4 BLE Server

U kontekstu *Bluetooth Low Energy* (BLE), GAP (*Generic Access Profile*) i GATT (*Generic Attribute Profile*) su dve važne komponente koja definišu kako BLE uređaji komuniciraju međusobno. Ukratko, GAP upravlja uspostavljanjem konekcije i reguliše interakciju između BLE uređaja, dok GATT definiše kako su podaci organizovani i kako se razmenjuju između uređaja nakon što je konekcija uspostavljena. GAP se oslanja na sloj veze (LL), a GATT na ATT (*Attribute Protocol*) BLE protokol-steka, koji su detaljno obrađeni u materijalu za predavanja. GAP i GATT su ključni za funkcionisanje BLE uređaja i čine osnovu BLE komunikacionog protokola. Poznavanje GAP i GATT je neophodno za razvoj (programiranje) BLE aplikacija.

4.1 GAP

GAP definiše uloge i procedure za BLE uređaje kako bi oni moguli da otkriju jedni druge i uspostave konekciju. GAP definiše četiri uloge uređaja:

Broadcaster (*Emiter*). Ova uloga je optimizovana za primene kod kojih postoji potreba samo za periodičnim emitovanjem podataka u formi paketa za oglašavanje. Na primer, uređaj-termometar, koji periodično emituje podatak o izmerenoj temperaturi zainteresovanim okolnim uređajima je tipičan primer emitera. Emiter šalje podatke u paketima za oglašavanje, a ne putem konekcije, što omogućava bilo kom uređaju koji osluškuje BLE kanale za oglašavanje da primi emitovane podatke.

Observer (*Osmatrač*). Ova uloga je optimizovana za uređaje koji žele da prikupljaju podatke koje emituju emiteri. Uređaj u ulozi osmatrača, osluškuje radio medijum i prima pakete za oglašavanje. Na primer, ulogu osmatrača ima tablet računar koji na svom ekranu prikazuje informaciju o temperaturi koju je dobio od BLE termometra u ulozi emitera.

Central (*Centralni uređaj* ili *centrala*). Zadatak centralnog uređaja je da otkriva prisustvo drugih BLE uređaja i uspostavlja konekcije s njima. Centralni uređaj osluškuje radio medijum, prima pakete za oglašavanje i uspostavlja konekciju sa izabranim uređajem.

Peripheral (*Periferni uređaj* ili *periferija*). Uređaj u ulozi periferije koristi pakete za oglašavanje kako bi omogućio centrali da ga pronađe i nakon toga uspostavi konekciju s njim. Uloga periferije je po pravilu dodeljena jednostavnim i jeftinim BLE uređajima, poput senzora.

Dakle, postoje dva scenarija primene BLE protokola. Prvi scenario je „emitovanje-osmatranje“ i on je optimizovan za veoma jednostavne primene, gde emiter emituje podatke, a osmatrač ih prima. Ovaj scenario na zahteva uspostavljanje konekcije između BLE uređaja i pogodan je samo za elementarne aplikacije. Primeri BLE uređaja-emitera su tzv. BLE bikoni, poput: *Apple iBeacon* (BLE emiter koji periodično emituje pakete za oglašavanje koji se mogu detektovati od strane aplikacija na *iPhone*-u. Koriste se u maloprodajnim objektima za pružanje personalizovanih iskustava kupcima.) i *Google Eddystone* (BLE emiter koji podržava različite vrste podataka koje se emituju, uključujući URL-ove, identifikatore i senzorske podatke.) Drugi scenario je

„centrala-periferija“ i on podrazumeva uspostavljanje konekcije između dva BLE uređaja, jednog u ulozi centrale i drugog u ulozi periferije. Ovaj scenario pruže neuporedivo veće mogućnosti od bazičnog „emitovanje-osmatranje“ načina rada. Pre svega, omogućava dvosmernu komunikaciju, tako da centrala ne samo što može da dobija podatke od periferije, već može i da upravlja radom periferije. Takođe, konekcija omogućava pouzdanu i sigurnu komunikaciju.

Režim „centrala-periferija“ zapravo obuhvata režim „emitovanje-osmatranje“, u smislu da periferija i centrala, pre uspostavljanja konekcije rade u režimu emitovanja, odnosno osmatranja. Periferija oglašava svoje prisustvo periodičnim emitovanje paketa za oglašavanje, dok centrala skenira kanale za oglašavanje. Kada primi paket za oglašavanje od periferije s kojom želi da uspostavi vezu, centrala inicira proceduru otvaranja konekcije (ukoliko to želi).

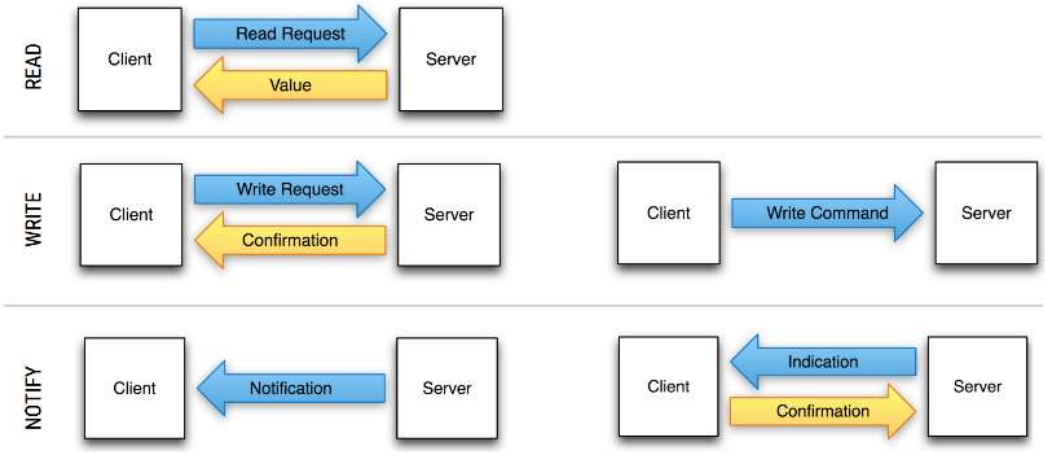
BLE uređaji koji funkcionišu isključivo kao emiteri ne implementiraju dodatne servise, već šalju sve podatke koje imaju putem paketa za oglašavanje. S druge strane, BLE uređaji u ulozi periferije obavezni su da implementiraju barem jedan GATT servis. Preko ovog servisa oni čine dostupnom svoju funkcionalnost i podatke centrali sa kojom su povezani.

4.2 GATT

GATT (*Generic Attribute Profile*) je specifikacija koja definiše način na koji BLE uređaji komuniciraju i razmenjuju podatke. GATT opisuje strukturu podataka i način na koji se ti podaci organiziraju i prenose između BLE uređaja.

4.2.1 GATT server i klijent

Opšta postavka je sledeća. Svaki BLE uređaj poseduje određeni skup podataka koje je spreman da izloži, tj. učini dostupnim, odnosno podeli sa drugim BLE uređajima. Na primer, BLE-termometar može izložiti podatak o izmerenoj temperaturi i podatak o periodi merenja temperature. To onda omogućava npr. mobilnom telefonu da, nakon uspostavljena BLE konekcije, ne samo preuzme podatak o temperaturi, već i da promeni periodu merenja BLE-termometra. Ova vrsta interakcije, radi razmene podataka, ostvaruje se posredstvom GATT servera i GATT klijenta. U konkretnom primeru, GATT klijent je pokrenut na strani centrale (tj. mobilnog telefona), a GATT server na strani periferije (tj. BLE-termometra). Kada je mobilnom telefonu potreban podatak o temperaturi, on se obraća svom GATT klijentu, koji upućuje zahtev GATT serveru BLE-termometra. GATT server pronalazi traženi podatak i njegovu vrednost vraća nazad GATT klijentu. Ako želi da promeni periodu merenja temperature, mobilni telefon se takođe obraća svom GATT klijentu koji taj zahtev, zajedno sa novom vrednošću periode šalje GATT serveru BLE-termometra. GATT server upisuje dostavljenu vrednost u podatak (promenljivu) koji reguliše periodu merenja. Osim transakcija tipa čitanje/upis, GATT takođe podržava notifikacije i indikacije, koje GATT server „samoinicijativno“ šalje GATT klijentu, obično radi dojave nekog izuzetnog događaja (npr. „nizak nivo baterije“). Razlika između notifikacija i indikacija je samo u tome što indikacije zahtevaju da klijent potvrdi prijem, dok su notifikacije jednosmerna obaveštenja, na koja server ne očekuje nikakav odziv klijenta. Tri vrste transakcija između GATT klijenta i servera su ilustrovane na Sl. 1.

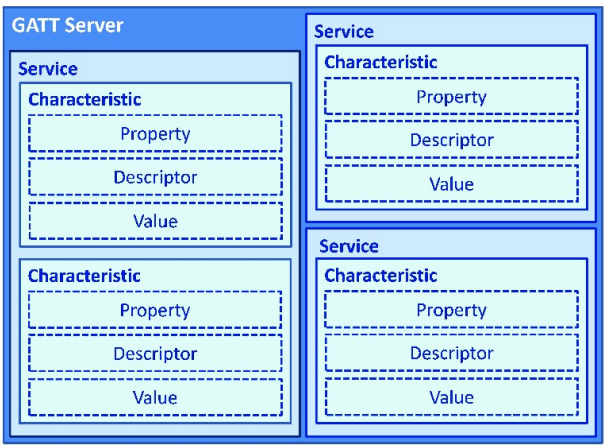


Sl. 1 Vrste GATT transakcija: čitanje, upis sa i bez potvrde, notifikacije i indikacije

Uz sve prethodno, potrebno je naglasiti da uloge GATT klijent i server nisu predodređene ulogama centrala i periferije, u smislu da periferija može igrati samo ulogu GATT servera, a centrala samo GATT klijenta. Nakon što je BLE konekcija uspostavljena, oba uređaja mogu lokalno da pokrenu i GATT server i GATT klijent. Na primer, pretpostavimo da BLE-termometar poseduje sat realnog vremena i displej za prikazivanje tekućeg vremena. Nakon uspostavljene konekcije sa mobilnim telefonom, BLE-termometar, putem svog GATT klijenta može, radi sinhronizacije svog sata realnog vremena, da zatraži informaciju o tekućem vremenu od GATT servera mobilnog telefona - naravno, pod uslovom da je ova informacija dostupna na strani mobilnog telefona.

4.2.2 GATT servisi

Drugi aspekt GATT-a se odnosi na organizaciju podataka na strani GATT servera. Naime, svi podaci koje BLE uređaj izlaže putem GATT servera moraju biti organizovano u skladu sa GATT specifikacijom. Podaci su organizovani u servise; servis sadrže karakteristika, a svaka karakteristika sadrži jedan elementarni podatak koji je dostupan za razmenu između BLE uređaja. Zapravo, karakteristika ne sadrži samo vrednost podatka (*Value*), već sadrži i prateće meta-podatke koji definišu osobine karakteristike (*Properties*) i dodatne informacije o karakteristici (*Descriptor*(*s*)). Osobine definišu kako se pristupa podatku i kako se podatkom manipuliše, odnosno da li je dozvoljeno čitanje i/ili upis podatka i da li je omogućeno slanje notifikacija/indikacija klijentu. Deskriptori sadrže dodatne meta-podatke o karakteristici, poput merne jedinice (da li je vrednost podatka izražena u npr. metrima, sekundama, milisekundama i sl.), formata (da li podatak predstavljen kao ceo ili realni broj i sl.), opsega vrednosti podatka, pa čak i informativni tekstualni opis podatka. Deskriptori su opcioni, za razliku od vrednosti i osobina koje moraju da postoje u svakoj karakteristici. BLE uređaj može da implementira jedan ili više servisa i svi oni su pod kontrolom lokalnog GATT servera. Svi servisi, zajedno, čine profil BLE uređaja (Sl. 2).



Sl. 2 Profil BLE uređaja se sastoji iz jednog ili više servisa; servis sadrži jednu ili više karakteristika, a svaka karakteristika sadrži podatak, osobine i (opciono) deskriptore. Pristup servisima reguliše GATT server

Primer BLE profila
BLE-termometar bi mogao da implementira dva servisa: 1) ***TempService***, preko kojeg povezanom uređaju čini dostupnim svoju funkcionalnost u vezi merenja temperature. Servis *TempService* bi se sastojao iz dve karakteristike: *TempChar*, koja sadrži ne samo vrednost izmerene temperature, već i prateće osobine koje definišu, npr. da se ova karakteristika može samo čitati, ali se u nju ne može upisivati. Takođe, ova karakteristika bi mogla da sadrži i deskriptor koji sadrži informaciju o formatu podatka (recimo, 16-integer) i mernoj jedinci u kojoj je izražena temperatura (^{0}C ili F). *PeriodChar*, koja sadrži podatak o periodi merenja, zajedno sa *Property* koja kaže da je ovu karakteristiku dozvoljeno čitati i upisivati, i, eventualno deskriptorom koji sadrži informaciju o dozvoljenom opsegu periode merenja (npr. između $100\ ms$ i $100\ s$). 2) ***BattService***, preko kojeg omogućava povezanom uređaju da dođe do informacije o trenutnom stanju napunjenosti baterije. Ovaj servis bi sadržao samo jednu karakteristiku: *BattLevelChar* za podatak o nivou napunjenosti baterije, zajedno sa *Property* u kojoj je naznačeno da ovu karakteristiku nije dozvoljeno ni čitati ni u nju upisivati, već da se njena vrednost prenosi GATT klijentu isključivo putem notifikacija.

UUID

Servisi, karakteristike, kao i svi ostali objekti u GATT specifikaciji, imaju svoja imena, po kojima se identifikuju. U prethodnom primeru pojavljuju se imena dva servisa i nekoliko karakteristika. Na primer, kada GATT klijent želi da dobije podatak o temperaturi, on bi trebalo da pošalje zahtev GATT serveru koji bi glasio: „Pošalji mi vrednost karakteristike *TempChar*“. Međutim, u GATT specifikaciji za imenovanje objekata se ne koriste tekstualni identifikatori, poput “*TempChar*“, već se ime, odnosno tip objekta specificira putem UUID-ova. UUID (*Universally Unique Identifier – Univerzalno jedinstveni identifikator*) je, prosto, binarni niz, dužine 128 bita (tj. 16 bajta), koji se koristi za jedinstveno identifikovanje resursa ili objekata u različitim sistemima, pa i kod BLE. U kontekstu BLE, UUID se koriste za identifikaciju servisa, karakteristika i deskriptora. Kada je potrebno imenovati npr. neku karakteristiku, umesto da smišljamo ime karakteristike, mi generišemo UUID koji pridružujemo toj karakteristici. UUID-ovi se generišu pomoću posebnih programa ili *online* na specijalizovanim veb sajtovima. Algoritam generisanja je takav da garantuje da se nikada neće generisati dva ista UUID. Na primer, na linku https://www.uuidgenerator.net/ je dostupan servis za generisanje UUID-ova (Sl. 3). Heksadecimalni zapis koji vidimo je UUID (16 bajta = 32 heksadecimalne cifre). Ovaj UUID je garantovano jedinstven – nikada nije, niti će ikada biti generisan UUID istog sadržaja. Dovoljno je da generisani UUID kopiramo i prenesemo tamo gde nam je potreban. Ukoliko želimo da generišemo još jedan UUID, potrebno je, kao što piše, osvežiti stranicu *online* UUID generatora. Pretpostavimo da je karakteristici koja sadrži podatak o temperaturi pridružen upravo generisani UUID. To znači da bi zahtev koji GATT klijent

šalje GATT serveru radi preuzimanja podatka o temperaturi mogao da glasi: „Pošalji mi vrednost karakteristike čiji je UUID: 0d35ac14-ef3f-45aa-a3c5-b839ea26d8fa“.

Sl. 3 *Online* UUID generator

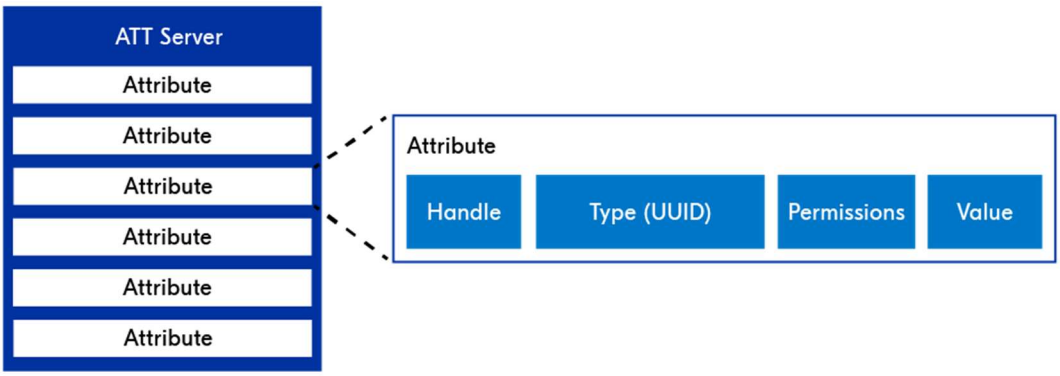
Iz razloga efikasnosti, kao i zbog toga što 16-bajtni UUID-ovi zauzimali veliki deo kratkih BLE paketa, BLE specifikacija predviđa dva dodatna kratka UUID formata: 16-bitni i 32-bitni UUID. Ovi skraćeni formati se mogu koristiti samo za UUID-ove koji su definisani direktno u BLE standardu. UUID-ovi koje uvodi korisnik moraju biti pune dužine. Puni, 128-bitni UUID se rekonstruiše na osnovu skraćene verzije umetanjem 16- ili 32-bitnog kratkog UUID u osnovni BLE UUID (na poziciji x-ova):

xxxxxxxx-0000-1000-8000-00805F9B34FB

BLE specifikacija propisuje UUID-ove za sve tipove, servise i profile koji su definisani ovim standardom. Na primer, BLE standard sadrži specifikaciju servisa *Battery Service*. Ovom servisu je u standardu dodeljen kratki UUID $0x180F$. Ovaj servis sadrži desetak karakteristika od kojih su sve opcione osim karakteristike *Battery Level* koja je obavezna. Kratki UUID karakteristike *Battery Level* je 0x2A19. Puni UUID iste karakteristike bi bio oblika: **00002A19**-0000-1000-8000-00805F9B34FB. Međutim, ukoliko su korisničkoj aplikaciji potrebni dodatni UUID-ovi (npr. korisnik uvodi svoj, novi servis ili karakteristiku), oni moraju imati pun formati i moraju se generisati na veb sajtu *uuidgenerator* (da bi se garantovala jedinstvenost).

<u>Attribute</u>

Atributi su najniži nivo organizacije podataka u GATT. Naime, sve informacije koje sadrži GATT servis upakovane su u atribute. GATT servis možemo zamisliti kao tabelu čija svaka vrsta sadrži jedan atribut, a svaki atribut čuva jedan elementarni korisnički podatak, plus meta-podatke koji opisuju sam atribut, tj. njegov tip, sigurnosna ovlašćenja, itd (Sl. 4). Razmena podataka između GATT servera i klijenata odvija se u obliku atributa.



Sl. 4 Strukturanje podataka po GATT-u

Napomena: Na Sl. 4, umesto GATT piše ATT server (od, *attribute-server*). Naime, ATT je poseban, unutrašnji sloj BLE protokol-steka koji je isključivo zadužen za pristup atributima. GATT je nadogradnja ATT. ATT ne ulazi

u značenja atributa, već samo obezbeđuje pristup atributima. GATT organizuje atribute u karakteristike i servise.

Na Sl. 4 je takođe prikazan i format atributa. Svaki atribut se sastoji iz 4 polja:

Handle: 16-bitna vrednost, u opsegu $0x0001 - 0xFFFF$, koja predstavlja indeks (adresu) atributa u tabeli atributa. *Handle*-ove atributima dodeljuje GATT server. Svaki atribut mora imati jedinstven *handle*. Koristi se za adresiranje atributa. GATT klijent se uvek obraća GATT serveru zahtevom za pristup određenom atributu. U zahtevu, GATT klijent može da navede UUID atributa, ili *handle* atributa. Druga mogućnost je bolja, jer je *handle* značajno kraći od UUID. Kada primi takav zahtev, GATT server pronalazi adresirani atribut, proverava pravo pristupa i izvršava zahtevanu akciju (npr. upis ili čitanje vrednosti atributa).

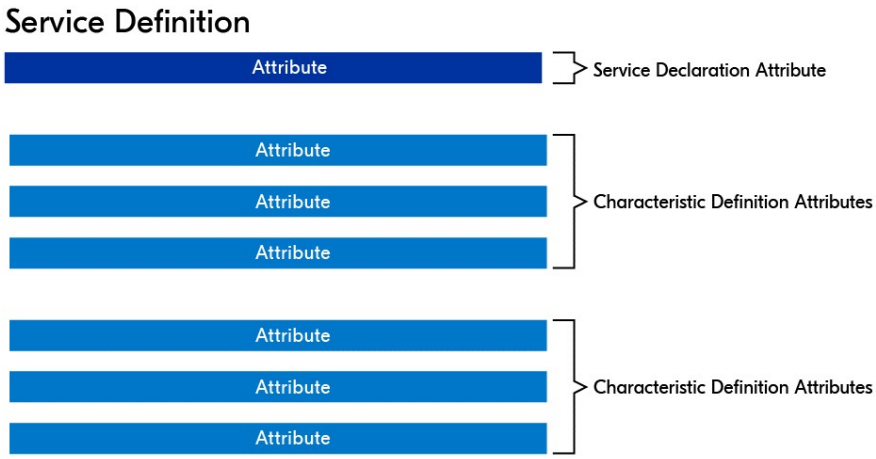
Type. Ovde je smešten UUID atributa, koji specificira prirodu, tj. tip atributa. Dakle, svaki atribut ima svoj UUID (bilo kratki ili pune dužine), koji nam govorio tome šta zapravo predstavlja, odnosno koju informaciju nosi taj atribut.

Permissions: indikacije o pravu pristupa atributu (da li je GATT klijentu dozvoljeno da čita i/ili menja atribut; da li pristup atributu zahteva prethodnu autorizaciju i sl.)

Value: Sadrži konkretan korisnički podatak, npr. senzorsko očitavanje. Takođe, može sadržati i informaciju (meta-podatak) a nekom drugom atributu.

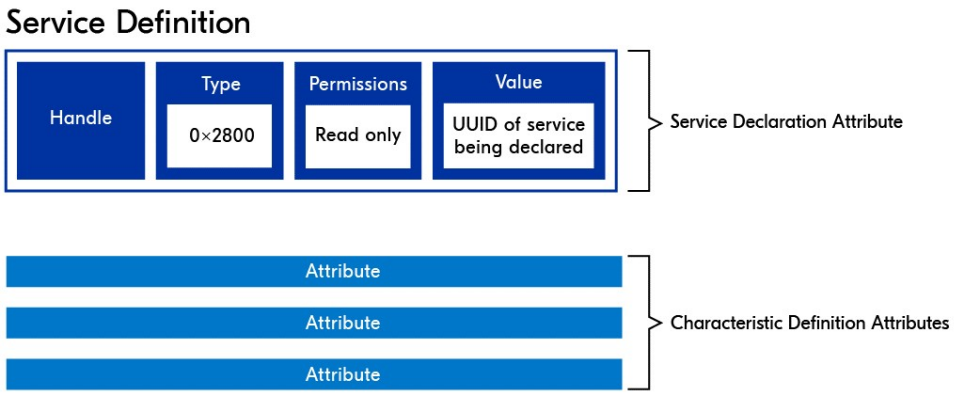
Service

Na Sl. 5 je prikazana organizacija GATT servisa. To je ista tabela kao na Sl. 4, ali sa atributima razvrstanim u logičke celine. Servis počinje atributom za deklaraciju servisa, nakon kojeg slede grupe od po dva ili više atributa, gde svaka grupa čini jednu karakteristiku (odnosno, definiciju karakteristike).



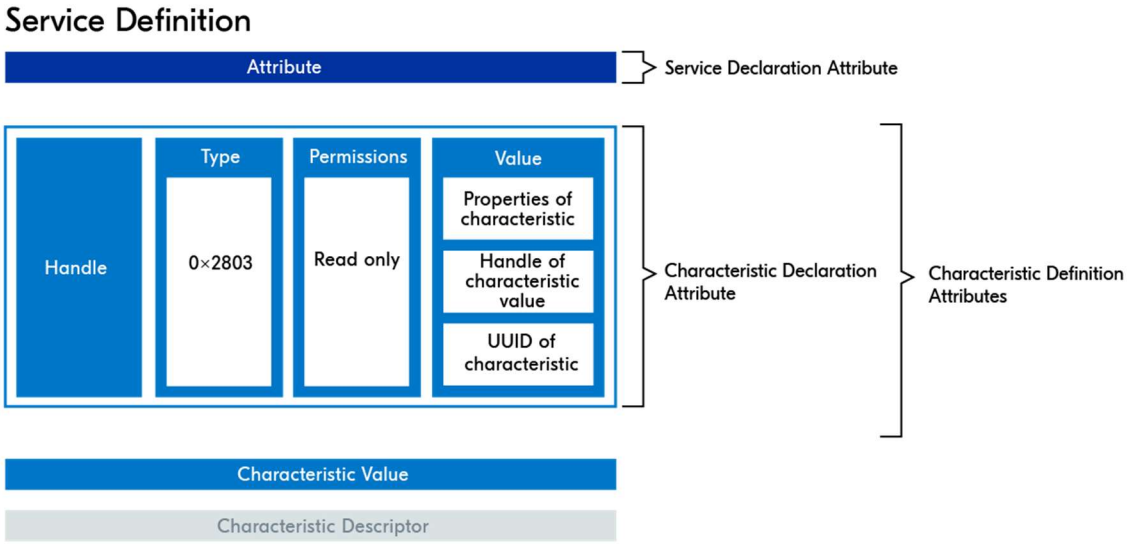
Sl. 5 Logička organizacija GATT servisa

Atribut za deklaraciju servisa. Servis, odnosno definicija servisa, uvek počinje atributom za deklaraciju servisa (Sl. 6). Ovaj atribut sadrži meta-podatke o servisu i ujedno označava početak servisa. U polju *Type* je (uvek) upisan kratki UUID vrednosti $0x2800$. Po ovom UUID znamo da je reč o atributu za deklaraciju servisa. U polju *Permissions* sadržane su indikacije da je ovaj atribut dozvoljeno samo čitati (*read-only*). Polje *Value* sadrži UUID servisa koji se deklariše (kratki, ako je reč o standardnom servisu, ili UUID pune dužine, za korisnički servis).



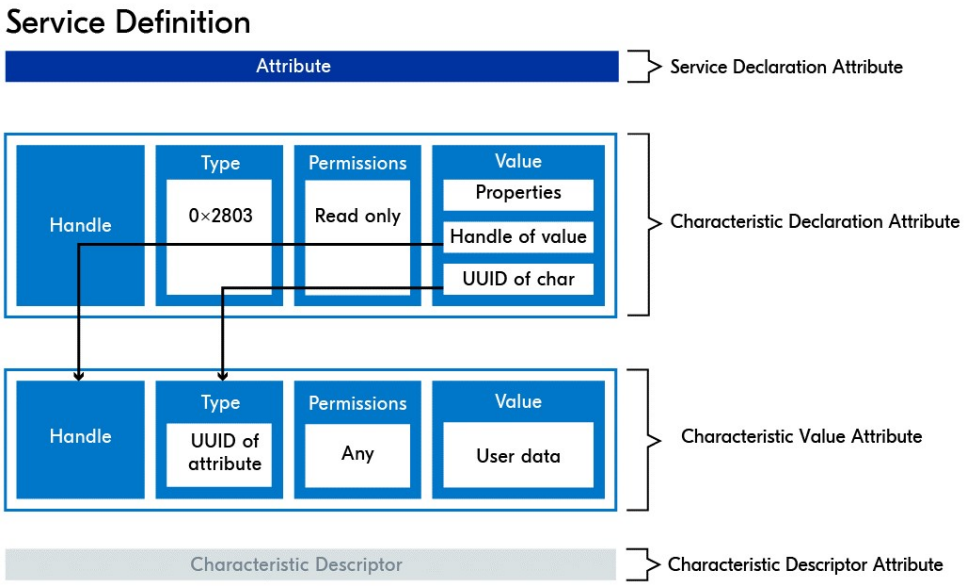
Sl. 6 Atribut za deklaraciju servisa

Atribut za deklaraciju karakteristike. Definicija svake karakteristike počinje atributom za deklaraciju karakteristike (Sl. 7). UUID ovog atributa je $0x2803$. U polju *Permissions* fiksno je postavljena indikacija dozvole čitanja (*read-only*) , što znači da GATT klijent može samo da čita ovaj atribut, ali ne i da menja njegovu vrednost. Polje *Value* sadrži bite informacije o karakteristici i podeljeno je na tri pod-polja: 1) osobine karakteristike (specificira dozvoljene operacije (čitanje, upis, notifikacije, ...) nad karakteristikom kao celinom, a ne samo na ovaj atribut); 2) *handle* vrednosti karakteristike (sadrži *handle* atributa za vrednost ove karakteristike), i 3) UUID karakteristike.



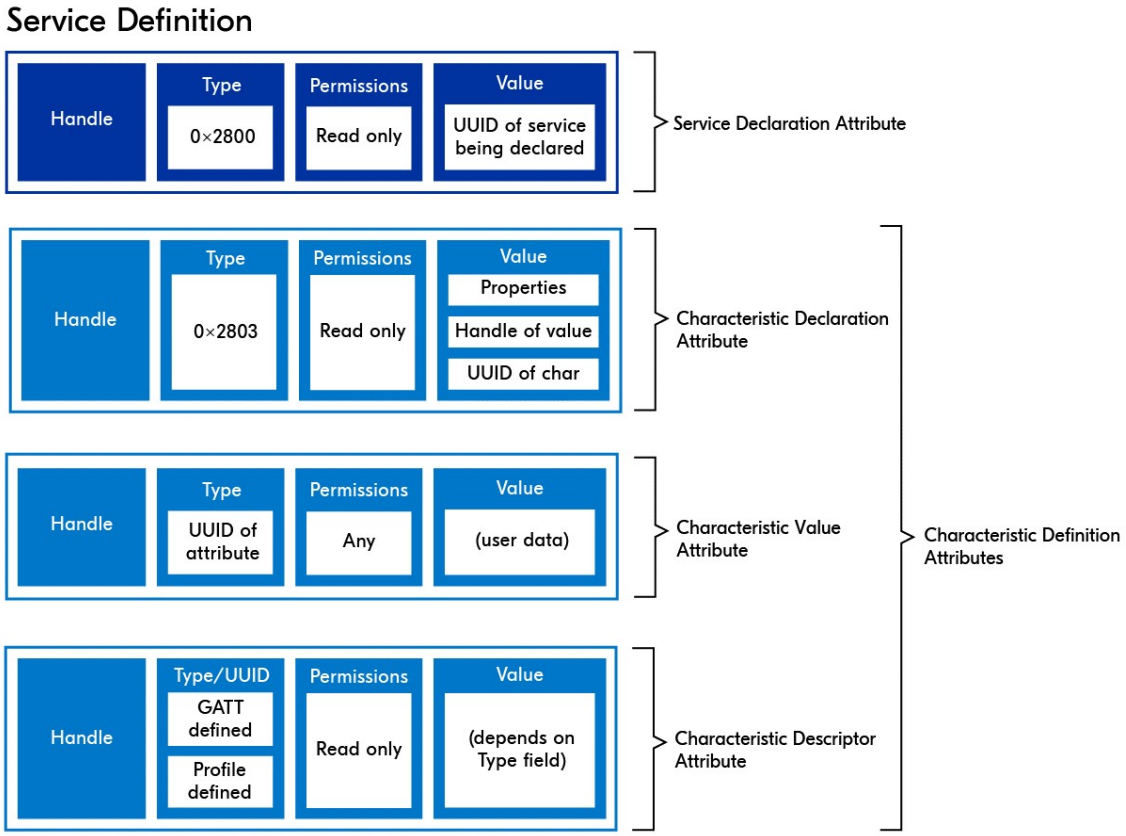
Sl. 7 Atribut za deklaraciju karakteristike

Atribut vrednosti karakteristike. Ovo je drugu atribut u nizu atributa karakteristike (Sl. 8), koji u polju *Value* sadrži vrednost korisničkog podatka o kome se ova karakteristika „brine“. Sadržaj polja *Handle* i *Type* je isti onaj koji je naveden u polju *Value* atributa za deklaraciju karakteristike.



Sl. 8 Atribut vrednosti karakteristike

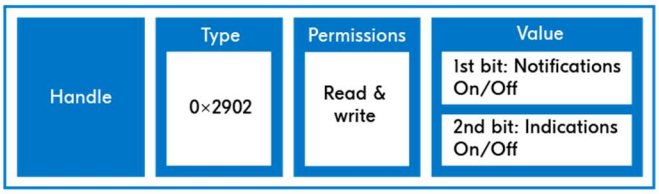
Atribut deskriptora karakteristike. Ovo je treći po redu atribut karakteristike, koji je, inače, opcioni. Ako postoji, sadrži meta-podatke o karakteristici, odnosno pruža GATT klijentu dodatne informacije o prirodi karakteristike (Sl. 9). Postoji nekoliko vrsta deskriptora koji se svrstavaju u dve kategorije: standardni (definisani GATT specifikacijom) i korisnički (definisani od strane korisnika, tj. projektanta GATT servisa). U našim primerima, koristićemo samo jedan deskriptor karakteristike, tzv. CCCD deskriptor.



Sl. 9 Atribut deskriptora karakteristike

Deskriptor za klijentske konfiguracije karakteristike ili CCCD (*Client characteristic configuration descriptor*) je specifičan tip deskriptora karakteristike koji je neophodan u slučajevima kada karakteristika podržava tzv.

serverski-inicirane operacije (misli se na notifikacije i indikacije). Ovaj deskriptor je dozvoljeno upisivati, što omogućava GATT klijentu da dozvoli ili zabrani notifikacije/indikacije koje potiču od konkretne karakteristike. CCCD je standardni deskriptor i njegov UUID je $0x2902$. U polju *Value* postoje samo dva bita, od kojih se jedan odnosi na dozvolu/zabranu notifikacija, a drugi indikacija.



Sl. 10 Deskriptor za klijentske konfiguracije karakteristike (CCCD)

4.3 BLEServer_1

Razvojni modul ESP32-S3 poseduje ugrađeni BLE 5.0 transiver. Ovaj transiver koristi istu antenu kao i WiFi. Za Android programiranje ESP32 BLE aplikacija na raspolaganju je nekoliko biblioteka, koje pružaju podršku za kreiranje GATT servisa, oglašavanje i konektovanje. U ovoj sekciji je predstavljen primera realizacije GATT servisa. ESP32-S3 igra ulogu BLE periferije na kojoj je pokrenut BLE server.

U listingu L.1 je dat programski kôd minimalnog BLE servera, koji implementira jedan servis sa jednom karakteristikom koja može da se čita i upisuje preko BLE konekcije. Krenimo sa objašnjenjem kôda.

Na početku programa uključene su tri biblioteke koje su potrebne za programiranje BLE servera. Ove biblioteke su sastavni deo Arduino IDE i nije ih potrebno instalirati.

BLE server koji realizujemo ima svoje „čitljivo“ ime, koje je definisano u liniji 5. BLE server sadrži jedan servis. Pošto će to biti naš servis, a ne neki od servisa definisanih BLE standardom, neophodno je da izaberemo UUID servisa, odnosno da na veb sajtu *uuidgenerator.com* generišemo UUID i kopiramo ga u program (linija 6). Takođe, potrebno je definisati i UUID za jedinu karakteristiku koju će servis imati (linija 7).

U funkciji *setup*(), u liniji 13, BLE server je inicijalizovan i postavljeno je njegovo ime. Tačnije, efekat funkcije *BLEDevice::init*() je inicijalizacija i startovanje BLE protokol-steka.

> **Napomena 1**: Obratimo pažnju na sintaksu. „*BLEDevice*“, kao i „*BLEServer*“ i „*BLEService*“ su klase, definisane u uključenim „.h“ fajlovima. Kao što znamo, klasa može imati i statičke funkcije, koje važe na nivou cele klase i nisu vezane za objekte koji se instanciraju iz te klase. Takva je upravu funkcija *init*(). Statička funkcija se poziva preko imena klase i znaka „::“, kao npr. *BLEDevice::init*().

U liniji 14, kreira se i pokreće GATT server, funkcijom *createServer*(), koja je, takođe, statička funkcija u klasi *BLEDevice*. Funkcija vraća pointer na kreirani GATT server (**pServer*). U sledećoj liniji (15), GATT serveru se, pozivom funkcije *createService*(), kaže da kreira GATT servis. Pointer na kreirani servis je **pServer*. Za sada, ovaj servis je prazan, odnosno sadrži samo atribut za deklaraciju servisa (prvi atribut u tabeli servisa u kome je smešten UUID servisa).

> **Napomena 2**: „*BLEService *pService*“ je pointer na objekat tipa (klase) *BLEService*. Naime, u funkciji *createService*() je kreiran (instanciran) objekat klase *BLEService* i funkcija vraća pointer na ovaj objekat. Kao što znamo, članovima objekta (njegovim funkcijama (metodama) i promenljivama (atributima)), se pristupa preko pointera operatorom “->“.

U liniji 16, pozivom funkcije *createCharacteristic*(), servisu **pService* je naloženo da kreira karakteristiku. Funkciji za kreiranje karakteristike se prenosi UUID karakteristike (prvi argument) i indikacije o pravu pristupa karakteristici (drugi argument). S obzirom na „PROPERTY_READ“ i „PROPERTY_READ“ u linijama 18 i 19,

očigledno je da je dozvoljeno čitanje i upis ove karakteristike. Na ovaj način, u pozadini, kreirana je karakteristika koja ima dva argumenta. Prvi je atribut za deklaraciju karakteristike i sadrži UUID karakteristike i *Properties*. Drugi atribut je predviđen za vrednost karakteristike, koja još uvek nije dodeljena karakteristici. Funkcija funkcije *createCharacteristic*() zapravo kreira objekat klase *BLECharacteristic* i vraća pointer na taj objekat koje se smešta u promenljivu **pCharacteristic*.

Napomena 3: Čini se kao da se u pozivu funkcije *createCharacteristic*() koriste tri, a ne dva ulazna argumenta. Međutim, treba uočiti operator „|“ između „BLECharacteristic::PROPERTY_READ“ i „BLECharacteristic::PROPERTY_WRITE“. „I“ je logički ILI operator, tako da je drugi argument u pozivu funkcije zapravo rezultat ILI operacije između dve navedene vrednosti. Ovaj argument definiše sadržaj polja *Properties* u prvom argumentu karakteristike. Svaki bit u ovom polju ima određeno značenje. PROPERTY_READ i PROPERTY_WRITE su, zapravo, celobrojne konstante koje su, zajedno sa još nekoliko indikatora, definisane kao statički članovi klase *BLECharacteristic*:

```
static const uint32_t PROPERTY_READ      = 1<<0; // 1
static const uint32_t PROPERTY_WRITE     = 1<<1; // 2
static const uint32_t PROPERTY_NOTIFY    = 1<<2; // 4
static const uint32_t PROPERTY_BROADCAST = 1<<3; // 8
static const uint32_t PROPERTY_INDICATE  = 1<<4; // 16
static const uint32_t PROPERTY_WRITE_NR  = 1<<5; // 32
```

Dakle, sadržaj polja *Property* je „0...011“, što dozvoljava GATT klijentu da čita i upisuje vrednosti ove karakteristike.

Vrednost upravo kreirane karakteristike (odnosno, vrednost podatka koji nosi karakteristika) se definiše u liniji 20, pozivom funkcije *setValue*(). Ova vrednost se upisuje u drugi atribut karakteristike, odnosno u polje *Value* atributa za vrednost karakteristike. Kao što vidimo, ovaj podatak je tipa string.

GATT servis je startovan u liniji 21. U ovom trenutku, GATT server je spreman da odgovara na zahteve GATT klijenta. Međutim, GATT klijent je na nekom drugom uređaju, koji prvo treba da se konektuje sa našim uređajem kako bi se uspostavila veza između GATT klijenta i servera. Procedura konektovanja, kao što je to objašnjeno u sekciji o GAT, podrazumeva da periferni uređaj (a to je naš ESP32-S3) emituje pakete za oglašavanje kako bi neki drugi uređaj-centrala detektovao njegovo prisustvo i tek onda inicirao uspostavljanje konekcije. Svrha programskog kôda u linijama 23-26 je upravo podešavanje i startovanje oglašavanja.

U liniji 23 kreiran je oglašivač (možemo ga zamisliti kao servis/proces BLE uređaja koji je zadužen za oglašavanje). Podrazumevani paketi za oglašavanje nose samo elementarne podatke, kao npr. ime uređaja. Međutim, često je korisno u oglašavanje uvrstiti i neke dodatne informacije, kao npr. spisak servisa koje uređaj podržava (tačnije, spisak UUID-ova podržanih servisa). To bi onda omogućilo uređaju-centrali da odluči da li zaista želi da uspostavi konekciju sa našim uređajem. Međutim, paketi za oglašavanje su veoma kratki (31 bajt). Ukoliko standardni 31-bajtni paket za oglašavanje nije dovoljan za sve potrebne podatke, BLE predviđa mogućnost da uređaj-osmatrač (centrala), neposredno nakon prijema paketa za oglašavanje, zatraži od oglašivača sekundarne podatke o oglašavanju – tzv. *Scan Response*. *Scan Response* može da sadrži maksimalno 254 bajta. U konkretnom primeru, u *Scan Response* paket je uvršten UUID implementiranog servisa (linija 24). U liniji 25 je setovan odgovarajući indikator u osnovnom paketu za oglašavanje kao indikacija centrali da postoji i sekundarni paket. Konačno, u liniji 26, pozivom funkcije *startAdvertising*(), startovano je oglašavanje.

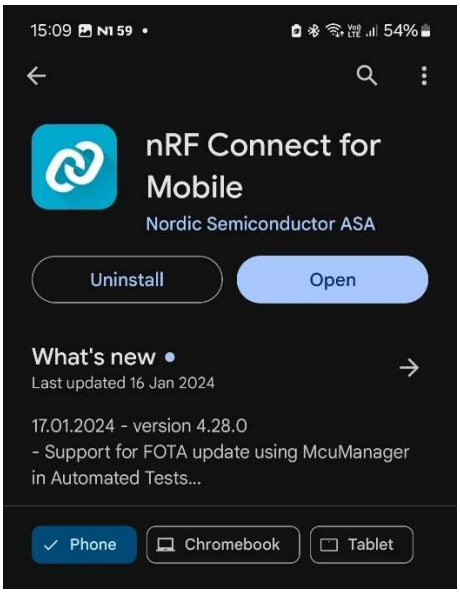
Dakle, sada se čeka da neki drugi uređaj, u ulozi centrale, primi paket za oglašavanje našeg uređaja i odluči da uspostavi konekciju. Preko uspostavljene konekcije, taj drugi uređaj imaće mogućnost da pristupa podatku karakteristike našeg uređaja i da ga eventualno menja.

L. 1. BLEServer_1

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

#define BLE_DEVICE_NAME     "ESP32 BLE Server"
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE!");
  //inicijalizacija BLE i kreiranje servisa
  BLEDevice::init(BLE_DEVICE_NAME);
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                         CHARACTERISTIC_UUID,
                                         BLECharacteristic::PROPERTY_READ |
                                         BLECharacteristic::PROPERTY_WRITE);
  pCharacteristic->setValue("Hello World");
  pService->start();
  //podesavanje i startovanje oglasavanja
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  BLEDevice::startAdvertising();
  Serial.println("Now you can read/write the characteristic at your phone!");
}

void loop() {
  delay(100);
}
```

4.4 *nRF Connect for Mobile* - tutorijal

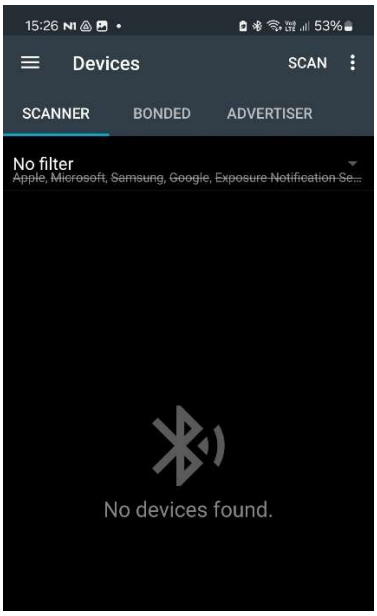
Kako testirati program koji smo upravo napisali?



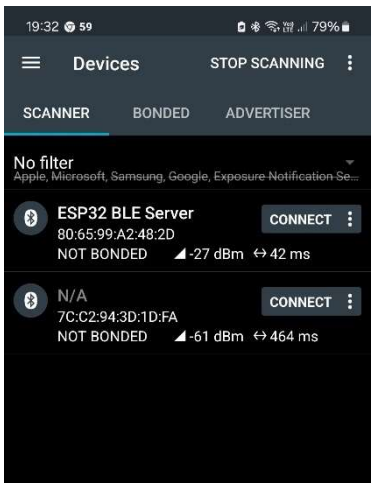
Za razvoj i testiranje BLE aplikacija, veoma su korisni uslužni programi tipa „Bluetooth scanner“ ili „BLE scanner“. Ovakav jedan program se instalira na laptopu ili PC računaru sa podrškom za BLE ili na mobilnom telefonu. (Skoro svi mobilni telefoni imaju podršku za BLE, dok mnogi PC računari, pa i neki laptopovi nemaju). Posredstvom *BLE scanner*-a je moguće skenirati BLE frekventne kanale i pronalaziti okolne BLE uređaje, uspostavljati konekciju sa njima, preuzimati definicije njihovih servisa, čitati i upisivati karakteristike i primati notifikacije. Jedan od najpraktičnijih programa iz ove kategorije je *nRF Connect for Mobile*. Dakle, potrebo je da ovu aplikaciju instalirate na svom mobilnom telefonu. Inače, kompanije *Nordic Semiconductor* je renomirani proizvođač RF SoC komponenti za BLE i srodne bežične tehnologije.

Nakon pokretanja aplikacije *nRFConnect*, vidimo ekran kao na slici nRF.1. Uočavamo tri kartice. Neka ostane izabrana kartica SCANNER. Na ovoj kartici se prikazuju podaci o otkrivenim BLE uređajima. Kartica je prazna, jer skeniranje nije ni uključeno. Za uključenje skeniranje dodirnuti SCAN (gore levo).

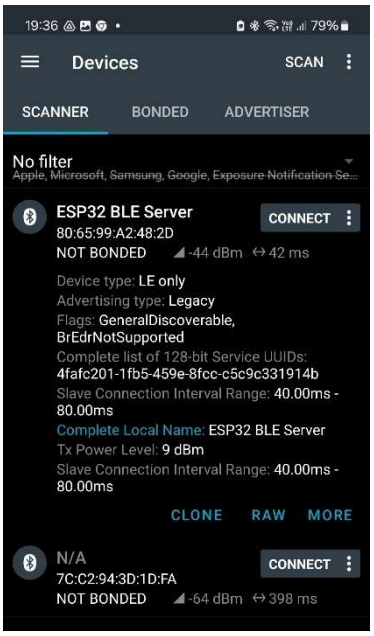
Vidimo da su pronađena dva BLE uređaja (slika nRF.2). Naš uređaj je prvi na listi – prepoznajemo ga po imenu „ESP32 BLE Server“. Ispod imena uređaja, navedena je njegova MAC adresa. „NOT BONDED“ znači da mobilni telefon i naš uređaj nisu „uvezani“ (objašnjenje se može naći u materijalu za predavanja). Takođe, vidimo i nivo prijemnog signala ($-27\ dBm$ je prilično visok nivo). Ako dodirnemo stavku za naš uređaj, prikazaće se dodatni podaci (slika nRF.3). Sve prikazane podatke, aplikacija *nRFConnect* je saznala iz paketa za oglašavanje (primarnog i sekundarnog). Uočimo da je tu i UUID servisa koji naš uređaj implementira.



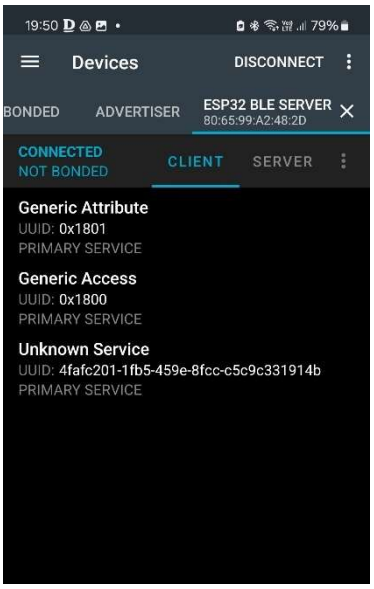
(nRF.1)



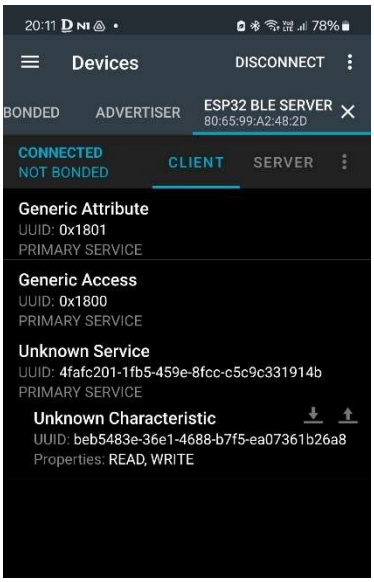
(nRF.2)



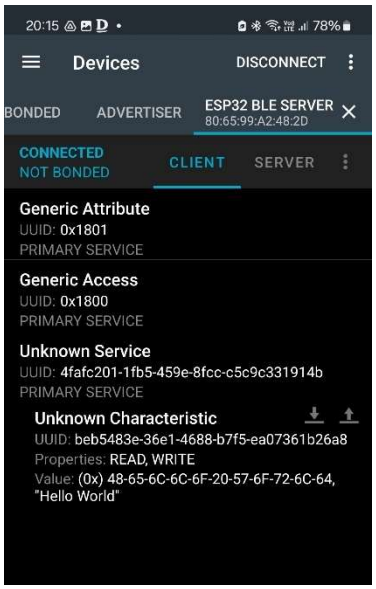
(nRF.3)



(nRF.4)



(nRF.5)



(nRF.6)

Za konektovanje uređaja, dodirnuti CONNECT. Nakon konektovanja, otvara se nova kartica, sa imenom uređaja (nRF.4). Na strani *nRFConnect* pokrenut je GATT klijent, koji je stupio u kontakt sa GATT serverom našeg uređaja i od njega prikupio sve podatke o implementiranim servisima. Procedura prikupljanja podataka se zove *Service and Characteristics Discovery* i obuhvata razmenu većeg broja paketa između dva uređaja.

Ovo što vidimo na kartici je kompletan profil našeg jednostavnog uređaja, koji sadrži dva obavezna atributa na početku i jedan servis.

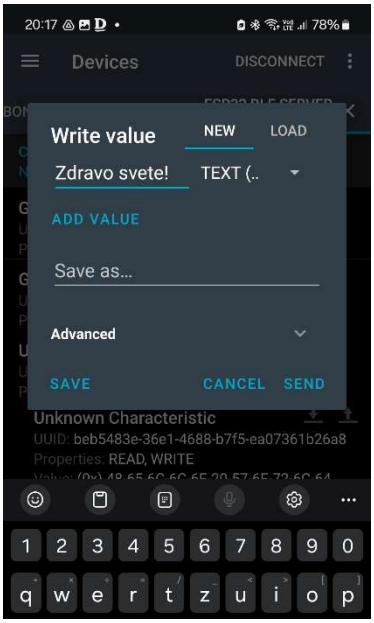
Dva atributa imaju kratke UUID-ove (zato što su definisana standardom). *nRFConnect* prepoznaje ove UUID-ove i zato zna šta da napiše iznad (*Generic Attribute* i *Generic Access*). Pritiskom na ove atribute može se videti njihov sadržaj.

Na trećem mestu, prepoznajemo servis koji smo implementirali. *nRFConnect* je ovaj servis označio kao *Unknown Service*. To je zato što ovaj servis nije definisan standardom (ima dugački UUID), niti smo mi, prilikom pisanja programa, servisu pridružili čitljivo ime. Isto važi i za karakteristike. Kada *nRFConnect* naiđe na objekte sa dugačkim UUID, takav objekat će označiti kao „*Unknown*“.

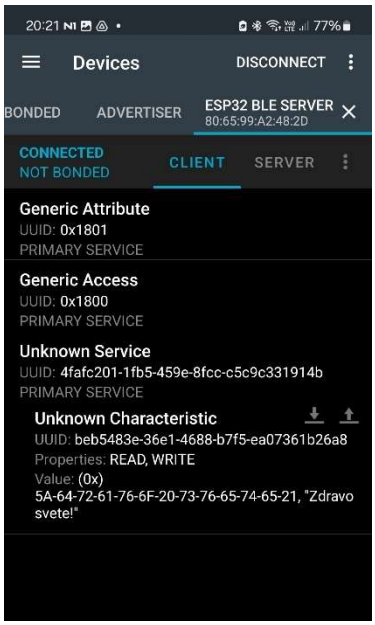
Dodirom na stavku za „*Unknown Service*“, možemo videti sadržaj servisa, odnosno sve njegove karakteristike (slika nRF.5). U ovom jednostavnom servisu postoji samo jedna karakteristika, koju prepoznajemo po UUID. Takođe, vidimo da je dozvoljeno čitanje i upis karakteristike, ali ne vidimo njenu vrednost. Da bismo pročitali vrednost karakteristike potrebno je dodirnuti strelicu usmerenu naniže (asocijacija na *download*). Uočavamo tekst „*Hello World*“ (slika nRF.6).

Vrednost karakteristike možemo promeniti upisom nekog drugog teksta. Za upis u karakteristiku potrebno je dodirnuti strelicu usmerenu naviše (*upload*). Otvara se dijalog za unos vrednosti karakteristike (slika nRF.7). Ispod stavke NEW, potrebno je izabrati tip vrednosti. Od svih ponuđenih opcija, biramo TEXT (UTF-8) – to je ASCII tekst. Novu vrednost pišemo u polju ispod VALUE. Ova vrednost se šalje našem uređaju dodirom na SEND i upisuje u atribut za vrednost karakteristike. Ponovno čitanje karakteristike vraća novi tekst (slika nRF.8).

BLE konekcija između mobilnog telefona i našeg uređaja se raskida dodirom na DISCONNECT ili zatvaranjem kartice sa imenom našeg uređaja. Međutim, ako se vratimo na karticu SCANNER i ponovo pokušamo da konektujemo naš uređaj, nećemo uspeti. Konektovanje je moguće tek posle resetovanja ESP32-S3. Ovo svakako nije korektno, a šta se zapravo desilo i kako omogućiti višestruko konektovanje/diskonektovanje biće objašnjeno u sledećem primeru.



(nRF.7)



(nRF.8)

4.5 BLEServer_2

U ovom primeru (listing L.2), projektujemo BLE periferijski uređaj koji je nešto složeniji u odnosu na uređaj iz prethodnog primera. Uređaj poseduje LED diodu i taster. Nakon uspostavljene BLE konekcije, centralni uređaj ima mogućnost da upravlja blinkanjem LED i dobija informaciju o broju pritisaka na taster. GATT servis periferijskog uređaja sadrži dve karakteristike. Prva karakteristika, koja može da se čita i upisuje, sadrži podatak o periodi blinkanja LED. Vrednost 0 ove karakteristike isključuje blinkanje LED; vrednost koja je različita od nule predstavlja periodu blinkanja u milisekundama. Druga karakteristika sadrži podatak o broju pritisaka na taster. Dozvoljeno je čitanje ove karakteristika, ali ne i upis. Informacija o broju pritisaka na taster se prenosi centrali putem notifikacija. Uz to, prekid BLE konekcije isključuje blinkanje LED. Primer, takođe, pokazuje kako se rešava problem u vezi ponovljene konekcije, koji je uočen u prethodnom primeru.

Nekoliko opštih napomena:

- Promenljive za periodu blinkanja (*blinkPeriod*) i brojač pritisaka na taster *(btnPressCounter)* su deklarisane sa tipom *uint16_t* (linije 11 i 13), a ne sa tipom *int*, ili *unsigned*. Kod ESP32-S3, promenljive tipa *int* i *unsigned* su dužine 32 bita (zato što je ESP32-S3 32-bitni procesor). Ako bi se isti program kompajlirao za neki drugi, ali 16-bitni procesor, promenljive *int* i *unsigned* bi bile 16-bitne. Radi univerzalnosti, i izbegavanja konfuzije oko toga kolika je dužina promenljivih, onda kada je podatak o dužini promenljive bitan (a u ovom primeru jeste bitan), dobra je praksa takve promenljive deklarisati sa tipom koji ima tačno definisanu dužinu, poput *uint16_t*, koji predstavlja 16-bitni neoznačeni ceo broj. Takođe, postoje tipovi *uint8_t*, *uint32_t*, ali i tipovi *int8_t*, *int16_t* i *int32_t*.
- GATT servis ovog uređaja ima dve karakteristike. Zato su nam potrebna tri UUID-a, koja generišemo na sajtu https://www.uuidgenerator.net/ - linije 20-22.
- Pointeri na BLE server i dve karakteristike su deklarisani kao globalne promenljive zato što se koriste u više funkcija (linije 24-26).
- Program je razbijen na nekoliko funkcija. U linijama 28-43 su tzv. *callback* funkcije, koje služe za dojavu promene statusa BLE konekcije i pristizanje zahteva za upis vrednosti karakteristike. U linijama 44-69 je funkcija (*init_service*) za startovanje BLE servera, kreiranje GATT servisa i pokretanje GATT servera. Startovanje oglašavanja je izdvojeno u posebnu funkciju - *start_advertising*(), linije 70-77.

***Callback* funkcije**. Aplikacioni program, koji vidimo u listingu L.2, samo je spoljašnja „ljuska“ kompletnog softvera koji se izvršava na našem uređaju. Softver na koji se oslanjamo je skriven u bibliotekama koje su uključene na početku programa (u suštini, to su različiti delovi BLE protokol-steka). Zadatak aplikacionog program je da konfiguriše i pokrene BLE stek, koji nakon toga samostalno radi. *Callback* funkcije su uobičajeni mehanizam putem kojeg „unutrašnji“ softver dojavljuje aplikacionom programu pojavu određenih događaja. Za svaki takav događaj vezana je jedna *callback* funkcija. Na primer, nakon što je uspostavljena BLE konekcija, BLE stek poziva na izvršenje *callback* funkciju *onConnect*() – linije 30-32, a kada se konekcija raskine poziva funkciju *onDisconnect*() – linije 33-35. Kôd u okviru *callback* funkcija pišemo mi. U konkretnom primeru, kôd ove dve funkcije je veoma jednostavan. Uvedena je globalna promenljiva *deviceConnected* koja sadrži indikaciju o trenutnom statusu BLE konekcije. Kada se konekcija uspostavi (*onConnect*), *deviceConnected* dobija vrednost HIGH, kada se konekcija raskine, dobija vrednost LOW. Dakle, preko ove promenljive, ostatak programa može da sazna trenutni status konekcije. Uočavamo da su *callback* funkcije *onConnect* i *onDisconnect* upakovane u posebnu klasu, *bleSeverCallbacks*, koja nasleđuje bibliotečku klasu *BLEServerCallbacks*. Ime izvedenoj klasi (*bleSeverCallbacks*) smo dali mi. *Callback* funkcije se povezuju sa BLE serverom pomoću funkcije *setCallbacks*() u liniji 50.

Callback funkcije se mogu definisati i za svaku pojedinačnu karakteristiku. Razlog za to postoji ukoliko je dozvoljen upis vrednosti karakteristike. Uvek kada GATT klijent uputi zahtev za upis vrednosti karakteristike,

GATT server će upisati dostavljenu vrednost u atribut za vrednost karakteristike i pozvaće odgovarajuću *callback* funkciju. U konkretnom primeru, dozvoljen je upis u karakteristiku 1 (ona koja čuva period blinkanja), a odgovarajuća *callback* funkcija je *onWrite*(), definisana u klasi *char1Callbacks* koja je izvedena iz klase *BLECharacteristicCallbacks* (linije 37-43). U ovoj *callback* funkciji, nova vrednost karakteristike se prenosi u promenljivu *blinkPeriod*, koja se koristi u ostatku programa kao perioda blinkanja. U prvoj liniji funkcije *onWrite*(), linija 40, funkcijom *getValue*() se preuzima novoupisana vrednost karakteristike, u obliku stringa. Pošto je promenljiva *blinkPeriod* tipa *uint16_t*, neophodna je konverzija tipa. String očitan funkcijom *getValue*() je dužine dva bajta. Prvi bajt, na poziciji 0, je bajt manje, a bajt na poziciji 1 je bajt veće težine. *Callback* funkcija *onWrite*() je povezana sa karakteristikom 1 u liniji 58.

GATT servis. Procedura kreiranja GATT servisa je slična kao u prethodnom primeru. Glavna razlika je u tome što sada servis sadrži, ne samo jednu, već dve karakteristike. Prvo se kreira karakteristika 1, linije 52-58, a onda i karakteristika 2, linije 59-65. Vidimo da je za karakteristiku 1 dozvoljeno čitanje i upis, a za karakteristiku 2, čitanje i notifikacije. Karakteristikama su dodeljene početne vrednosti, prepisane iz odgovarajućih promenljivih. Sadržaj promenljive *blinkPeriod* se prepisuje u karakteristiku 1 (linija 57), a sadržaj *btnPressCounter* u karakteristiku 2 (linija 65). Sintaksa argumenata funkcije *setValue*() je pomalo neobična: `setValue((uint8_t*)&blinkPeriod, 2)`. Prvi argument funkcije *setValue*() je niz bajtova, a drugi dužina tog niza. Naime, „`(uint8_t*)&blinkPeriod`“ konvertuje promenljivu *blinkPeriod* u niz bajtova. Pošto je ova promenljiva tipa *unit16_t*, njena dužina je 2 bajta.

Za karakteristiku 2 dozvoljene su notifikacije. Zbog toga je ova karakteristika proširena CCCD deskriptorom, preko kojeg GATT klijent može da dozvoljava ili zabranjuje notifikacije. CCCD deskriptor se kreira u liniji 64, pozivom funkcije *addDescriptor*(*new BLE2902*()). Sadržaj CCCD deskriptora je uvek isti i unapred je pripremljen u klasi BLE2902.

Oglašavanje. Procedura za startovanje oglašavanja (identična kao u prethodnom primeru) izdvojena je u funkciji *start_advertising*(), linije 70-77. Ova funkcija se poziva u funkciji *setup*(), odmah nakon kreiranja i pokretanja GATT servisa. To znači da će uređaj početi sa oglašavanjem odmah nakon reseta (punjenja programa). Međutim, nakon što se uspostavi BLE konekcija, oglašavanje prestaje. Nakon što se tekuća konekcija prekine, a da bi uređaj mogao da prihvati novu konekciju, oglašavanje je neophodno ponovo uključiti. To se postiže u funkciji *loop*(), linije 90-101. Naime, neprekidno se ispituje da li se status konekcije promenio (linija 91). Ako se status promenio i pri tom je novi status „konekcija prekinuta“, oglašavanje se startuje (obnavlja) u liniji 98. Takođe, uočimo da se u liniji 97, u promenljivu *blinkPeriod* upisuje 0, što za posledicu ima gašenje LED.

Notifikacije. Notifikacije su predviđene za karakteristiku 2. Ideja je da svaki pritisak na taster inicira notifikaciju kojom će se GATT klijentu preneti nova vrednost brojača pritisaka na taster (linije 102-110). Nakon što se detektuje pritisak na taster, brojač pritisaka, *bntPressCounter* se inkrementira i njegova nova vrednost se prenosu u karakteristiku 2. Iako se vrednost karakteristike 2 promenila, notifikacije se neće sama od sebe generisati, već ju je potrebno programski inicirati (pozivom funkcije *notify*(), linija 107), ali samo ako je BLE konekcija uspostavljena.

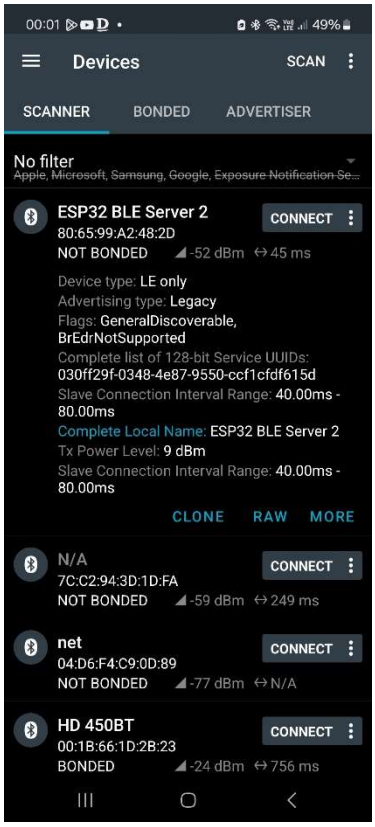
L. 2. BLEServer_2

```
1    #include <BLEDevice.h>
2    #include <BLEUtils.h>
3    #include <BLEServer.h>
4    #include <BLE2902.h>
5
6    const int BTN_PIN = 20;  //pin tastera
7    const int LED_PIN = 5;   //pin LED diode
8
```

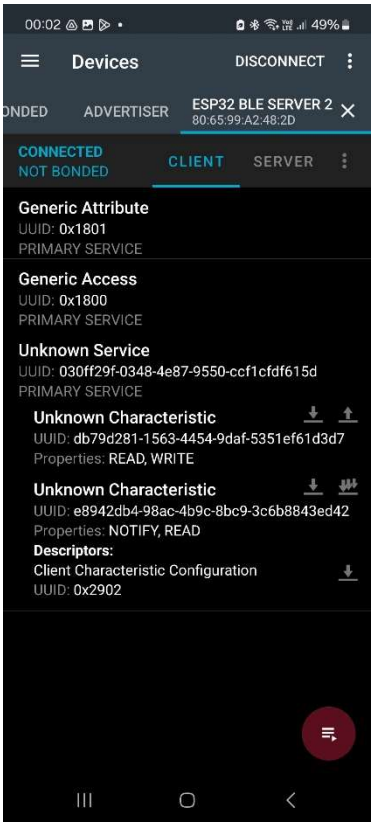
```
9   int btnState = LOW;             //trenutno stanje tastera
10  int lastBtnState = LOW;         //prethodno stanje tastera
11  uint16_t btnPressCounter = 0;   //brojac pritisaka na taster
12  int ledState = LOW;             //stanje LED
13  uint16_t blinkPeriod = 0;       //period blinkanja LED
14  unsigned long currentTime = 0;  //tekuce vreme u ms
15  unsigned long lastBlinkTime = 0;//tren. posl. LED blinka
16  int deviceConnected = LOW;      //stanje BLE konekcije
17  int lastDeviceConnected = LOW;  //prethodno stanje BLE konekcije
18
19  #define BLE_DEVICE_NAME       "ESP32 BLE Server 2"
20  #define SERVICE_UUID          "030ff29f-0348-4e87-9550-ccf1cfdf615d"
21  #define CHARACTERISTIC_1_UUID "db79d281-1563-4454-9daf-5351ef61d3d7"
22  #define CHARACTERISTIC_2_UUID "e8942db4-98ac-4b9c-8bc9-3c6b8843ed42"
23
24  BLEServer* pServer = NULL;
25  BLECharacteristic* pCharacteristic1 = NULL;
26  BLECharacteristic* pCharacteristic2 = NULL;
27
28  //Callback funkcije BLE servera
29  class bleServerCallbacks: public BLEServerCallbacks {
30      void onConnect(BLEServer* pServer) {
31        deviceConnected = HIGH;
32      };
33      void onDisconnect(BLEServer* pServer) {
34        deviceConnected = LOW;
35      }
36  };
37  //Callback funkcije karakteristike za periodu blinkanja
38  class char1Callbacks: public BLECharacteristicCallbacks {
39      void onWrite(BLECharacteristic *pCharacteristic) {
40        std::string value = pCharacteristic->getValue();
41        blinkPeriod = (value[1] << 8) + value[0];
42      }
43  };
44  //Inicijalizacija BLE i GATT
45  void init_service(){
46    //Inicijalizacija i startovanje BLE servera
47    BLEDevice::init(BLE_DEVICE_NAME);
48    pServer = BLEDevice::createServer();
49    //Kreiranje GATT servisa
50    pServer->setCallbacks(new bleServerCallbacks());
51    BLEService *pService = pServer->createService(SERVICE_UUID);
52    //Kreiranje karakteristike za periodu blinkanja
53    pCharacteristic1 = pService->createCharacteristic(
54                                CHARACTERISTIC_1_UUID,
55                                BLECharacteristic::PROPERTY_READ |
56                                BLECharacteristic::PROPERTY_WRITE);
57    pCharacteristic1->setValue((uint8_t*)&blinkPeriod, 2);
58    pCharacteristic1->setCallbacks(new char1Callbacks());
59    //Kreiranje karakteristike za brojac pritisaka na taster
60    pCharacteristic2 = pService->createCharacteristic(
61                                CHARACTERISTIC_2_UUID,
62                                BLECharacteristic::PROPERTY_READ |
63                                BLECharacteristic::PROPERTY_NOTIFY);
64    pCharacteristic2->addDescriptor(new BLE2902());
65    pCharacteristic2->setValue((uint8_t*)&btnPressCounter, 2);
66    //Startovanje GATT servera
67    pService->start();
68    Serial.println("GATT service started");
69  }
70  //Startovanje oglasavanja
71  void start_advertising(){
```

```
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  BLEDevice::startAdvertising();
  Serial.println("Advertising started");
}

void setup() {
  Serial.begin(115200);
  pinMode(BTN_PIN, INPUT_PULLDOWN);
  pinMode(LED_PIN, OUTPUT);
  init_service();
  start_advertising();
}

void loop() {
  currentTime = millis();
  btnState = digitalRead(BTN_PIN);
  //Pracenje statusa BLE konekcije
  if(deviceConnected != lastDeviceConnected){
    if(deviceConnected == HIGH){
      //Ako treba nesto uraditi u trenutku uspostavljanja
      //BLE konekcije, to bi se onda ovde uradilo
    }else{
      //BLE konekcija je raskinuta
      blinkPeriod = 0;
      start_advertising();
    }
    lastDeviceConnected = deviceConnected;
  }
  //Brojanje pritiska na taster i slanje notifikacije (ako ima BLE konekcije)
  if(btnState == HIGH && lastBtnState == LOW){
    ++btnPressCounter;
    pCharacteristic2->setValue((uint8_t*)&btnPressCounter, 2);
    if(deviceConnected == HIGH){
      pCharacteristic2->notify();
    }
  }
  lastBtnState = btnState;
  //Upravljanje blinkanjem
  if(blinkPeriod > 0){
    if(currentTime - lastBlinkTime > blinkPeriod){
      ledState = !ledState;
      lastBlinkTime = currentTime;
    }
  }else{
    ledState = LOW;
  }
  digitalWrite(LED_PIN, ledState);
  delay(20);
}
```

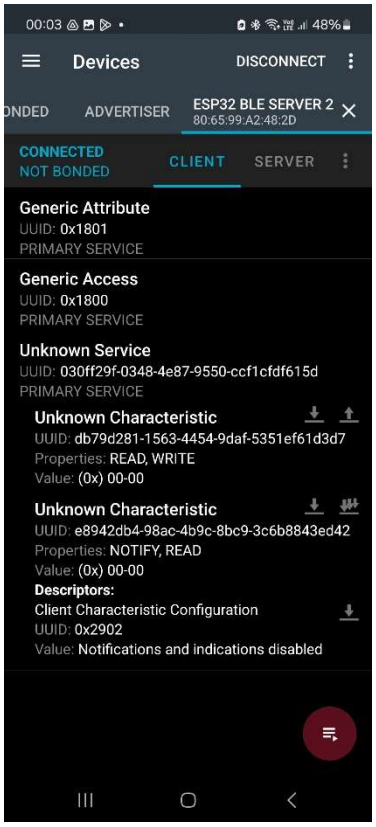
Testiranje. Za testiranje programa BLEServer_2 koristimo aplikaciju *nRFConnect*. Na slici nRF.9 vidimo da je naš uređaj otkriven. Prepoznajemo UUID novog servisa. Nakon konektovanja, u novoj kartici (slika nRF.10), prikazan je servis sa dve karakteristike. Uočavamo da je za prvu karakteristiku dozvoljeno čitanje i upis, a za drugu notifikacije i čitanje. Takođe, druga karakteristika ima i deskriptor (CCCD). Za čitanje vrednosti dve karakteristike i deskriptora druge karakteristike treba redom pritisnuti strelice usmerena naniže. Vidimo da je sadržaj obe karakteristike 0. Sadržaj CCCD-a je takav da notifikacije i indikacije nisu dozvoljene (slike nRF11).



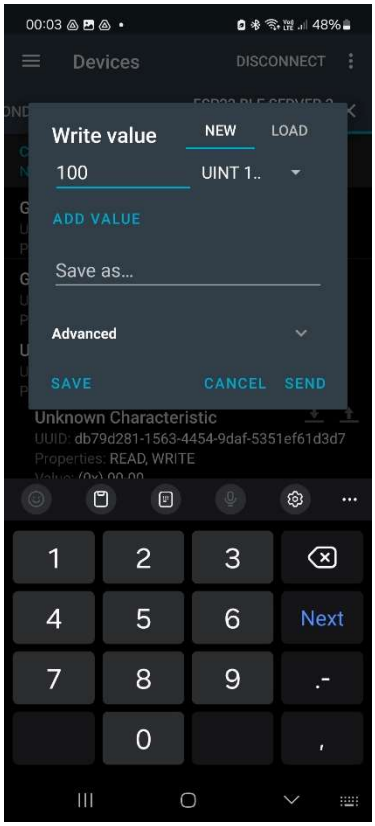
(nRF.9)



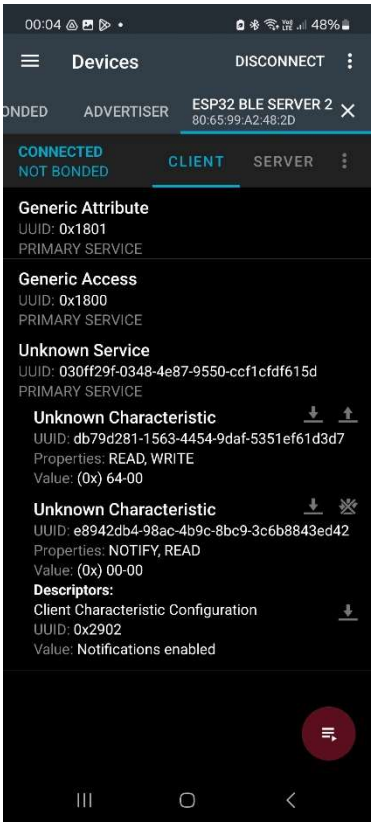
(nRF.10)



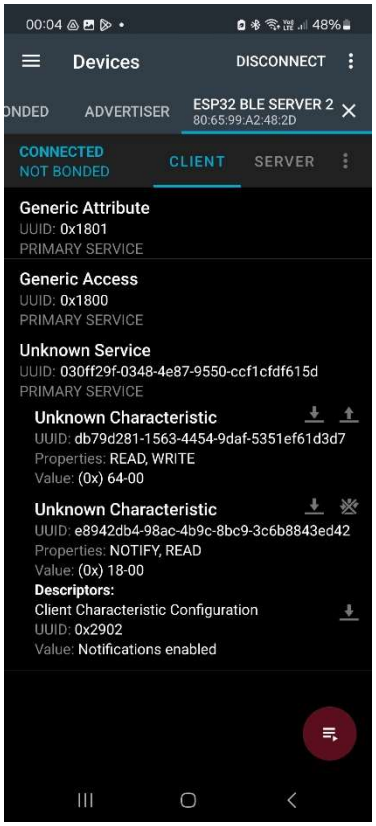
(nRF.11)



(nRF.12)



(nRF.13)



(nRF.14)

Promenimo sada vrednost karakteristike 1 (ona koja definiše periodu blinkanja) – pritisak na strelicu naviše uz prvu karakteristiku, što otvara dijalog za unos vrednosti (nRF.12). Za tip vrednosti treba birati UINT16 (Little Endian). UINT16, zato što je podatak ove karakteristike tog tipa – neoznačeni ceo broj dužine 2 bajta. *„Little*

Endian“ zato da bi GATT klijent prilikom slanja ove vrednosti GATT serveru prvo poslao njen niži, a onda viši bajt. Nakon pritiska na SEND, vrednost periode blinkanja ($100\ ms$) odlazi uređaju i LED počinje da blinka. *nRF Connect* takođe prikazuje novu vrednost ove karakteristike, nRF.13, (0x64 = 100_{10}).

Da bi GATT klijent primao notifikacije, ono prvo moraju biti dozvoljene – pritisak na tri strelice naniže uz drugu karakteristiku. Efekat je da je vrednost CCCD promenjena i on sada pokazuje da su notifikacije dozvoljene (nRF.14). Svaki klik na taster inicira slanje notifikacije, a ukupan broj pritisaka možemo pratiti u polju *Value* druge karakteristike (u heksadecimalnoj notaciji).

4.6 Zadatak

Zadatak 4.1 (20): Cilj ovog projekta je realizacije BLE perifernog uređaja za merenje temperature na bazi ESP32-S3 platforme. Uređaj treba da sadrži senzor temperature i dve LED i da implementira odgovarajući GATT servis, prema sledećim funkcionalnim zahtevima:

Prenos podataka o temperaturi: Podatak o temperaturi se prenosi BLE centrali na dva načina: putem čitanja (READ) i periodičnih notifikacija (NOTIFY).

Perioda notifikacija: Nakon uspostavljanja BLE konekcije, uređaj periodično šalje notifikacije o temperaturi na svakih 5 sekundi. Centrala treba da ima mogućnost podešavanja ove periode u opsegu od 1 do 255 sekundi, sa korakom 1 sekunda.

Pristup senzoru temperature: ESP32-S3 pristupa senzoru temperature samo kada centrala zahteva čitanje vrednosti temperature ili kada je potrebno poslati notifikaciju.

Indikatori statusa:

- Zelena LED signalizira status BLE konekcije (svetli dok je konekcija uspostavljena).
- Crvena LED signalizira razmenu podataka između BLE centrale i periferije: kratkotrajni treptaj (na primer, trajanja $50\ ms$) pri svakom slanju notifikacije ili pri pristiglom zahtevu za čitanje/upis neke karakteristike.

Za testiranje koristiti program *nRF Connect*.

Zadatak 4.2 (30): Modifikovati BLE uređaj iz Zadatka 4.1 tako da realizuje dva režima slanja notifikacija: sinhroni i asinhroni.

Sinhroni režim. Uređaj šalje notifikacije centrali kako je to opisano u Zadatku 1:

- Notifikacije se šalju periodično, uz mogućnost da centrala promeni period notifikacija.
- Senzor temperature se očitava samo kada treba poslati notifikaciju ili kada centrala zahteva čitanje vrednosti temperature.

Asinhroni režim. Slanje notifikacije je inicirano promenom vrednosti temperature:

- Notifikacija se šalje uvek kada se vrednost temperature promeni za 0.1°C u odnosu na poslednju vrednost dostavljenu centrali (notifikacijom ili putem zahteva za čitanje).
- Tokom trajanja BLE konekcije, ESP32-S3 periodično očitava senzor temperature sa periodom od 1 sekunde.

Upravljanje režimima:

- Podrazumevani način slanja notifikacija je sinhroni i primenjuje se nakon uspostavljanja BLE konekcije.

- Centrala prebacuje uređaj u asinhroni način slanja notifikacija postavljanjem periode notifikacija na 0.
- Postavljanje periode notifikacija na vrednost različitu od 0 vraća uređaj u sinhroni način rada.

- ♦ ♦ ♦ -