модул извършва ролята на сървър, докато приложението е клент.

* + 1. **База данни - Firebase**

Базата данни използвана в проекта е “Firestore Database”. Връзката между базата и приложението се установява чрез Firebase библиотеката на Android Studio. Главната цел на базата данни е да пази информация за потребителите и техните свързани устройства.

а) Потребител

* Име
* Имейл
* Парола
* Дата на създаване на акаунта

б) Устройство

* Идентификатор на потребителя
* Име на устройството (заданено от потребителя)
* State на устройството (вкл/изкл, цвят)
* Дата на инсталация на устройството
* Запасени светови опции и други настройки

Тази информацият за устойството се пази в базата като “бакъп” с цел лесно възстановяване на настройките на устройството от страна на потребителя в случай загуба на тази информация от устройството и приложението.

* + 1. **Хардуер**

Микроконтролерът(ESP32) във всеки модул установява връзка с мобилното приложение чрез Bluetooth Low Energy (BLE), което позволява бърз обмен на данни между приложението и множесто микроконтролери.

При отваряне на приложението и свързване със вече запазено BLE устройство (микроконтролера), потребителят може да изпраща заявки към микроконтролерът, като включи/изключи осветлението или смяна на чвете на RGB лентата.

Тези заявки биват приети и обработени от микроконтролерът, който след това извършва зададените операций чрез периферният хардуер в модула и изпраща информация със статуса (успешен, неуспешен …) на заявката обратно към мобилното приложение.

* 1. **Алгоритъм и програмен код**
     1. **Микроконтролер**
        1. **Инициализация**

Инициализират се идентификатори и **Characteristic** обекти и технните **Descriptor** обекти за цвят и за state (влкючено/изключено)

| #define SERVICE\_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b" #define CHARACTERISTIC\_STATE\_UUID "93758842-6624-49aa-a286-abdfff1f4efa" #define CHARACTERISTIC\_CURRENT\_COLOR\_UUID "8c25e317-ac1a-47ee-bedc-53c6241487c2"  BLECharacteristic rgbStripStateCharacteristics(CHARACTERISTIC\_STATE\_UUID, BLECharacteristic::PROPERTY\_READ |  BLECharacteristic::PROPERTY\_NOTIFY |  BLECharacteristic::PROPERTY\_WRITE\_NR); BLEDescriptor rgbStripStateDescriptor(BLEUUID((uint16\_t)0x2903)); BLECharacteristic rgbStripCurrentColorCharacteristics(CHARACTERISTIC\_CURRENT\_COLOR\_UUID, BLECharacteristic::PROPERTY\_READ |  BLECharacteristic::PROPERTY\_NOTIFY |  BLECharacteristic::PROPERTY\_WRITE\_NR); BLEDescriptor rgbStripCurrentColorDescriptor(BLEUUID((uint16\_t)0x2903)); |
| --- |

- Създава се Callback клас за Service-a, която следи дали има свързан клиент със сървъра, и Callback клас за всеки един Characteristic обект.

Callback класовете за “характеристиките” наследяват клас BLECharacteristicCallbacks и имплементират 2 функций отговорни за трафика на информация през Characteristic-ите

| class MyServerCallbacks: public BLEServerCallbacks {  void onConnect(BLEServer\* pServer) {  deviceConnected = true;  };  void onDisconnect(BLEServer\* pServer) {  deviceConnected = false;  } }; |
| --- |

| void onWrite(BLEDescriptor \*desc)  {  Serial.println("Descript write Data received!");  }   void onRead(BLEDescriptor \*desc)  {  Serial.println("Descript read Data received!"); |
| --- |

* + - 1. **Setup()**

**-** Дефинират се in/out пинове

**-** Създава се сървър с име и му се назначава BLEServerCallback

**-** Създава се Service и към него се добавят се Characteristic обектите и техните Descriptor-и

| //STATE  pService->addCharacteristic(&rgbStripStateCharacteristics);  rgbStripStateDescriptor.setValue("Current State");  rgbStripStateCharacteristics.setCallbacks(new StateCharacteristicCallBack());  rgbStripStateCharacteristics.addDescriptor(&rgbStripStateDescriptor); |
| --- |

**-** Сървърът се стартите

**-** Започва се рекламирането на BLE устройството на други устройства с BLE възможности

* + - 1. **Loop()**

**-** При включване на устройсвото (първа итерация на loop()) се задават начални стойности на “характеристиките” и се включват светлините с червен цвят

| **if(!loopStarted) {  rgbStripStateCharacteristics.setValue("off");  rgbStripStateCharacteristics.notify();   rgbStripCurrentColorCharacteristics.setValue("16711680");  rgbStripCurrentColorCharacteristics.notify();   analogWrite(RED\_PIN, red\_value);  analogWrite(GREEN\_PIN, green\_value);  analogWrite(BLUE\_PIN, blue\_value);  loopStarted = true;  }** |
| --- |

**-** На всяка итерация се проверява дали сървърът има връзка с клиент. Ако няма, то тогава се рестартира рекламата на устройството.

| **if (!deviceConnected && oldDeviceConnected) {  delay(500);**  **BLEDevice::startAdvertising();**  **Serial.println("start advertising");  oldDeviceConnected = deviceConnected;  }** |
| --- |

* + 1. **Мобилно приложение**
       1. **Стартиране на приложението**

- При стартиране на мобилното приложение се зарежда **SpashActivity**, чиято работа е да вземе настоящата сесия на приложението и да провери дали има потребител, влезнал в профила си.

- В случей, че има влезнал потребител, се стартира **MainActivity**, което предоставя всички функционалности за управление на светлини.

- В противен случей - ако няма логнат потребител, то се стартира **AuthActivity,** което се състой от екрани за влизане в профил и регистрация на нов профил.

* + - 1. **Вход и регистрация (AuthActivity)**

Профилите на потребителите се пазят и обработрват от “Authentication” инструмента на Firebase. Начините за вход са чрез имейл и парола или чрез профил в Google (oAuth)

**а) Вход**

- Първият екран видим за потребителя при стартиране на **AuthActivity**, е екранът за вход в профил. Влизането в профил се случва чрез въвеждане на имейл и парола.

- При опит за вход се извиква методът “signInWithEmailAndPassword” от класа “FirebaseAuth” - част от библиотеката на Firebase за Android Studio.

| public void signIn(String email, String password, Context context) {  session = new UserSessionManager(context);  mAuth = FirebaseAuth.getInstance();  mAuth.signInWithEmailAndPassword(email, password)  .addOnCompleteListener((Activity) context, new OnCompleteListener<AuthResult>() {  @Override  public void onComplete(@NonNull Task<AuthResult> task) {  if (task.isSuccessful()) {  Log.d(TAG, "signInWithEmail:success");  FirebaseUser user = mAuth.getCurrentUser();  updateUI(user);  session.createUserLoginSession(email, password);  Log.d("preference", session.getUserDetails().toString());  Intent intent = new Intent(context, MainActivity.class);  context.startActivity(intent);  } else {  Log.w(TAG, "signInWithEmail:failure", task.getException());  Toast.makeText(context, "Authentication failed.", Toast.LENGTH\_SHORT).show();  updateUI(null);  }  }  }); } |
| --- |

- При успешен вход се се пълни сесията с информацията на потребителя от базата данни и се стартира **MainActivity**

**-** При неуспешен опит за вход се визуализира съобщение за потребителя с пояснителна информация

- На дъното на екрана за вход има бутон с препратка към страницата за регистация

**б) Регистрация**

**-** Екранът за регистрация се състои от форма с полета за имена, имейл и парола, бутон за “submit” на формата и бутон с препратка към страницата за вход.

- Процесът на регистрация извиква методът “createUserWithEmailAndPassword” от класа “FirebaseAuth”-част от библиотеката на Firebase за Android Studio.

| public void createAccount(String display\_name, String email, String password, Context context) {  mAuth.createUserWithEmailAndPassword(email, password)  .addOnCompleteListener((Activity) context, new OnCompleteListener<AuthResult>() {  @Override  public void onComplete(@NonNull Task<AuthResult> task) {  if (task.isSuccessful()) {  Log.d(TAG, "createUserWithEmail:success");  FirebaseUser user = mAuth.getCurrentUser();  updateUI(user);  UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()  .setDisplayName(display\_name)  .build();  session.createUserLoginSession(email, password);  Log.d("preference", session.getUserDetails().toString());  Intent intent = new Intent(context, MainActivity.class);  context.startActivity(intent);  } else {  Log.w(TAG, "createUserWithEmail:failure", task.getException());  Toast.makeText(context, "Authentication failed.",  Toast.LENGTH\_SHORT).show();  updateUI(null);  }  }  });  // [END create\_user\_with\_email] } |
| --- |

- При успешена регистрация се стартира **MainActivity**

**-** При неуспешена регистрация се визуализира съобщение за потребителя с пояснителна информация

* + - 1. **Начална страница (свързани устройства)**

В началната станица се състои от grid със карти съдържащи името на устройството и бутон за включване/изключване. Всяка карти води до страница за управление на устройството.

* + - 1. **Страница “Моят акаунт”**

Страница с информацията на потребителя и функционалност за менажиране на акаунта

Страницата съдържа имената на потребителя, имейл адрес, дата на регистрация, и друга подобна информация

Също така предоставя възможност за смяна на паролата на профила чрез updatePassword функцията от библиотеката на FireBase или при забравена парола - изпращането на имейл за смяна на паролата с sendPasswordResetEmail()

През тази страница потребителят също така може да излезне от профила си чрез “Log Out” бутон.

При натискане на бутон се извиква FireBase функцията signOut и след това и custom функция за прекратяване на сесията, която запазва всичката информация от сесията в базата данни и след това изтрива сесията и препраща потребителя обратно към **AuthActivity** където има избор между вход и регистрация

| public void logoutUser() {  Log.d("Logged user out", this.pref.toString());  this.editor.clear();  this.editor.apply();  Intent intent = new Intent(\_context, AuthActivity.class);  \_context.startActivity(intent); } |
| --- |

* + - 1. **Страница за сканиране за нови устройства**

- Потребителят има достъп до тази страница през бутон във началната страница. Състои се от списък със всички засечени от сканиращата функция BLE устройства със избрано от разработчика име. По този начин приложението търси и се свързва само със специално за тази цел направените контролни модули.

- Във всеки ред от списъка се вижда името на устройството и бутон за “сдвояване”.

- В този екран също има бутон за връщане към начланата страница

**а)** Сканирането за налични устойства се усъществятва чрез custom класа **FindBLEdevice.java.** Използват се **android.bluetooth** и **android.bluetooth.le** пакетите за процеса на сканиране и за всички други операции свързани с BLE.

| public void scanLeDevice(Activity activity, Context context, ListView availableDevicesListView) {  leDeviceListAdapter = new LeDeviceListAdapter(activity);  availableDevicesListView.setAdapter(leDeviceListAdapter);   ScanCallback leScanCallback = new ScanCallback() {  @Override  public void onScanResult(int callbackType, ScanResult result) {  if(result.getDevice().getName().equals("ESP32 LightHouse")){  leDeviceListAdapter.addDevice(result.getDevice());  leDeviceListAdapter.notifyDataSetChanged();  }  }  };  if (!scanning) {  // Stops scanning after a predefined scan period.  handler.postDelayed(new Runnable() {  @Override  public void run() {  scanning = false;  bluetoothLeScanner.stopScan(leScanCallback);  }  }, SCAN\_PERIOD);   scanning = true;  bluetoothLeScanner.startScan(leScanCallback);  } else {  scanning = false;  bluetoothLeScanner.stopScan(leScanCallback);  }  } |
| --- |

- **bluetoothLeScanner.startScan()** функцията сканира за налични BLE устройства.

- След опреден период от време се вика **bluetoothLeScanner.stopScan()**, която прекратява скранирането.

- При намерено устройство се проверява дали, то е специализиран светлинен модул според името му (всички модули имат едно и също име за услеснено сканиране, но MAC адресите им и UUID-тата на “Service”-ите им са уникални). Ако името съвпада, устройството се добавя в списъка с налични и се визуализира за потребителя чрез **LeDeviceListAdapter()** класа. За всяко устройство в списъка има бутон за сдвояване, който води към DeviceControlActivity (страница за контрол на устройство). Препратката към тази страница става по следния начин:

| Intent intent = new Intent(view.getContext(), DeviceControlActivity.class); intent.putExtra("deviceName", deviceName); intent.putExtra("deviceAddress", deviceAddress); view.getContext().startActivity(intent); |
| --- |

- Приложението установява връзка като клент със GATT Server-а хостнат на микорконтролера **connect()** метода на **BluetoothLeService** класа:

| public boolean connect(final String address) {  if (mBluetoothAdapter == null) {  Log.d(TAG, "BluetoothAdapter not initialized");  return false;  }   if(address == null) {  Log.d(TAG, "Unspecified address.");  return false;  }  if (mBluetoothDeviceAddress != null && address.equals(mBluetoothDeviceAddress) && mBluetoothGatt != null) {  Log.d(TAG, "Trying to use an existing mBluetoothGatt for connection.");  if (mBluetoothGatt.connect()) {  mConnectionState = STATE\_CONNECTING;  return true;  } else {  return false;  }  }   final BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);  if (device == null) {  Log.d(TAG, "Device not found. Unable to connect.");  return false;  }   mBluetoothGatt = device.connectGatt(this, false, mGattCallback);  Log.d(TAG, "Trying to create a new connection.");  mBluetoothDeviceAddress = address;  mConnectionState = STATE\_CONNECTING;  return true; } |
| --- |

* + - 1. **Страница за контрол на устройство**

Страницата за контрол на устройството се състои от няколко елемента.

1. Име на устройството

В началото името на устройството се взима от Service-a за който сме се вързали през приложението, но това може да бъде променено

1. Бутон за смяна на името

Бутон който позволява промяната на името на устройството.

Това променя само името в локалната сесия и в базата данни, но не променя името на Service-a от страна на микроконтролера

1. Бутон за включване/изключване

PowerButton, който изпраща стойности “off”/”on” до отредения за това **Characteristic** на сървъра

1. ColorSeekBar - слайдър за избор на цвета

XML елемент от библиотеката com.rtugeek.android.colorseekbar. Представлява слайдър със предопределен диапозон от цветове.

| <com.rtugeek.android.colorseekbar.ColorSeekBar  android:id="@+id/color\_seek\_bar"  ...  app:colorSeekBarColorSeeds="@array/color\_picker\_slider\_colors"  ...  /> |
| --- |

| <array name="color\_picker\_slider\_colors">  <item>#ff0000</item>  <item>#ff7700</item>  <item>#fff700</item>  <item>#55ff00</item>  <item>#00ff9d</item>  <item>#00d0ff</item>  <item>#004cff</item>colorSeekBar  <item>#1500ff</item>  <item>#9500ff</item>  <item>#fb00ff</item>  <item>#ffffff</item> </array> |
| --- |

При всяка промяне в цвета на слайдъра се взима Color int и се изпаща към отговорният за цвета Characteristic, където е обработен и разбит на три отделни стойности за червено, зелено и синьо.

1. Палитра със запзени цветове

Грид със няколко предварително записани (при първото създаването на инстанция на продукта в сесията и базата данни) цветове.

Накрая след всички налични цветове има бутон за добавяне на нов цвят.

При активацията на тази функция се показва попъп със друг вид цветови селектор - color wheel.

При запазване на нов цвят, той както винаги се запзва във локалната сесия и във базата, при всички други данни за устройството

* + - 1. **Комуникация с устройство**

**а) Клиента (приложението)**

- След свързване на приложението към **GATT Server**-а на микроконтролера, приложението има достъп до всички **Service**-и на сървъра, както и адресите и стойностите на техните **Characteristic**-и.

- Characteristic**-**ите прикачени са средството, чрез което BLE сървъра и клиента комуникират. Service-ите и Chatacteristic-ите имат идентификатори (UUID), чрез които може да се чете и променя информацията пазена в тях.

- Във всеки микроконтролер имаме две инстанции на Characteristic - една за статуса на светлинния източник (включено/изключено) и друг за цвета на RGB лентата.

- Изпращане на нова информация към GATT Server-a се случва по следният начин:

| public void WriteCharacteristic(BluetoothGattCharacteristic characteristic, String newCharValue) {  characteristic.setValue(newCharValue);   if (!mBluetoothGatt.writeCharacteristic(characteristic)) {  Log.e(TAG, String.format("ERROR: writeCharacteristic failed for characteristic: %s", characteristic.getUuid()));  } else {  Log.d(TAG, String.format("writing <%s> to characteristic <%s>", newCharValue, characteristic.getUuid()));  }  } |
| --- |

characteristic.setValue(newCharValue) променя стойността на локално запазения BluetoothGattCharacteristic обект, а mBluetoothGatt.writeCharacteristic(characteristic) изпраща промененият обект към GATT Server-a

- Статуса се пази като обикновен string със 2 възможни стойност - “on”/”off”, а пък информация за цвета се чете и пише като color int конвертиран в string. Цвета се променя и изпраща към сървъра при промяна в стойността на colorSeekBar компонента от библиотеката com.rtugeek.android.colorseekbar в страницата за контрол.

| colorSeekBar.setOnColorChangeListener(new OnColorChangeListener() {  @Override  public void onColorChangeListener(int progress, int color) {  mBluetoothLeService.WriteCharacteristic(mColorCharacteristic, String.valueOf(color));  }  }); |
| --- |

- Четенето на информацията на GATT Characteristic-ите от страна на клиента се осъществява чрез метода readCharacteristic(characteristic) от класа android.bluetooth.BluetoothGatt

**б) Сървър (микроконтролер)**

**-** Четенето на изпратената от клиента(приложението) информация от страна на сървъра се извършва с помоща на наследници на клас **BLE Characteristic Callbacksот** библиотеката <BLEServer.h>

| class ColorCharacteristicCallBack: public BLECharacteristicCallbacks {  //This method not called  void onWrite(BLECharacteristic \*characteristic\_)  {  int color = String(characteristic\_->getValue().c\_str()).toInt();  Serial.println(color);   red\_value = (color >> 16) & 0xFF;  green\_value = (color >> 8) & 0xFF;  blue\_value = color & 0xFF;   analogWrite(RED\_PIN, red\_value);  analogWrite(GREEN\_PIN, green\_value);  analogWrite(BLUE\_PIN, blue\_value);  }   void onRead(BLECharacteristic \*characteristic\_)  {  } }; |
| --- |

Това е callback класа за “Характеристиката” отговоряща за цвета на RGB лентата. При отчитане на нова информация се взима стойността на “характеристиката”, която в случея е color int конвертиран в string и се обработва, така че да се разделят стойностите за трите основни цвята - червено, зелено и синьо. След това трите сигнала за цветовете се изпращат към светлинната лента през съответните пинове.

* + - 1. **Запазване и повторно връзване с устройство**

При първо успешно свързване с устройството, неговите MAC адрес и име се запазва в локалната сесия на потребителя и в последствие - в базата данни.

| private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {  @Override  public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {  ...  if (newState == BluetoothProfile.STATE\_CONNECTED) {  ...  HashSet<String> addressHash = new HashSet<String>();  addressHash.add(mBluetoothGatt.getDevice().getAddress());  session.updateDeviceAddresses(addressHash);  }  ...  } |
| --- |

- В началната страница се визуализира грид пълен с карти, представляващи всяко едно запазено устройство. Те показват името му и бутони с бърз достъп за включване и изключване на светлините.

- При натискане на тези бутони се осъществява връзка със сървъра на микроконтролера чрез MAC адреса му и **connect()** метода на **BluetoothLeService** клас

- При натискане на самата карта на устройството, потребителят бива препратен към **DeviceControlActivity** където може да манипулира устройството свободно. Връзката с устройството, отново се осигурява от **connect()** метода

* 1. **Тестове и резултати**
  2. **Поддръжка и отстраняване на проблеми**
* Обратна връзка от потребителите: Насърчавам потребителите редовно да предоставят обратна връзка за техния опит със системата. Тази обратна връзка се анализира внимателно и се използва за идентифициране на тенденциите и проблемите. След това предприемаме необходимите корекции, за да подобрим цялостната производителност и надеждност на системата.
* Поради липсата на тестове и достатъчно данни за остановяване на често срещани проблеми, няма имплементиран актуален и подробен наръчник за потребителя с информация за настойване, диагностика и поправка на устройствата, но въпреки това има инструкции за употреба, макар и оскъдни

1. **Заключение**
   1. **Обобщение на постигнатите цели**
      1. **Потребителски интерфейс (UI)**

Приложението разполага с интуитивен и естетически приятен UI, който позволява на потребителите лесно да контролират своите RGB осветителни системи.

* + 1. **Документация**

Документация: Беше изготвена изчерпателна техническа документация, описваща дизайна, архитектурата и функционалността на приложението.

* + 1. **Приложения**

Системата има множество приложения в различни области, където интелигентното осветление и енергийната ефективност са приоритет, като домашна автоматизация, търговска автоматизация, архитектура и дизайн, здравеопазване, образование, спорт и развлечения.

Този дипломен проект демонстрира способността ми да проектирам и разработвам софтуерни решения, които отговарят на специфични изисквания и предоставят стойност на потребителите. Той също така подчертава значението на енергийната ефективност и интелигентната автоматизация в различни аспекти на нашия живот, от осветлението до образованието и здравеопазването.

* 1. **Разширения и бъдещо развитие**
     1. **Потребителски интерфейс**

Приложението предлага интуитивен и естетически приятен потребителски интерфейс, който позволява на потребителите лесно да контролират своите RGB осветителни системи. Необходими са обаче допълнителни изследвания и разработки, за да се подобри достъпността и използваемостта на приложението за потребители с различни нива на технически познания.

* + 1. **Съвместимост и продукти**

В този начален етап на разработка приложението поддържа само един модул за контрол на осветление, който е съвеместим с много оскъдна селекция светлинни тела. Това ограничава изключително много потребителската аудитория и създава неудобство за потребителя.

Една голяма стъпка в плана за развитие на проекта е да се създадат множество вариации на физическия модул, които да работят с различни видове осветителни тела, като обикновенни крушки, RGB крушки и повече разновидности на RGB ленти.

* + 1. **Ефективност**

Приложението работи гладко и надеждно на различни устройства и операционни системи, осигурявайки оптимално потребителско изживяване. Необходими са обаче допълнителни усилия за оптимизиране на ефективността и намаляване на консумацията на ресурси

* + 1. **Оптимизация на хардуера**

Планирано е използването на по-малки и по-практични компоненти за намаляване на обема и улеснения на инсталацията на устройството.

Главното необходимо подобрение е заменянето на сегашният чип ESP32 със някой по-малъл микроконтролер, като ArduinoUNO или пък комбинация от изключително компактният ESP8266-01 и отделен блутут модул, тъй като сегашният микроконтролер е ненужно голям и мощен и не бива употребяван в пълния си капацитет

1. **Информация и източници**
   1. **Мобилно приложение**

* <https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview>
* <https://github.com/rtugeek/ColorSeekBar>
* <https://github.com/android/connectivity-samples>
  1. **Хардуер**
* <https://github.com/mo-thunderz/Esp32BluetoothApp>
* <https://newbiely.com/tutorials/esp8266/esp8266-control-led-via-bluetooth>
* <https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/>
* <https://randomnerdtutorials.com/esp32-ble-server-client/>
* <https://web.hosting.nyu.edu/bluetooth-le-low-power-wireless-interactions/w2-build-a-ble-service-with-characteristics-with-read-write-and-notify-properties-by-using-built-in-sensors/>
  1. **База данни**
* <https://firebase.google.com/docs/firestore/quickstart#java_7>
  1. **Други**

1. **Приложения**

Системата за управление на RGB осветлението, разработена за този дипломен проект, има множество приложения в различни области, където интелигентното осветление и енергийната ефективност са приоритет. Някои от тези приложения включват:

* 1. Домашна автоматизация: Системата може да се използва за създаване на персонализирани светлинни сценарии за жилищни пространства, подобрявайки комфорта и атмосферата. Потребителите могат да планират графици за осветление, да регулират яркостта и цвета на светлината и да активират сцени въз основа на тяхното местоположение, време или събитие.
  2. Търговска автоматизация: В търговски обекти системата може да се използва за създаване на динамични дисплеи, привличане на вниманието към определени продукти или създаване на приятна атмосфера за клиентите. Системата може също да се използва за оптимизиране на потреблението на енергия в търговските помещения, намалявайки разходите за електроенергия.
  3. Здравеопазване: В здравните заведения системата може да се използва за създаване на успокояваща и терапевтична среда за пациенти. Светлинните сценарии могат да бъдат персонализирани за различни състояния и нужди, подпомагайки възстановяването и облекчавайки безпокойството.
  4. Образование: В образователните институции системата може да се използва за създаване на ангажираща учебна среда, която подобрява концентрацията и ученето. Светлинните сценарии могат да бъдат адаптирани към различни учебни дейности и графици, създавайки по-гъвкава и ефективна учебна среда.
  5. Спорт и развлечения: В спортни съоръжения и места за забавление системата може да се използва за създаване на вълнуващи визуални ефекти и подобряване на цялостното изживяване на зрителите. Системата може също да се използва за оптимизиране на потреблението на енергия по време на събития, намалявайки въглеродния отпечатък на съоръжението.

Чрез предоставянето на универсално решение за управление на осветлението, системата има потенциала да революционизира начина, по който използваме осветлението в различни аспекти на нашия живот, като подобрява комфорта, ефективността и устойчивостта.