Sustav za udaljeni nadzor pametne kuće temeljen na platformi ESP32-C3 i servisima AWS IoT

Čuljak, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 685

SUSTAV ZA UDALJENI NADZOR PAMETNE KUĆE TEMELJEN NA PLATFORMI ESP32-C3 I SERVISIMA AWS IOT

Marko Čuljak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 685

SUSTAV ZA UDALJENI NADZOR PAMETNE KUĆE TEMELJEN NA PLATFORMI ESP32-C3 I SERVISIMA AWS IOT

Marko Čuljak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 685

Pristupnik:	**Marko Čuljak (0036513739)**
Studij:	Računarstvo
Profil:	Računalno inženjerstvo
Mentor:	prof. dr. sc. Hrvoje Džapo

Zadatak: **Sustav za udaljeni nadzor pametne kuće temeljen na platformi ESP32-C3 i servisima AWS IoT**

Opis zadatka:

Upoznati se s arhitekturom i razvojnim alatima za mikrokontroler ESP32-C3. Upoznati se s mogućnostima servisa AWS IoT za udaljeno upravljanje (AWS IoT OTA, AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Device Defender, AWS IoT Fleet Provisioning). Razviti pokazni sustav prikladan za umrežavanja senzora i aktuatora koji se koriste kod pametnih kuća. Sustav treba omogućiti spajanje novih čvorova putem WiFi-sučelja na lokalnu pristupnu točku (Wi-Fi provisioning), registraciju novih senzorskih čvorova na AWS infrastrukturu (korištenjem servisa AWS IoT Fleet Provisioning), nadogradnju programske potpore (korištenjem servisa AWS IoT OTA) i pristup zadnje poznatom stanju čvora u slučaju privremenog gubitka veze (AWS IoT Device Shadow). Izraditi pokaznu web aplikaciju na AWS infrastrukturi koja će omogućiti prikupljanje podataka sa senzora, prikaz izmjerenih vrijednosti i pohranu u bazu podataka.

Rok za predaju rada: 28. lipnja 2024.

Želim izraziti duboku zahvalnost svom mentoru, prof. dr. sc. Hrvoju Džapi, na pruženoj prilici i podršci te savjetima tijekom studija. Prijateljima i kolegama hvala na pomoći i prenesenom znanju.

Na kraju, najiskrenije hvala onima čija podrška i vjera nikada nije bila upitna – mojoj obitelji i djevojci. Vaše razumijevanje, strpljenje, motivacija i ljubav bili su temelj ovog uspjeha.

Hvala vam od srca!

Sadržaj

1. Uvod

U današnjem vremenu karakteriziranom sveprisutnošću međusobne bežične povezanosti i pametnog upravljanja uređajima putem interneta, možemo zaključiti da je jedna od najvažnijih karakteristika i funkcionalnosti modernih uređaja upravo mogućnost povezivanja na internet i njegovog daljinskog upravljanja. Popularnost pametnih kućanskih uređaja, kao što su pametni klimatizacijski uređaji, perilice suđa, televizori, *„Roomba"* roboti (roboti za autonomno usisavanje) i pametne digitalne brave, neprestano raste jer olakšavaju i pojednostavljuju svakodnevni život. Ta rastuća popularnost zahvatila je proizvođače ugradbenih uređaja koji najednom grade svoje *IoT* (engl. *Internet of Things*) platforme za kontroliranje i nadzor svojih uređaja ili traže takva rješenja u već naprednim i razvijenim platformama koje nude industrijski divovi kao što su *AWS*, *Microsoft, Google* i *IBM*.

Cilj ovog diplomskog rada je upoznati se s mogućnostima *AWS IoT* platforme i servisima koji se pružaju, a služe za udaljeno upravljanje *IoT* uređaja. Neki od tih servisa su: *AWS IoT OTA, AWS IoT Device Shadow* i *AWS IoT Fleet Provisioning*. Rezultat rada trebao bi biti cjelokupni sustav za udaljeno upravljanje uređaja temeljeno na ESP32 obitelji mikrokontrolera. Takav pokazni sustav treba biti prikladan za umrežavanje senzora i aktuatora koji se koriste kod pametnih kuća, a upravljanje i prikaz senzorskih podataka treba biti izvedeno putem *web* aplikacije na *AWS* infrastrukturi.

Rad je organiziran tako da je prvo izrađen opis sustava unutar kojega se razrađuju zahtjevi i arhitektura sustava (poglavlje 2). Zatim, dan je pregled i opis korištenih komponenti i protokola, a sastoji se od razrade tema kao što su *ESP* platforma mikrokontrolera, postupak *Wi-Fi provisioning*a (spajanja mikrokontrolera na novu *Wi-Fi* mrežu) te *MQTT* protokol (poglavlje 3). Nakon pregleda komponenti slijedi poglavlje koje opisuje mogućnosti i proces korištenja *AWS IoT* servisa (poglavlje 4). Sama implementacija *AWS IoT* servisa i komponenti objašnjena je u poglavlju 5. Na kraju, u poglavlju 6, prikazan je rezultat sustava u formi konačne *web* aplikacije i njenih funkcionalnosti.

2. Opis sustava

2.1. Zahtjevi na funkcionalnost sustava

Zadatak diplomskog rada bio je razviti funkcionalni pokazni sustav baziran na obitelji uređaja ESP32 i servisima *AWS* (engl. *Amazon Web Services*) *IoT* (engl. *Internet of Things*) *cloud* platforme koji bi bio prikladan za umrežavanje raznovrsnih senzora i aktuatora te njihovo udaljeno upravljanje korištenjem internetske veze. Iako su implementacije sustava pametnih kuća raznovrsne, u ovome radu naglasak će biti na korištenju (simuliranog) senzora za mjerenje temperature i vlažnosti zraka te upravljanju svjetlećom diodom (*LED*-om). Razvijeni sustav trebao bi svojim korisnicima pružiti osnovu pri razvoju i implementaciji različitih sustava pametnih kuća, a samom uporabom sustava korisnici bi trebali moći pristupiti podacima senzora temperature i vlage zraka te udaljeno upravljati svjetlećom diodom spojenom na sami sustav.

Kao i kod većine ozbiljnijih i komercijalnih uređaja koji svojom domenom pripadaju u sustave pametnih kuća i Interneta stvari (*Internet of Things*), pokazni sustav korisniku mora omogućiti spajanje novih čvorova putem *Wi-Fi* sučelja na lokalnu pristupnu točku kako bi čvor mogao uspješno pristupiti računalnom oblaku (engl. *cloud*) i omogućiti udaljeno upravljanje čvora putem internetske veze. Metoda kojom se to omogućuje naziva se *Wi-Fi provisioning*.

U slučaju prvog pristupa računalnom oblaku, čvor se mora moći automatski registrirati na *cloud* infrastrukturu. Bez ovog zahtjeva i automatizacije registracijskog procesa, a u slučaju masovne proizvodnje ili jednostavno s ciljem da se sustav proširi većim brojem čvorova (uređaja), svaki novi čvor morao bi se prvo ručno dodavati i registrirati na *cloud* infrastrukturu, a zatim ručno isprogramirati novim *firmware*-om ili na drugi način u čvor spremiti nove pristupne certifikate kako bi spajanje na računalni oblak bilo uspješno.

S obzirom na to da se upravljanje uređaja i njegov nadzor obavlja korištenjem *Wi-Fi* sučelja, internetske mreže te *cloud* infrastrukture, u cijelome lancu može nastati pogreška te, slijedno tome, uređaj može privremeno izgubiti vezu. U tom slučaju, mora se omogućiti uvid u zadnje poznato stanje sustava. Konkretno, uvid u zadnje stanje svjetleće diode (*LED*-a).

Krajnji zahtjev sa strane računalnog oblaka jest implementacija nadogradnje programske podrške bežičnim putem (engl. *FOTA, Firmware over-the-air*). Implementacijom *FOTA*-e

omogućuje se smanjenje troškova održavanja, produljenje životnog vijeka uređaja, popravak pogrešaka u kodu i sigurnosnih propusta te dodavanje novih funkcionalnosti ili optimiziranje tehničkih karakteristika uređaja.

Udaljeno upravljanje i nadzor svakog registriranog uređaja mora biti dostupno putem pokazne *web* aplikacije. U sklopu aplikacije treba se moći pristupiti svim prikupljenim podatcima sa senzora korištenjem baze podataka koja je također na *cloud*u. Osim prikaza prikupljenih podataka putem korisničkog sučelja, mora se omogućiti prikaz trenutnog stanja svjetleće diode (*LED*-a) i njeno udaljeno upravljanje.

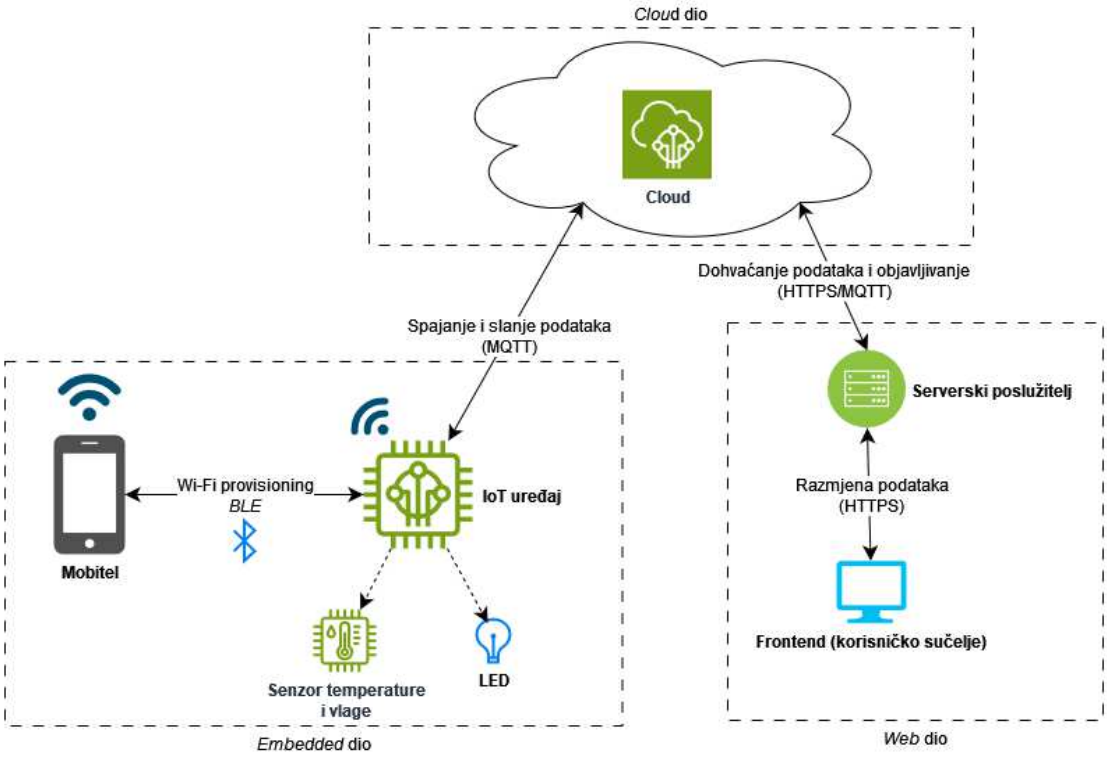
2.2. Arhitektura sustava

Dijagram cjelokupnog sustava vidljiv je na slici 2.1. Jasno su iskazana tri dijela na koja je sustav podijeljen: *embedded*, *cloud* i *web* dio sustava.

Embedded dio sustava sastoji se od: *IoT* uređaja (mikrokontrolera u ESP32 obitelji uređaja), (simuliranog) senzora temperature i vlage, svjetleće diode kojom upravljamo te *Wi-Fi provisioning* postupka koji putem *BLE* (engl. *Bluetooth Low Energy*) protokola korisniku omogućuje povezivanje *IoT* uređaja na *Wi-Fi* mrežu.

Cloud dio sustava bazira se na *AWS IoT* platformi te raznim servisima dostupnim putem platforme. Zadužen je za registraciju uređaja, njihovo upravljanje, primanje podataka preko *MQTT* i *HTTPS* protokola, pohranjivanje senzorskih podataka u *cloud* bazu podataka te posluživanje podataka *web*-aplikaciji.

Web dio sustava organiziran je na *backend* (serverski poslužitelj) i *frontend* (korisničko sučelje) dio. Serverski poslužitelj zadužen je za dohvaćanje senzorskih podataka iz *cloud* baze podataka i zadnjeg poznatog (prijavljenog) stanja svjetleće diode na uređaju, slanje novog željenog stanja svjetleće diode na zahtjev korisnika te posluživanje *frontend* dijela aplikacije. Korisničko sučelje dohvaća podatke od serverskog poslužitelja te korisniku prikazuje stanje svjetleće diode na uređaju, kontrole za upravljanje stanjem diode i graf sa senzorskim vrijednostima.



Slika 2.1 Dijagram cjelokupnog sustava

3. Pregled komponenti i tehnologija sustava

Osnova ugradbene strane ovoga sustava je mikrokontroler iz obitelji ESP32 uređaja. U okviru sustava, implementirana je podrška za ESP-WROOM-32 i ESP32-C3 modul te je sustav moguće izvoditi na oba navedena mikrokontrolera. Zbog svoje rasprostranjenosti i popularnosti, kao referentni modul poslužit će ESP-WROOM-32 modul, a nakon toga će biti napravljena usporedba njega i ESP32-C3 modula (poglavlje 3.1.5 i 3.1.6).

3.1. ESP32 platforma

Espressif Systems je proizvođač uređaja ESP obitelji te je time postao jedan od glavnih aktera u domeni Interneta stvari. Kroz svoje odlično integrirane bežične *IoT* čipove niske potrošnje, iznimno visokih performansi i niskih cijena kao što su ESP8266 i ESP32, profilirali su se većini kao prvi izbor čipova u domeni Interneta stvari [1].

3.1.1. Arhitektura

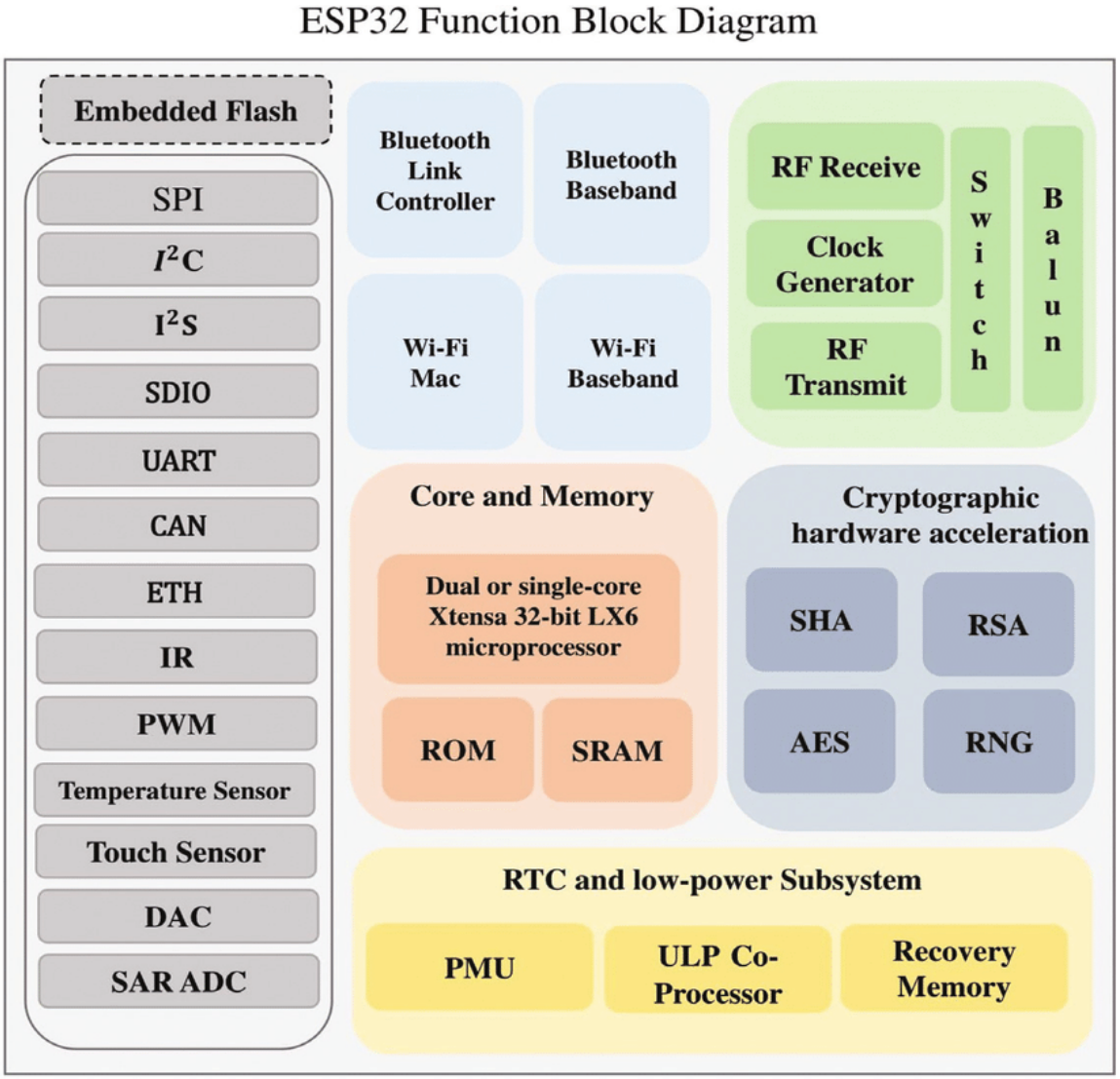
ESP32 je serija jeftinih *SoC* (engl. *System on Chip*) mikrokontrolera s integriranim *Wi-Fi* modulom i *Bluetooth*om s dva načina rada. Dok mnogi ostali mikrokontroleri koriste jednu jezgru, ESP32 ima implementiran dvojezgreni sustav s dva *Xtensa LX6* 32-bitna procesora harvardske arhitekture koji mogu raditi na 160 i 240 MHz. Njegova dvostruka jezgra omogućava istovremeno obavljanje više zadataka, čime se značajno poboljšavaju performanse i učinkovitost. Jedna jezgra (*„PRO_CPU“*) služi za pokretanje i inicijalizaciju cijelog sustava te izvođenje *Bluetooth* i *Wi-Fi* protokola, a druga (*„APP_CPU“*) služi za izvođenje aplikacijskog koda. Doduše, namjena jezgri se može mijenjati [2].

Jedan od ključnih atributa ESP32 modula je i njegova memorija. S obzirom na to da kroz ranije spomenute zahtjeve sustava mikrokontroler mora moći koristiti *BLE* (engl. *Bluetooth Low Energy*) protokol, *Wi-Fi* te raznovrsne *cloud* servise, uređaj bi trebao imati dovoljno memorije kako bi sustav mogao funkcionirati. Sa svojih 520 KB *SRAM*-a, 448 KB ROM-a i dvije 8 KB *RTC* memorije, ovaj mikrokontroler pruža veliku fleksibilnost i slobodu prilikom pisanja aplikacija i rukovanju podatcima. Osim interne memorije, podržava i vanjsku flash memoriju do 16 MB. To mu dodatno omogućava pohranu većih programa i biblioteka potrebnih za naprednije *IoT* aplikacije.

Iako je u razvijenom sustavu korišten samo aktivni način rada koji zahtjeva 160-260 mA pri 3.3 V (kada su *Wi-Fi* i *Bluetooth* aktivni, potrošnja može narasti i do 790 mA), razni načini rada i optimiziranja potrošnje ESP32 čine vrlo pogodnim za primjene s niskom potrošnjom energije poput *IoT* uređaja, bežičnih senzora, pametnih poljoprivrednih proizvoda i ostalih sličnih primjena. Dostupni načini rada uključuju: aktivni način (radio modul aktivno radi), *modem-sleep* način (procesori rade, no *Wi-Fi* i *Bluetooth* su isključeni) te *light* i *deep-sleep* načini gdje ili jedan ili oba procesora rade u slabijem načinu rada [1].

Funkcijski blok dijagram ESP32 modula prikazan je na slici 3.1. Na dijagramu se može vidjeti kako je ESP32 opremljen naprednim modulom za bežičnu komunikaciju koji podržava *Wi-Fi* tehnologiju (802.11 b/g/n) koja radi u 2.4 GHz frekvencijskom pojasu te *TCP*/*IP* i *UDP* protokole kao i *SSL*/*TLS* (engl. *Transport Layer Security*) enkripciju. Značajno za razvijeni sustav, modul podržava i *MQTT* (engl. *Message Queuing Telemetry Transport*) koji omogućava slanje podataka na *cloud*-ov *MQTT* poslužitelj. ESP32 može podržati *BSS* (engl. *Basic Service Set*) *STA* (engl. *station*) i *SoftAP* (engl. *Access Point*) operacije, što znači da može služiti kao *Wi-Fi* stanica i biti spojen na internet ili server, a ujedno može služiti i kao pristupna točka (*SoftAP* način rada). Osim *Wi-Fi* tehnologije, modul za bežičnu komunikaciju uključuje i podršku za *Bluetooth* (v4.2 *BR/EDR* i *BLE* – koji dijeli radio s *Wi-Fi* modulom). *Bluetooth* v4.2 je savršen za aplikacije koje zahtijevaju visoku brzinu prijenosa podataka, dok je *BLE* (engl. *Bluetooth Low Energy*) optimiziran za iznimno nisku potrošnju energije (no s kraćim dometom od klasičnog *Bluetooth*a) te ga to čini savršenim izborom za uređaje s baterijskim napajanjem kao što su senzori i nosivi uređaji.

Osim navedenoga, na dijagramu su prikazane različite podržane periferije i sučelja poput *ADC, DAC, SPI, I2C, I2S, UART, CAN, ETH, PWM* i drugih. Sadrži i *GPIO* (engl. *General-purpose input/output*) sučelje s 34 izvoda koje omogućuje spajanje raznih senzora i aktuatora. Služeći se *GPIO* sučeljem, sustav upravlja svjetlećom diodom, ali i dobiva ulazni signal preko *„BOOT“* tipke (GPIO0 izvoda).



Slika 3.1 ESP32 funkcijski blok dijagram [3]

3.1.2. Sigurnost

Sigurnost *embedded* uređaja, pogotovo u *IoT*-u, jedno je od najvećih pitanja i bolnih točaka prilikom razvijanja aplikacija. Na sreću, sigurnost je integralni dio arhitekture ESP32 te se i svojim sigurnosnim značajkama ističe među konkurencijom. Posjeduje ugrađene enkripcijske module za *AES, SHA, RSA* i *ECC* koji omogućuju zaštitu podataka, autentikaciju i integritet komunikacijskih kanala. Pruža i podršku za sigurno podizanje i učitavanje sustava (*Secure Boot*), pohranu sigurnosnih ključeva te enkripciju *flash* memorije. *Secure Boot* značajka osigurava izvođenje isključivo provjerenog i ključem potpisanog programskog koda. Javni ključ za *Secure Boot* pohranjuje se na uređaju, a privatni ključ (kojim se potpisuje programski kod) drži se na sigurnome mjestu. Osim osiguravanja uređaja

*Secure Boot*om, uređaj se može osigurati i enkripcijom *flash* memorije. Time se osigurava od fizičkog iščitavanja *flash* memorije i pristupa podataka zapisanim unutar *flash*-a. Implementiranjem enkripcije *flash* memorije pri razvoju sustava, *bootloader*, *partition table i* svi dijelovi aplikacijskog koda postaju zaštićeni od fizičkog iščitavanja [4]. *Secure Boot* i enkripcija *flash* memorije mogu se koristiti zajedno kako bi se maksimalno osigurao sustav. U ovome radu, ne koristi se enkripcija *flash* memorije, no koristi se značajka slična *Secure Boot*u (pripada *AWS IoT* platformi) koja osigurava da se samo ispravno potpisani kod može izvoditi nakon *OTA (*engl. *Over-the-Air*) servisa.

3.1.3. Tablica particija

Sustav baziran na ESP32 modulu može sadržavati više različitih aplikacija i vrsta podataka. Za definiranje svih vrsta podataka i njihov raspored u memoriji koristimo tablicu particija (engl. *partition table*). Ona služi kao komponenta za organizaciju memorije, definirajući kako *flash* memorija treba biti podijeljena i korištena. Tablica particija daje skicu memorije te pritom određuje lokaciju, svrhu i veličinu raznih memorijskih segmenata. Sama tablica particija nalazi se na lokaciji 0x8000 u *flash* memoriji. Tipično je definirana u CSV formatu, a sastoji se od više unosa (redova) od kojih svaki ima: naziv particije, vrstu particije (aplikacija, podatci ili drugo), podvrstu particije i pomak u memoriji na kojoj lokaciji treba biti smještena particija.

Za jednostavnije projekte dovoljno je koristiti predefinirane tablice particija kao što su „Jedna aplikacija, bez *OTA*“ i „Tvornička aplikacija, dvije *OTA* definicije“. Za kompliciranije i zahtjevnije projekte potrebno je koristiti prilagođenu tablicu.

Primjer tablice za „Jedna aplikacije, bez *OTA*“ prikazan pomoću kôda 3.1:

```
# ESP-IDF Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
```

Kôd 3.1 - Tablica particija za slučaj „Jedna aplikacija, bez *OTA*“

Na lokaciji 0x10000 (64 KB) u *flash* memoriji nalazi se aplikacija s nazivom *„factory“*. Program za učitavanje glavnog programa (engl. *bootloader*) pokrenut će ovu aplikaciju. U tablici se također nalaze i dvije lokacije koje sadrže podatke: *„nvs“* i *„phy_init“*, a služe za pohranjivanje *NVS* (engl. *Non-volatile storage*) i *„PHY init“* podataka.

Polje *„Name“* pohranjuje nazive particija, a služi tome da razlikuje različite particije. Nije od posebne važnosti za ESP32 modul.

Polje „*Type“* je vrsta podatka koja se nalazi u toj particiji. Uobičajeno je i preporučljivo koristiti tipove *„app“* (aplikacija, 0x00) i *„data“* (podatci, 0x01).

Polje *„SubType“* predstavlja podvrstu particije.

- Ako je vrsta particije *„app“* (aplikacija), podvrsta može biti:
    - *„factory“* (0x00) je početna vrsta aplikacijske particije. *Bootloader* će pokrenuti ovu aplikaciju osim ako nema druga particija vrste *„data/ota“* u kojoj je navedeno koji *OTA* program pokrenuti. U slučaju da se želi smanjiti iskorištena *flash* memorija, ova particija se može zamijeniti za *„ota_0“* podtip.
    - *„ota_0“* do *„ota_15“* su namijenjeni *OTA* aplikacijama. Prilikom korištenja *OTA* servisa, aplikacija bi trebala imati najmanje dvije *OTA* particije.
    - „test“ je rezerviran podtip za postupak tvorničkog testiranja.
- Ako je vrsta particije *„data“*, podvrsta može biti:
    - *„ota“* u kojoj se pohranjuje informacija o željenoj i trenutno odabranoj *OTA* aplikaciji.
    - *„phy“* koja se koristi za pohranjivanje inicijalizacijskih podataka fizičkog sloja (*PHY*) *OSI* modela.
    - *„nvs“* koji služi za pohranu podataka u *NVS*.
    - *„nvs_keys“* gdje se pohranjuju *NVS* enkripcijski ključevi u slučaju da je *NVS* enkripcija uključena.

Polje *„Offset“* i *„Size“* služe za definiranje lokacije gdje se particija treba nalaziti u *flash* memoriji te njene veličine. Particije *„app“* vrste moraju biti smještene na lokacijama poravnatim s 0x10000 (64 K).

Polje *„Flags“* služi za zastavice. Trenutno je podržana samo jedna zastavica – *„encrypted“*. Ako je zastavica navedena, particija će se zaštititi enkripcijom podataka ako je i *flash* enkripcija omogućena [5].

Tablica particija za razvijeni sustav morala je biti prilagođena potrebama i zahtjevima sustava, uključujući dodavanje dvije *OTA* particije (u koje se mogu pohranjivati aplikacije), jedna *„otadata“* particija s informacijom o željenoj i trenutno korištenoj *OTA* aplikaciji te *„nvs“* particija koja sadrži podatke (*Wi-Fi* pristupne podatke, vjerodajnice za *AWS cloud* platformu, itd.) pohranjene u *NVS*-u. Prikazana je putem kôda 3.2.

```
# Name,   Type, SubType, Offset,  Size, Flags
# Note: if you change the phy_init or app partition offset,
make sure to change the offset in Kconfig.projbuild
nvs,      data, nvs,      ,        60K,
otadata,  data, ota,      ,        8K,
phy_init, data, phy,      ,        4K,
ota_0,    app,  ota_0,    ,        1952K,
ota_1,    app,  ota_1,    ,        1952K,
```

Kôd 3.2 - Tablica particija prilagođena razvijenom sustavu

3.1.4. Usporedba s drugim mikrokontrolerima

Uspoređujući ESP32 obitelj s uređajima drugih proizvođača koji također imaju podršku za *Wi-Fi* i *Bluetooth* (i *BLE*) tehnologiju, kao najveći konkurenti se ističu Arduino Nano 33 IoT (Arduino) i nRF52840 (*Nordic Semiconductors*) mikrokontroleri. U tablici 3.1 navedene su ključne značajke kao što je podrška za *Wi-Fi* i *Bluetooth*, memorija, potrošnja energije i cijena. Iako nRF52840 ima iznimno dobru potrošnju energije (pogotovo u modovima iznimno niske potrošnje), jasno se može zaključiti da ESP32 (obitelj uređaja) znatno prednjači po performansama, memoriji, podršci za *Wi-Fi* i *Bluetooth* te, najviše od svega, cijeni. Nijedan proizvođač osim *Espressif*-a ne nudi uređaj sa svim navedene funkcionalnostima za tako pristupačnu cijenu.

Tablica 3.1 Usporedba s drugim mikrokontrolerima

Značajke	ESP32	Arduino Nano 33 IoT	nRF52840
Mikrokontroler	Tensilica Xtensa LX6 (dvojezgreni)	ARM Cortex-M0+	ARM Cortex-M4
Brzina	Do 240 MHz	48 MHz	64 MHz
***Flash* memorija**	4 MB	256 KB	1 MB
SRAM	520 KB	32 KB	256 KB
Wi-Fi	802.11 b/g/n (2.4 GHz)	802.11 b/g/n (2.4 GHz)	Samo neki moduli imaju podržan *Wi-Fi*
Bluetooth	Bluetooth 4.2 & BLE	Bluetooth 4.2 & BLE	Bluetooth 5.0 & BLE
GPIO izvodi	Do 34	14 digitalnih, 8 analognih	Do 48
Potrošnja energije	Modovi niske potrošnje (*light* i *deep sleep)*	Modovi niske potrošnje (*sleep)*	Modovi iznimno niske potrošnje
Cijena	Vrlo jeftini (5€)	Jako skupi	Skupi

3.1.5. Usporedba uređaja ESP obitelji

S obzirom na to da niti jedan proizvođač nema toliko pristupačan uređaj sa zahtjevima za velikom memorijom, iznimnim performansama i podrškom za *Wi-Fi* i *Bluetooth* (*BLE*), ostaje jedino usporediti različite uređaje unutar ESP obitelji. Najvažniji mikrokontroleri u toj obitelji su: ESP-WROOM-32, ESP32-C3 i ESP32-S3.

U tablici 3.2 je prikazana usporedba najvažnijih značajki navedenih mikrokontrolera.

ESP-WROOM-32 modul je originalni i najčešće korišteni ESP32 uređaj. Pogodan je za veliki raspon *IoT* aplikacija koje zahtijevaju podršku za *Wi-Fi* i *Bluetooth*.

ESP32-C3 je najslabiji modul od navedenih, no to nadoknađuje svojom jeftinijom cijenom. Optimiziran je za aplikacije s manjim budžetom dok svejedno pruža sve bitne funkcionalnosti uz manju potrošnju energije i optimiziranu memoriju.

ESP32-S3 je najnaprednija verzija koja dolazi s dvojezgrenim procesorom novije generacije te uključuje podršku za neuronske mreže. To ga čini idealnim za aplikacije s potrebom za

umjetnom inteligencijom (engl. *Artificial Intelligence, AI*) i strojnim učenjem (na rubu). Također, pruža više periferija i sučelja uspoređujući s drugim modulima, ali posljedično dolazi uz malo veću cijenu.

Tablica 3.2 Usporedba mikrokontrolera ESP32 obitelji [6]

Značajke	**ESP-WROOM-32**	**ESP32-C3**	**ESP32-S3**
Mikrokontroler	Tensilica Xtensa LX6 (dvojezgreni)	RISC-V jednojezgreni	Tensilica Xtensa LX7 (dvojezgreni)
Brzina	Do 240 MHz	Do 160 MHz	Do 240 MHz
***Flash* memorija**	4 MB	4 MB	4 MB
SRAM	520 KB	400 KB	512 KB
Wi-Fi	802.11 b/g/n (2.4 GHz)	802.11 b/g/n (2.4 GHz)	802.11 b/g/n (2.4 GHz)
Bluetooth	Bluetooth 4.2 & BLE	Bluetooth 5.0 & BLE	Bluetooth 5.0 & BLE
***ADC* kanali**	18 (12-bitni)	6 (12-bitni)	20 (12-bitni)
***DAC* kanali**	2 (8-bitna)	Nema	2 (8-bitna)
GPIO izvodi	Do 34	22	Do 45
Potrošnja energije	Modovi niske potrošnje (*light* i *deep sleep)*	Modovi iznimno niske potrošnje	Modovi niske potrošnje (light i deep sleep)
Podrška za neuronske mreže	Nema	Nema	Ima
Cijena	Jeftin (5€)	Jeftiniji	Skuplji

Razvijeni sustav implementiran je za ESP32-C3 i ESP-WROOM-32 module, no vrlo lagano se implementira podrška za ostale mikrokontrolere u ESP obitelji.

3.1.6. Usporedba iskorištene memorije za ESP-WROOM-32 i ESP32-C3

Na slikama 3.2 i 3.4 prikazan je rezultat naredbe `idf.py size` za ESP-WROOM-32 i ESP32-C3 mikrokontrolere. Iz slike 3.2 može se vidjeti kako je postotak iskorištenog *IRAM* i *DRAM* prostora na ESP-WROOM-32 modulu iznimno visok. Do toga dolazi radi činjenice da je dostupna veličina *IRAM*-a približno jednaka svega 131 KB, a veličina *DRAM*-a 124.5 KB. Iako je u tablici 3.2 navedeno kako ESP-WROOM-32 ima dostupno 520 KB *SRAM*

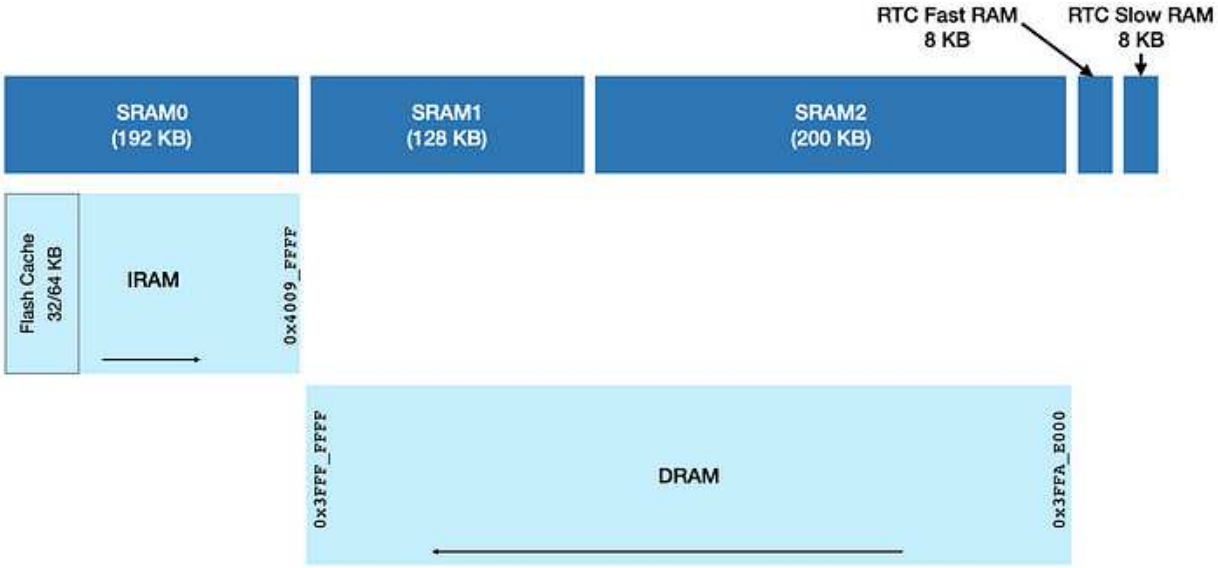
memorije, kroz sliku 3.2 može se zaključiti da nije sva memorija dostupna korisniku. ESP-WROOM-32 uistinu ima 520 KB dostupne *SRAM* memorije (od toga 320 KB dodijeljeno je *DRAM*-u, dok je ostalih 200 KB dodijeljeno *IRAM*-u), ali ona je dodatno smanjena za 64 KB u slučaju korištenja *Bluetooth* stoga (i 16 ili 32 KB za trag memorije, odnosno *trace memory*). Dodatno, zbog tehničkih ograničenja, najveća statički alocirana *DRAM* memorija iznosi 160 KB. Ostalih 160 KB može biti alocirano kao gomila (engl. *heap*) prilikom rada uređaja [7]. Memorijska mapa ESP-WROOM-32 modula prikazana je na slici 3.3.

Memory Type Usage Summary

Memory Type/Section	Used [bytes]	Used [%]	Remain [bytes]	Total [bytes]
Flash Code	920902	27.55	2421402	3342304
.text	920902	27.55		
Flash Data	258212	6.16	3936060	4194272
.rodata	257956	6.15		
.appdesc	256	0.01		
IRAM	126203	96.29	4869	131072
.text	125175	95.5		
.vectors	1027	0.78		
DRAM	95556	76.7	29024	124580
.bss	66768	53.59		
.data	28788	23.11		
RTC SLOW	24	0.29	8168	8192
.rtc_slow_reserved	24	0.29		

Total image size: 1334104 bytes (.bin may be padded larger)

Slika 3.2 Sažetak upotrebe tipova memorije za ESP-WROOM-32 modul



Slika 3.3 Memorijska mapa ESP-WROOM-32 modula [8]

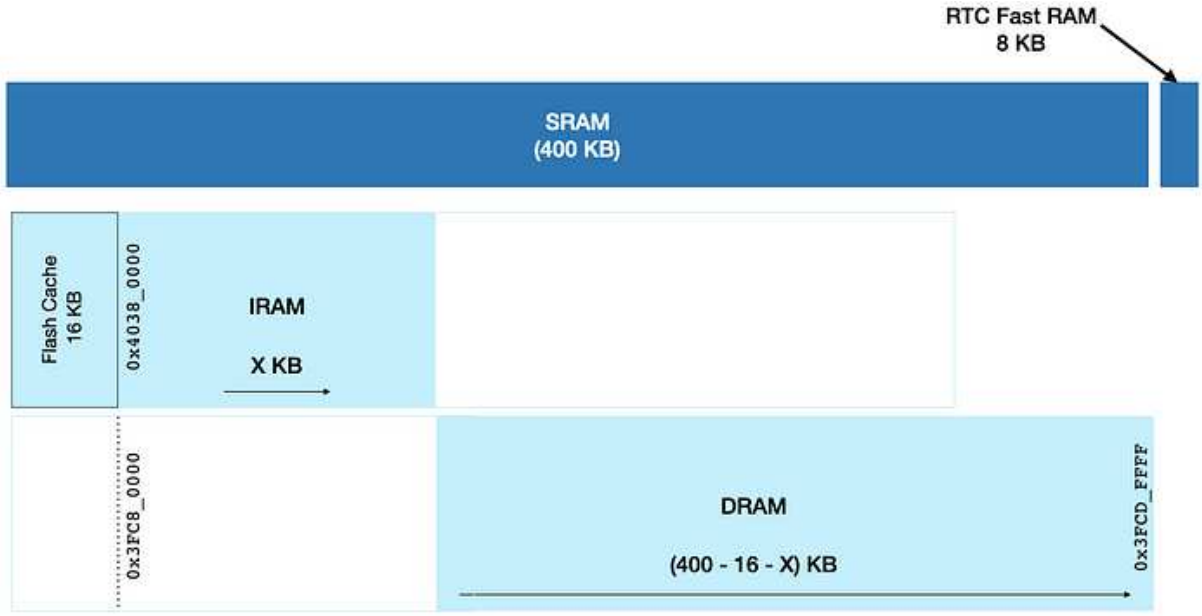
Memorijska mapa ESP32-C3 modula vidljiva je na slici 3.5, dok je sažetak upotrebe memorije dostupan na slici 3.4. Uz 400 KB *SRAM*-a, ESP32-C3 u nekim slučajevima uspijeva pružiti veću iskoristivost memorije od ESP-WROOM-32 modula koji sadrži 520 KB *SRAM*-a. To je ostvareno spajanjem *IRAM* i *DRAM* memorije u jednu particiju, širenjem memorije u jednome smjeru (za razliku od slike 3.3 gdje se *IRAM* i *DRAM* šire u suprotnim smjerovima), boljim upravljanjem *Bluetooth* memorijom te smanjenom upotrebom *IRAM*-a [8].

Memory Type Usage Summary

Memory Type/Section	Used [bytes]	Used [%]	Remain [bytes]	Total [bytes]
Flash Code	1047748	12.49	7340828	8388576
.text	1047748	12.49		
Flash Data	261804	3.12	8126772	8388576
.rodata	261548	3.12		
.appdesc	256	0.0		
DRAM	182192	56.71	139104	321296
.text	97798	30.44		
.bss	64560	20.09		
.data	19328	6.02		
RTC SLOW	536	6.54	7656	8192
.rtc_reserved	24	0.29		

Total image size: 1426678 bytes (.bin may be padded larger)

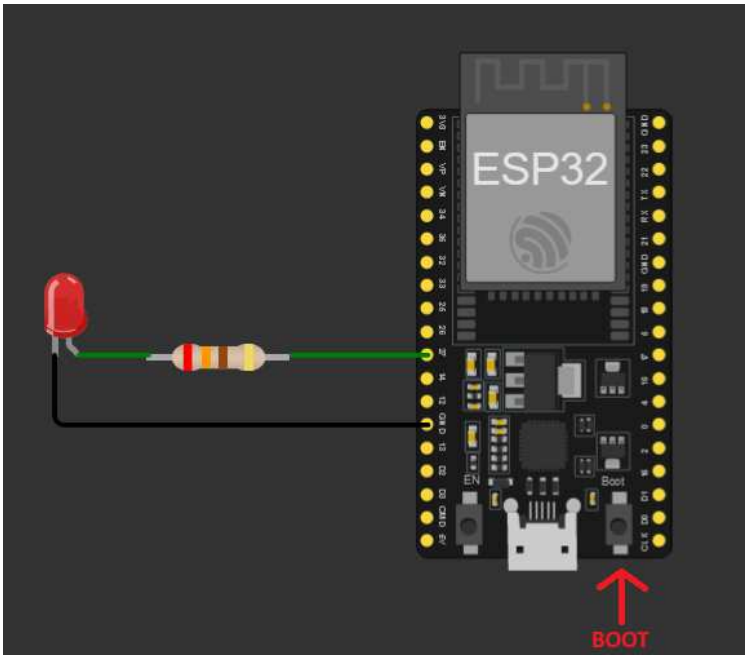
Slika 3.4 Sažetak upotrebe tipova memorije za ESP32-C3 modul



Slika 3.5 Memorijska mapa ESP32-C3 modula [8]

3.2. Senzori i aktuatori

Gruba skica u kojoj su naglašeni aktuatori u sustavu prikazana je na slici 3.6. Na slici je istaknuta svjetleća dioda u seriji s otpornikom te tipka *„BOOT“* koja je integrirana u razvojni sustav.



Slika 3.6 Skica aktuatora u razvijenom sustavu

S obzirom na to da na ESP-WROOM-32 razvojnome sustavu nema ugrađena svjetleća dioda, morali smo iskoristiti *GPIO* izvod kako bismo omogućili upravljanje svjetlećom diodom. Na anodu diode spojen je otpornik u seriju. Otpornik služi kako bi smanjio jakost struje kroz svjetleću diodu, a preporučljivo je da bude unutar vrijednosti od 230 do 500 oma. Svjetlećom diodom (spojenom na *GPIO27* izvod) upravljat ćemo korištenjem *web* aplikacije i tipke *„BOOT“*.

Na razvojnom sustavu može se pronaći tipka *„BOOT“*. Ona je spojena na *GPIO0* izvod te služi kako bi se mijenjalo stanje svjetleće diode ili brisali pohranjeni podatci iz memorije (pristupni podatci za *Wi-Fi* ili *AWS cloud* platformu) ovisno o duljini trajanja pritiska. Kratkim pritiskom tipke mijenja se stanje svjetleće diode, držanjem tipke 2 do 8 sekundi resetiraju se podatci za pristup *Wi-Fi* mreži te se pokreće *Wi-Fi provisioning*, a držanjem tipke dulje od 8 sekundi i puštanjem se brišu spremljene vjerodajnice za pristup *AWS IoT* platformi te se uređaj resetira.

Senzor temperature i vlažnosti zraka je simuliran. Simulacija je izvedena tako da iznos temperature počinje od 23 °C te se prilikom čitanja (dohvaćanja) povećava za 1 °C, smanjuje za 1 °C ili ostaje isti. Iznos vlažnosti počinje od 30 % te se svakim čitanjem (dohvaćanjem) povećava za 2 %, smanjuje za 2 % ili ostaje isti.

3.3. NVS (Non-Volatile Storage)

NVS, poznat i pod nazivom *NVM* (*Non-Volatile Memory*), vrsta je računalne memorije koja omogućuje trajno pohranjivanje podataka čak i nakon gubitka napajanja. *NVM* memorija uključuje memorije kao što su: *flash*, *SSD* (engl. *Solid State Drive*), *ROM* (engl. *Read-only memory*) te *EPROM* i *EEPROM*. ESP platforma nudi više vrsti *NVM*-a, uključujući *flash*, *eFuse* i *RTC* (engl. *Real-Time Clock*) memoriju. Glavna vrsta je *flash* memorija koja se koristi za pohranjivanje programske podrške (*firmware*a), aplikacija i različitih korisničkih podataka.

ESP biblioteka za pristup *NVS*-u je razvijena za pohranjivanje *key-value* (ključ-vrijednost) parova u *flash* memoriji. Lokacija u *flash* memoriji gdje se pohranjuju parovi su particije označene *„data“* vrstom i *„nvs“* podvrstom (kao što se može vidjeti u tablici particija prikazanoj pomoću kôda 3.2). U razvijenom sustavu se pomoću *NVS*-a pohranjuju podatci za pristup *Wi-Fi* mreži, *AWS cloud* platformi te status registracije uređaja na *cloud* platformi.

ESP platforma nudi podršku za zaštitu *NVS* podataka pomoću enkripcije. *NVS* je tako moguće zaštititi od fizičkog iščitavanja i mijenjanja, no nije ga moguće zaštititi od brisanja.

3.4. Wi-Fi provisioning

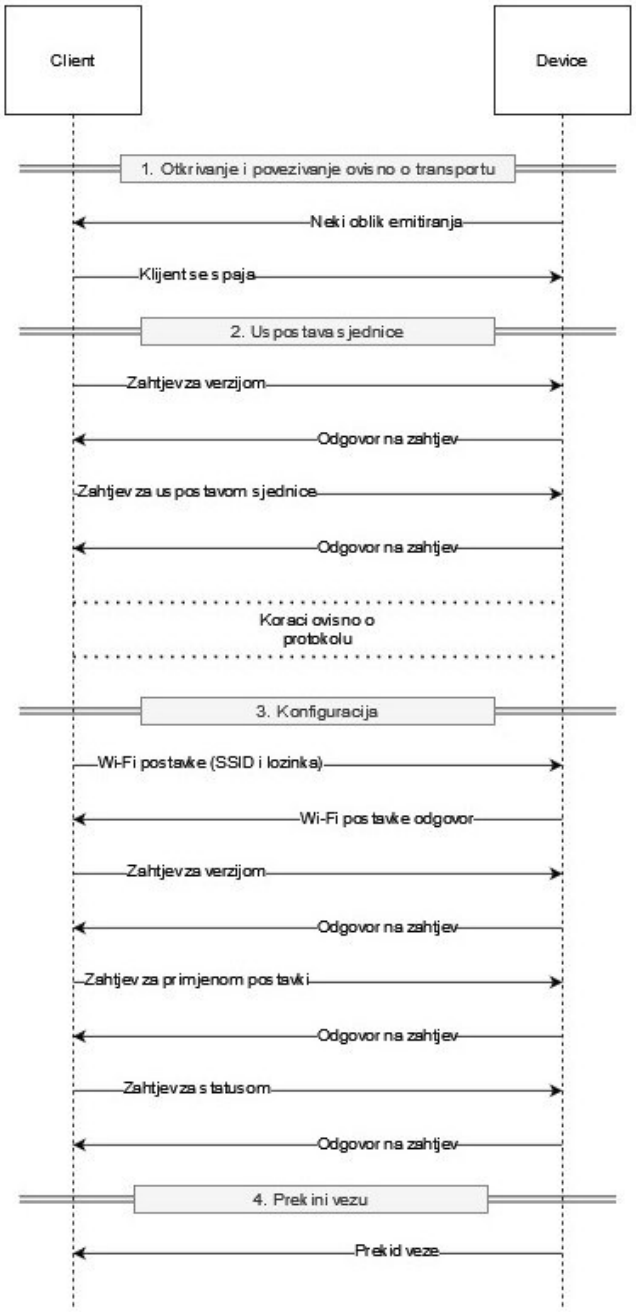
Wi-Fi provisioning je proces povezivanja novog *Wi-Fi* uređaja (stanice) s *Wi-Fi* mrežom. Taj proces uključuje prijenos informacija – naziva mreže (*SSID* mreže) i njenih sigurnosnih vjerodajnica (pristupnih informacija). Nakon gubitka napajanja potrebno je sačuvati te podatke kako se ne bi svaki puta pri paljenju uređaja morao ponavljati isti proces. Najčešći način spremanja tih podataka je korištenjem *NVS*-a (engl. *Non-Volatile Storage*), odnosno trajne memorije koja koristi poseban dio *flash* memorije čineći informacije dostupnima i nakon isključivanja uređaja ili gubitka napajanja [9].

3.4.1. Terminologija

- ***STA* Mode** (*Station Mode*) – jedan od 2 uobičajena moda uređaja. *Station Mode* (*STA*) je ono što ljudi smatraju normalnim modom za *Wi-Fi* uređaje. Uređaj koristi *Station Mode* kada se spaja na mrežu koja već postoji.
- ***AP Mode*** (*Access Point Mode*) – drugi način moda uređaja. U ovome slučaju, uređaj predstavlja pristupnu točku koja može biti spojena na internet i na koju se spajaju ostali uređaji (u *Station Mode*u).
- ***SSID*** (engl. *Service Set IDentefier*) – naziv mreže. Ako se otvori popis Wi-Fi mreža na laptopu ili mobitelu, vidjet će se popis *SSID*-ova. Ruter ili pristupna točka oglašavaju svoje *SSID*-ove kako bi ih obližnji uređaji mogli pronaći. Može sadržavati do 32 znaka.
- ***BSSID*** (engl. *Basic Service Set IDentefier*) – MAC fizička adresa pristupne točke ili bežičnog rutera koji se koristi za spajanje na *Wi-Fi*. 48-bitni niz brojeva koji slijedi konvenciju zapisa kao i MAC adresa.
- ***WPA/WPA2/WPA3*** (engl. *Wi-Fi Protected Access*) – protokol za dodatnu sigurnost mreže enkripcijom podataka. Svaki sljedeći je nadogradnja prošle verzije *WPA*. Najpoznatiji mehanizam je *WPA2-PSK* koji se zasniva na dijeljenoj šifri za pristup i najviše se koristi u kućanstvima. *WPA3* je nadogradnja *WPA2* te uključuje značajku individualizirane enkripcije podataka koji dopušta pristup novim uređajima i na nove načine osim unosa dijeljenje lozinke. *WPA3* koristi *Wi-Fi „Device Provisioning Protocol“ (DPP)* sistem koji omogućuje korisnicima korištenje *NFC*-a i *QR*-kodova kako bi pristupili mreži [9].

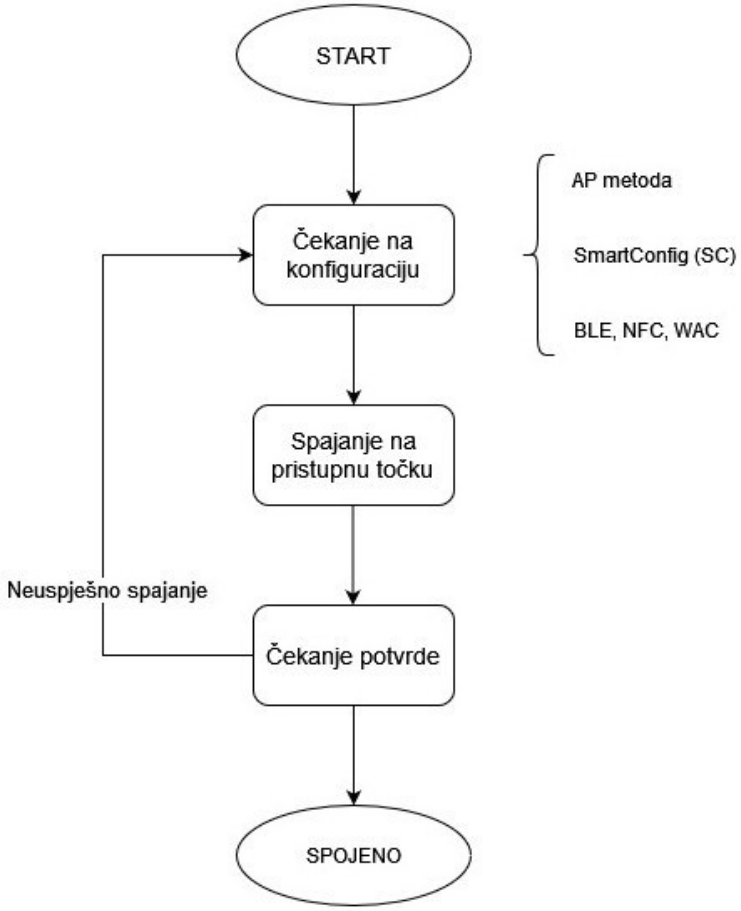
3.4.2. Proces povezivanja

Na slici 3.7 prikazan je uobičajeni proces povezivanja uređaja na *Wi-Fi* putem metode *Wi-Fi provisioning*-a. U prvom koraku uređaj koji se spaja emitira ovisno o metodi transporta koja se koristi (većinom *SoftAP* i *BLE* metoda) te se klijent zatim spaja na njega, predaje mu podatke za spajanje na mrežu te se na kraju veza prekida [9].



Slika 3.7 Proces povezivanja

Dijagram stanja pri povezivanju putem *Wi-Fi provisioning* metode prikazan je na slici 3.8. S dijagrama je vidljivo da nakon početka procesa uređaj čeka na konfiguraciju koja se najčešće dobiva metodama *SoftAP* + *HTTPS* i *BLE*. Nakon toga uređaj se pokušava spojiti na mrežu te prelazi u stanje čekanja na potvrdu. Ako je spajanje neuspješno, proces se ponavlja dok korisnik ne prekine proces ili se ne dogodi istek vremena (engl. *timeout*) [9].



Slika 3.8 Dijagram povezivanja

3.4.3. Sigurnost

Prilikom razvoja uređaja s mogućnošću *Wi-Fi provisioning*-a važno je učiniti proces jednostavnim i sigurnim. Također:

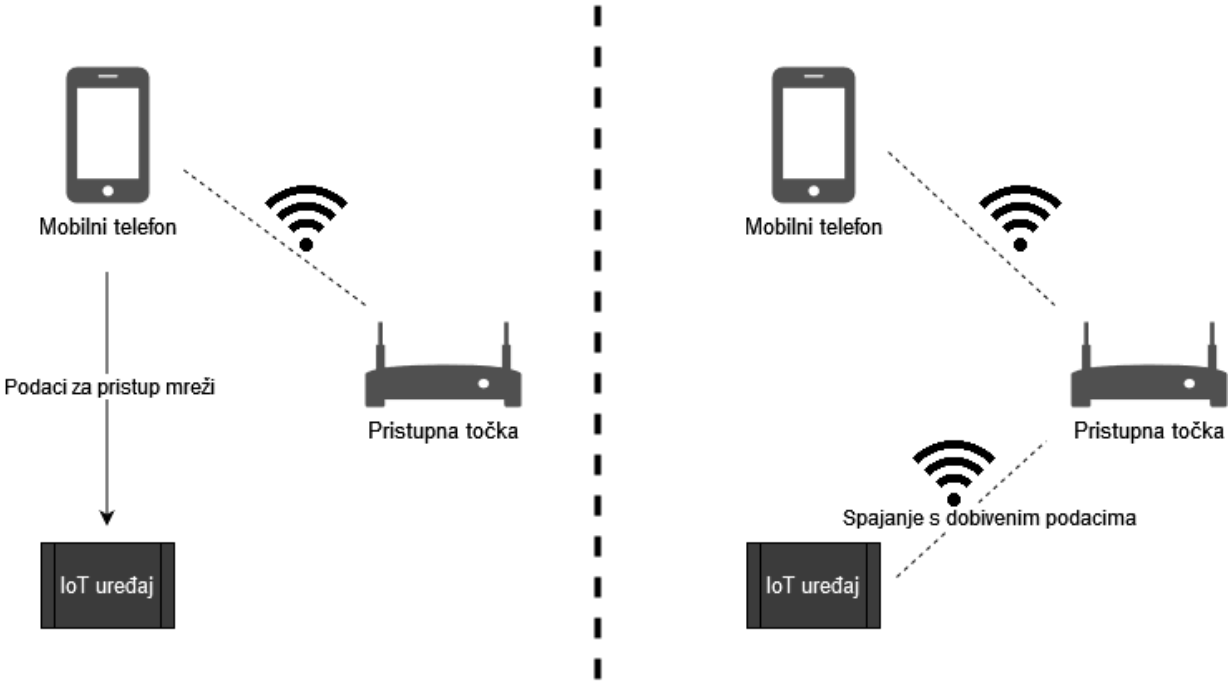
- Klijent bi trebao autenticirati uređaj s kojim je spojen.
- Konfiguracijski podaci između klijenta i uređaja moraju biti zaštićeni.
- Preporučljivo je korištenje *„*dokaza o posjedovanju*“* (engl. *Proof-of-Possesion*, *PoP*), jedinstvenog parametra koji za svaki uređaj osigurava da isključivo dozvoljeni korisnik ima ovlast konfigurirati uređaj.

„Dokaz o posjedovanju“ (*„PoP“*) moguće je ostvariti izvođenjem fizičke radnje na samome uređaju, poput pritiska posebne tipke ili unosa jedinstvenog nasumičnog ključa prikazanog na uređaju. Prilikom proizvodnje, svaki je uređaj moguće programirati jedinstvenim nasumičnim ključem koji kasnije može koristiti kao „dokaz o posjedovanju“ [9]. U

razvijenom sustavu, radi jednostavnosti, „dokaz o posjedovanju“ čini niz znakova „abcd1234“.

3.4.4. Metode prijenosa podataka

Dok postoji velik broj metoda prijenosa podataka tijekom *Wi-Fi provisioning*a, neki od kojih su: *WPS*-a (engl. *Wi-Fi Protected Setup*), *Smart Config*, *Wi-Fi Easy Connect*, u ovome odjeljku fokusirat ćemo se na dvije najpoznatije metode: *SoftAP* i *BLE*. Općeniti dijagram predaje podataka i spajanja na *Wi-Fi* pristupnu točku prikazan je na slici 3.9.

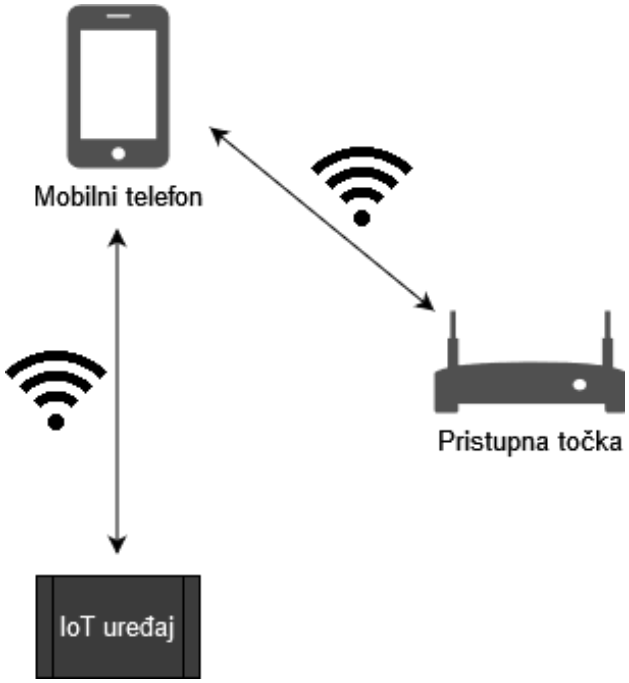


Slika 3.9 Općeniti dijagram predaje podataka i spajanja na *AP*

Prilikom korištenja *SoftAP* metode prijenosa podataka, *IoT* uređaj postaje pristupna točka na koju se korisnik sustava spaja te tim putem prenosi pristupne podatke za mrežu na *IoT* uređaj (slika 3.10).

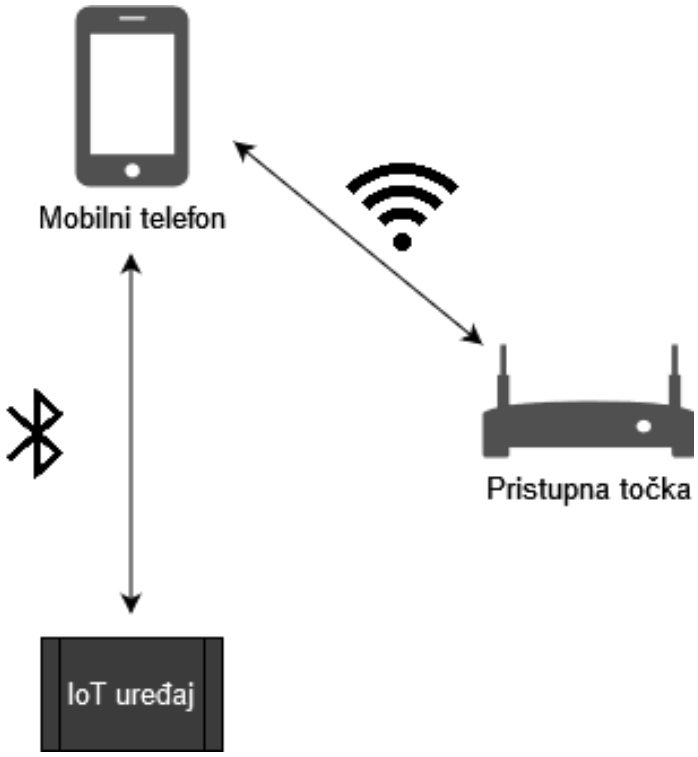
Koraci:

1. Spojiti se na privremenu pristupnu točku *IoT* uređaja.
2. Otvoriti sučelje koje pruža *IoT* uređaj.
3. Odabrati *Wi-Fi* mrežu za spajanje i unijeti podatke za spajanje.
4. Rekonfiguracija *IoT* uređaja u *Station (STA)* način rada te spajanje na *Wi-Fi mrežu* pomoću primljenih podataka za spajanje.



Slika 3.10 *SoftAP* metoda

Kod *BLE* metode prijenosa podataka, podatci se prenose putem *Bluetooth Low Energy* tehnologije koja se temelji na principu „oglašavanja“. Mobilni uređaji s *BLE* podrškom vidjet će obližnje „oglašavanje“ *IoT* uređaja, spojiti se na njega te mu novostvorenim kanalom prenijeti podatke za pristup mreži (slika 3.11). U slučaju korištenja *BLE* metode, uređaj mora implementirati i *Bluetooth* biblioteku koja utječe na smanjenje dostupne *flash* memorije te predstavlja jedan od nedostataka ove metode. Prilikom samog procesa *provisioning*a, *BLE* modul će također zauzimati i značajan dio *RAM* memorije. Srećom, nakon završetka procesa *Wi-Fi provisioning*a, sva zauzeta *RAM* memorija se može osloboditi.



Slika 3.11 *BLE* metoda

Uspoređujući *SoftAP* i *BLE* metodu, zaključujemo:

- *BLE* metoda ima prednost pri održavanju nesmetanog komunikacijskog kanala između uređaja, što osigurava pouzdan proces.
- *BLE* metoda ima bolje korisničko iskustvo (prilikom korištenja već gotovih aplikacija za *iOS* i *Android*). Korisnik ne mora mijenjati aplikacije tijekom procesa.
- *BLE* koristi dosta memorije (*flash* i *RAM*), dok *SoftAP* metoda ne. Dio memorije (*RAM*) moguće je osloboditi nakon završetka *BLE* metode.
- *SoftAP* metoda je vrlo interoperabilna. Međutim, korisnik se mora odspojiti od trenutne mreže kako bi se spojio na privremenu mrežu *IoT* uređaja [9].

U razvijenome sustavu implementirana je *BLE* metoda prijenosa podataka.

3.5. *HTTP* protokol

HTTP (engl. *Hypertext Transfer Protocol*) je temeljni protokol za prijenos informacija na *World Wide Web*u. Osnovna namjena ovog protokola je omogućavanje objavljivanja i prezentacije *HTML* dokumenata, odnosno *web* stranica. Zasnovan je na modelu klijent-poslužitelj, gdje klijent šalje zahtjeve, a poslužitelj odgovara traženim resursima ili

informacijom o statusu operacije. Osnovni elementi *HTTP* zahtjeva uključuju metodu, *URI* (engl. *Uniform Resource Identifier*), verziju korištenog protokola, opcionalna zaglavlja te tijelo zahtjeva [10].

3.5.1. *HTTP* metode

HTTP metode specificiraju željenu akciju nad resursom. Najčešće metode su:

- ***GET*** – dohvaća reprezentaciju resursa te ne mijenja stanje na poslužitelju
- ***POST/PUT*** – šalje podatke za stvaranje novog ili ažuriranje postojećeg resursa.
- ***DELETE*** – briše resurs.
- ***HEAD*** – Slično *GET* metodi, no vraća samo zaglavlje odgovora bez tijela odgovora.

3.5.2. Zaglavlja i tijelo

HTTP zaglavlja prenose metapodatke o zahtjevu ili odgovoru. Uključuju informacije poput vrste sadržaja (*Content-Type*), duljini sadržaja (*Content-Length*), podatcima o keširanju (*Cache-Control*) i autorizaciji (*Authorization*).

Tijelo *HTTP* zahtjeva ili odgovora obično sadržava podatke u formatu *HTML*, *JSON* ili *XML*, ovisno o tipu operacije i resursu.

3.5.3. Sigurnost

HTTP sam po sebi ne pruža sigurnost podataka pri prijenosu. Međutim, *HTTPS* (*HTTP Secure*) integrira *HTTP* s *TLS* (engl. *Transport Layer Security*) protokolom kako bi osigurao enkripciju, integritet podataka i autentikaciju. *HTTPS* postao je standardom zaštite povjerljivih informacija, kao što su lozinke, brojevi kreditnih kartica i privatne komunikacije.

3.6. *MQTT* protokol

MQTT (engl. *Message Queuing Telemetry Transport*) je „lagani“ (engl. *lightweight*) protokol za razmjenu poruka temeljen na objavi-pretplati (engl. *publish-subscribe*) modelu, osmišljen za uređaje s ograničenim resursima te za nisko propusne, visoko latentne i nestabilne mreže. Široko je prihvaćen i korišten u domeni *IoT* (engl. *Internet of Things*)

aplikacija omogućujući senzorima, aktuatorima i ostalim uređajima efikasnu komunikaciju [11].

3.6.1. Značajke *MQTT* protokola

Neke od najvažnijih značajki koje profiliraju *MQTT* protokol kao jedan od najvažnijih u *IoT* domeni su:

- **Iznimno „lagan“ protokol** – minimalni *overhead* i mala veličina paketa čini ga idealnim za *IoT* uređaje s ograničenim resursima (najmanja poruka je veličine 2 bajta). Poruke su jednostavne te se sastoje od zaglavlja (vrsta poruke, *QoS* razina i zastavice), teme (identificira kategoriju na koju se poruka odnosi) i tijela poruke (u bilo kojem formatu).
- **Pouzdan** – raznim *QoS* (engl. *Quality of Service*) razinama i dodatnim svojstvima za brže ponovno povezivanje osigurava pouzdanu isporuku poruka i u visoko latentnim i nestabilnim mrežama.
- **Siguran** – podržava *TLS/SSL* enkripciju i implementira moderne autentikacijske protokole (*OAuth, CMC,* itd.), pružajući dodatnu razinu sigurnosti pri isporuci poruka.
- **Skalabilan** – objavi-pretplati model odvajanjem pošiljatelja i primatelja omogućuje iznimnu skalabilnost.
- **Dobro podržan** – velik broj programskih jezika pruža izvrsnu programsku podršku za *MQTT* [11].

3.6.2. Princip rada

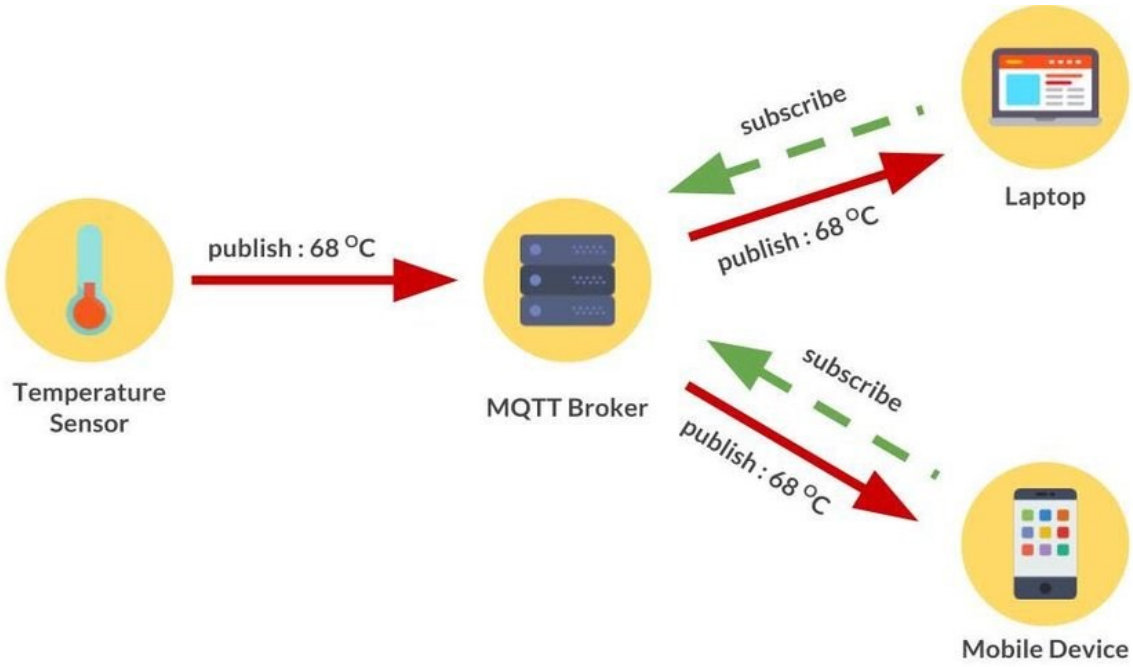
Osnovni elementi *MQTT*-a uključuju:

- **Klijent** – svaki uređaj ili aplikacija može djelovati kao klijent koji ima mogućnost objavljivati (slati, *publish*ati) i pretplaćivati se (*subscribe*) na teme kako bi primao poruke.
- ***MQTT* broker (posrednik)** – Centralni poslužitelj koji prima poruke od pošiljatelja (*publisher*) te ih prosljeđuje pretplatnicima (*subscribers*). Time omogućuje odvajanje pošiljatelja i primatelja te pruža skalabilnost i fleksibilnost sustava. Također je zadužen za zahtjeve pretplaćivanja, spajanja, odspajanja i slično.

- ***Teme*** – Kategorije pomoću kojih broker razdvaja i usmjerava poruke. Organizirani su hijerarhijski, slično *URL* putanjama.
- ***Kvaliteta usluge (engl.* Quality of Service, QoS)** – postoje tri razine usluge. Korištenjem *QoS* 0 razinu, poruka se šalje jednom. Kod *QoS* 1 razine, poruka se šalje barem jednom tako da broker mora potvrditi primitak poruke. Najviša razina, *QoS* 2, osigurava da se poruka primi točno jednom.

Objavi-pretplati model razlikuje se od modela klijent-server po tome što odvaja pošiljatelja poruke od primatelja poruke. Objavljivači i pretplatnici ne ostvaruju direktnu vezu, već je za usmjeravanje i dostavljanje poruka zadužen *MQTT* broker [11].

Dijagram na slici 3.12 prikazuje proces u modelu objavi-pretplati.



Slika 3.12 Model objavi-pretplati [12]

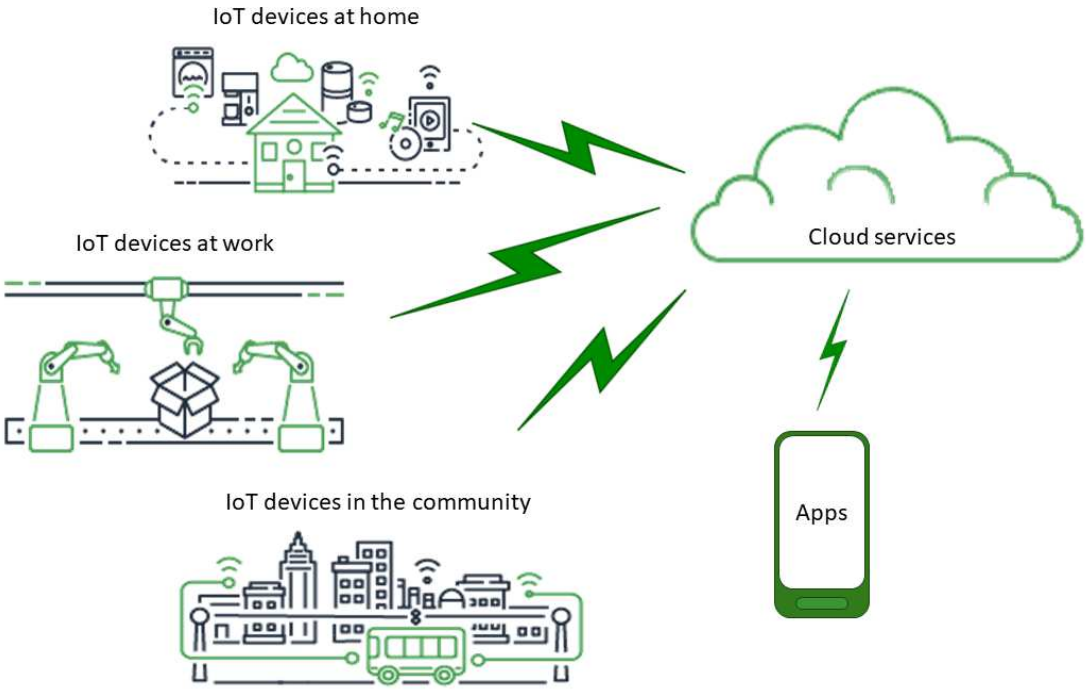
Prvi korak procesa, koji nije vidljiv na dijagramu, čini uspostavljanje veze između *MQTT* klijenta i brokera. Nakon toga, klijent ima mogućnost objavljivati ili pretplatiti se na neku od tema. Klijenti objavljivanjem šalju poruku brokeru, dok pretplaćivanjem na temu iskazuju zanimanje za primanjem poruka. U primjeru sa slike 3.12, laptop i mobilni uređaj su klijenti koji su se pretplatili na određenu temu (npr. „sensor/temperature“). Nakon toga, temperaturni senzor objavljuje poruku sadržaja „68 °C“ na navedenu temu. Broker prima poruku te ju zatim prosljeđuje svim klijentima pretplaćenim na temu (u ovome slučaju broker poruku prosljeđuje laptopu i mobilnom uređaju, koji naposljetku primaju poruku sadržaja „68 °C“).

4. *AWS IoT* platforma

4.1. Uvod

Uobičajeno, Internet stvari (*IoT*) sastoji se od ključnih komponenti prikazanih slikom 4.1:

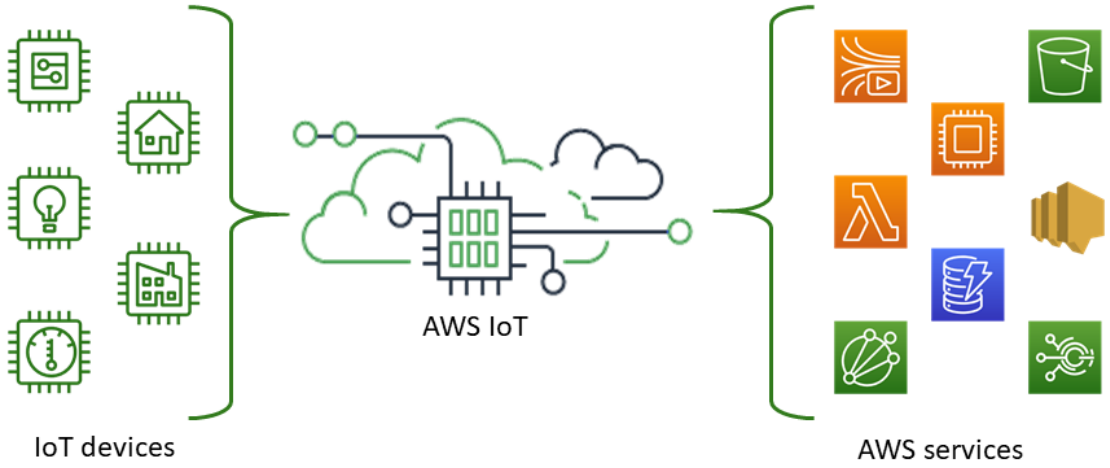
- **Sučelja** – komponenta koja povezuje uređaj s fizičkim svijetom. Neka od kojih su: korisnička sučelja, senzori, aktuatori.
- **Uređaji** – upravljaju sučeljima i komunikacijama te su uobičajeno u blizini stvari koje nadziru i kontroliraju. Njih čine razni mikrokontroleri, memorije i procesori.
- **Komunikacije** – preko kojih uređaji komuniciraju s *cloud* servisima. Kao na primjer: *Wi-Fi,* mobilni podatci (širokopojasni i uskopojasni), *LoraWAN* te druge *RF* komunikacije.
- ***Cloud* servisi** – distribuirani servisi za pohranu i obradu podataka te upravljanje uređaja.
- **Aplikacije** – pružaju korisnicima pristup i kontrolu nad spojenim *IoT* uređajima putem *cloud* servisa.



Slika 4.1 Komponente Interneta stvari [13]

4.2. Općenito

AWS IoT platforma je potpuno upravljan set servisa unutar *AWS* ekosustava koji omogućava povezivanje *IoT* uređaja s *cloud* aplikacijama ili drugim uređajima. Također, pruža alate za registraciju, organizaciju, učinkovito i sigurno upravljanje i praćenje *IoT* uređaja na jednostavan i skalabilan način te *MQTT* brokera. *AWS IoT* platforma je namijenjena za sigurno i skalabilno povezivanje milijarde uređaja, prikupljanje i analizu podataka u stvarnom vremenu te potpunu integraciju s ostalim *AWS* uslugama i servisima kao što su: *AWS Lambda, Amazon S3, Amazon DynamoDB* i tako dalje (slika 4.2) [14].



Slika 4.2 *AWS IoT* platforma [13]

4.2.1. Stvari („*Things*“)

„Stvar“ (engl. *thing*), u sklopu *AWS IoT* platforme, jest reprezentacija fizičkog uređaja ili logičkog entiteta (aplikacije) koji se spaja na *IoT* platformu. „Stvari“ se razlikuju po svome imenu te se spremaju u registar „stvari“ gdje se mogu organizirati u vrste i grupe.

Svaka „stvar“ može imati pripadajući certifikat za autentikaciju, politike pristupa (engl. *policy*) i svoj *shadow* (predstavljajući zadnje prijavljeno stanje uređaja).

4.2.2. Autentikacija uređaja

AWS IoT platforma podržava tri vrste autentikacije uređaja ili klijenta:

- **X.509 certifikati** – uobičajeno ih koriste *IoT* uređaji
- ***IAM* korisnici, grupe i uloge** – *web* i *desktop* aplikacije

- ***Amazon Cognito* identiteti** – mobilne aplikacije

Najvažniji prilikom razvoja *IoT* uređaja su upravo X.509 certifikati. To su digitalni dokumenti u X.509 standardu te su povezani sa „stvarima“ (engl. *things*) kako bi autenticirali uređaje pri spajanju na platformu. Preporučljivo je svakoj „stvari“ dati jedinstveni certifikat kako bi se olakšale radnje poput povlačenja (onemogućenja) certifikata.

AWS IoT podržava certifikate generirane preko *AWS IoT*-a, certifikate potpisane od registriranih certifikacijskih tijela (engl. *certificate authority, CA*) te certifikate od neregistriranih *CA*. Najčešće korištena opcija, pogotovo za demo sustave, jesu certifikati generirani preko *AWS IoT* platforme.

X.509 certifikati omogućuju korištenje asimetričnih (javnih i privatnih) ključeva koji pružaju sigurniju autentikaciju klijenta. Dobiveni ili generirani privatni ključevi uređaja trebali bi biti sigurno pohranjeni u uređaj i nikada dijeljeni. *AWS IoT* autenticira certifikate pomoću *TLS* protokola [13].

Pojednostavljen proces autentikacije uređaja obuhvaća:

1. Registraciju uređaja
    - prilikom registracije, uređaj dobiva certifikat koji sadrži javni ključ
    - uređaj dobiva i pripadajući privatni ključ
2. Sigurnu komunikaciju
    - prilikom spajanja na *AWS IoT,* uređaj koristi privatni ključ kako bi dokazao svoj identitet
    - *AWS IoT* platforma koristi javni ključ iz certifikata te potvrđuje ispravnost privatnog ključa klijenta
    - Komunikacija između *AWS IoT* platforme i uređaja može biti enkriptirana koristeći javni ključ
3. Autentikaciju poruka
    - prilikom slanja poruke s uređaja na *AWS IoT*, uređaj potpisuje poruku svojim privatnim ključem
    - *AWS IoT* koristi javni ključ prilikom potvrđivanja potpisa i izvora poruke

4.2.3. Politike pristupa (*„Policies“*)

Politike pristupa (engl. *policies*) *IoT* uređajima dodjeljuju i ograničavaju pristup resursima i uslugama platforme. Dodjeljuju se „stvarima“ i servisima, a definirane su u *JSON* formatu i omogućuju precizno definiranje radnji koje uređaji mogu obavljati te kojim resursima i uslugama *AWS IoT* platforme mogu pristupiti.

Primjer politike pristupa vidljiv je u kôdu 4.1.

```
{
"Version": "2012-10-17",
 "Statement": [
      {
      "Effect": "Allow",
      "Action": [
            "iot:Connect"
      ],
      "Resource": "arn:aws:iot:eu-central-
      1:accountID:client/${iot:Connection.Thing.ThingName}“
      },
      {
      "Effect": "Allow",
      "Action": [
            "iot:Publish",
            "iot:Receive"
      ],
      "Resource": [
            "arn:aws:iot:eu-central-1:
            accountID:topic/sensor/temperature",
            "arn:aws:iot:eu-central-1: accountID:topic/sensor/humidity"
      ]},
      {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
            "arn:aws:iot:eu-central-1: accountID:topicfilter/command/#",
      ]}
]}
```

Kôd 4.1 Primjer politike pristupa (*„policy“*-ja)

„Effect“ atribut unutar politike pristupa može biti *„Allow“* ili *„Deny“* te se njime dopušta ili ukida pristup definiranoj akciji nad zadanim resursom.

U primjeru *policy*ja, dopustili smo spajanje uređaja s atributom *„clientID“* koji se podudara s nazivom „stvari“ u registru „stvari“. Također, dopustili smo objavu i primanje poruka na temi (engl. *topic*) *„sensor/temperature“* i *„sensor/humidity“*. Na kraju, omogućili smo uređaju i pretplaćivanje na bilo koju temu koja počinje na *„command/“*.

4.2.4. Pravila (*„Rules“*)

Pravila (engl. *rules*) omogućuju uređajima automatizaciju akcija (putem *AWS* servisa) ovisno o događajima na *IoT* uređaju. Pravila su definirana korištenjem *SQL*-a, a njihove primjene mogu biti raznovrsne: filtriranje dolaznih podataka, spremanje podataka u bazu podataka i *Amazon S3*, slanje *push* obavijesti korisnicima putem *Amazon SNS* servise i sl. [13]

4.3. AWS IoT Fleet Provisioning

Prilikom ručne registracije uređaja, potrebno je na *IoT* uređaj spremiti jedinstveni privatni ključ i certifikat koji omogućuje uređaju autentikaciju te pristup *IoT* platformi i uslugama koje platforma nudi. Međutim, povećanjem broja uređaja ili prilikom razvoja uređaja velikih serija, takav pristup nije praktičan te se treba okrenuti drugim metodama. Među svojim servisima, *AWS IoT* platforma implementira *Fleet Provisioning*, servis koji nudi više raznih metoda s pomoću kojih se može osigurati jednostavna registracija i spremanje jedinstvenog klijentskog certifikata za masovan broja uređaja [13].

AWS IoT Fleet Provisioning usluga, također referencirana kao i *device provisioning* usluga, omogućuje sigurnu i automatiziranu registraciju i pridruživanje velikog broja *IoT* uređaja na *cloud* platformu te jednostavno skaliranje od *PoC*-a (engl. *Proof of Concept*) do masovne proizvodnje.

Fleet Provisioning prilikom automatizacije i registracija koristi sljedeće ključne elemente i pojmove:

- **Registracijski predložak (engl. *provisioning template*)** – opisuje pravila pristupa, postavke i resurse dodijeljene prilikom *provisioning*a uređaja. Također sadrži varijable koje omogućuju *provisioning* više uređaja koristeći jedan predložak.

- ***Provisioning Claim*** – privremeni certifikat koji uređaju služi kako bi se privremeno prijavio na *AWS IoT* dok se ne dobije trajni certifikat.
- ***Fleet Provisioning Policy*** - pravila koja kontroliraju pristup i akcije uređaja tijekom procesa *provisioninga*. Preporučljivo je maksimalno ograničiti pristup i akcije te omogućiti spajanje, objavljivanje i pretplaćivanje samo na strogo potrebne *MQTT* teme.

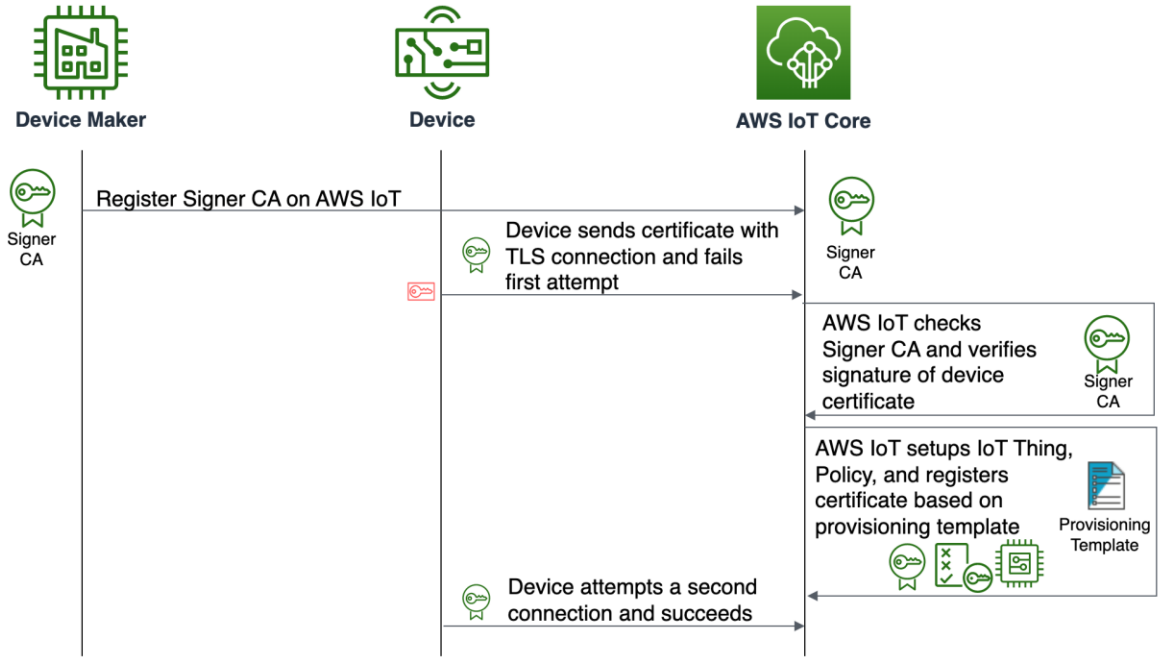
Moguće metode *provisioning*a uređaja razlikuju se ovisno o situaciji i dostupnim opcijama prilikom razvoja projekta:

- **Moguće je sigurno spremiti certifikate u *IoT* uređaj prije dostavljanja uređaja korisniku** – preporučljivo je koristiti *JITP* (engl. *just-in-time provisioning*) ili *JITR* (engl. *just-in-time registration*).
- **Korisnici ili ovlaštene osobe mogu pomoću aplikacije sigurno spremiti certifikate na *IoT* uređaje** – preporučljivo je koristiti registraciju putem provjerenog korisnika (engl. *provisioning by trusted user*).
- **Korisnici ne mogu koristiti aplikaciju za sigurno spremanje certifikata na *IoT* uređaj** – mogućnost je koristiti registraciju putem zahtjeva (engl. *provisioning by claim*)

4.3.1. *JITP* (*just-in-time provisioning*)

Uređaji prije korištenja *JITP* procesa moraju imati certifikate i privatne ključeve sigurno spremljene u memoriju. Certifikati moraju biti potpisani od strane ovlaštenog certifikacijskog tijela (engl. *certificate authority, CA*) koji je registriran na *AWS IoT*. Certifikati generirani od strane *AWS IoT*-a ne mogu biti korišteni za *JITP* i *JITR*.

Prilikom *JITP*-a uređaj se spaja na *AWS IoT Core*, a potpis njegovog certifikata se provjerava putem registriranih certifikacijskih tijela. Prilikom prvog spajanja uspostava *TLS* veze neće uspjeti jer certifikat još nije učitan u *AWS IoT* račun. Nakon provjere certifikata, registracijski predložak registrira „stvar“ i certifikat te uređaju dodjeljuje politiku pristupa. Uređaj mora imati implementiranu logiku za ponovno spajanje na *AWS IoT Core* kako bi nakon neuspješnog pokušaja ponovno pokušao uspostaviti vezu s *AWS*-om. Ako je proces *provisioning*a prošao uspješno, uređaj bi se trebao spojiti na *AWS IoT Core* [15]. Cijeli proces prikazan je dijagramom na slici 4.3.

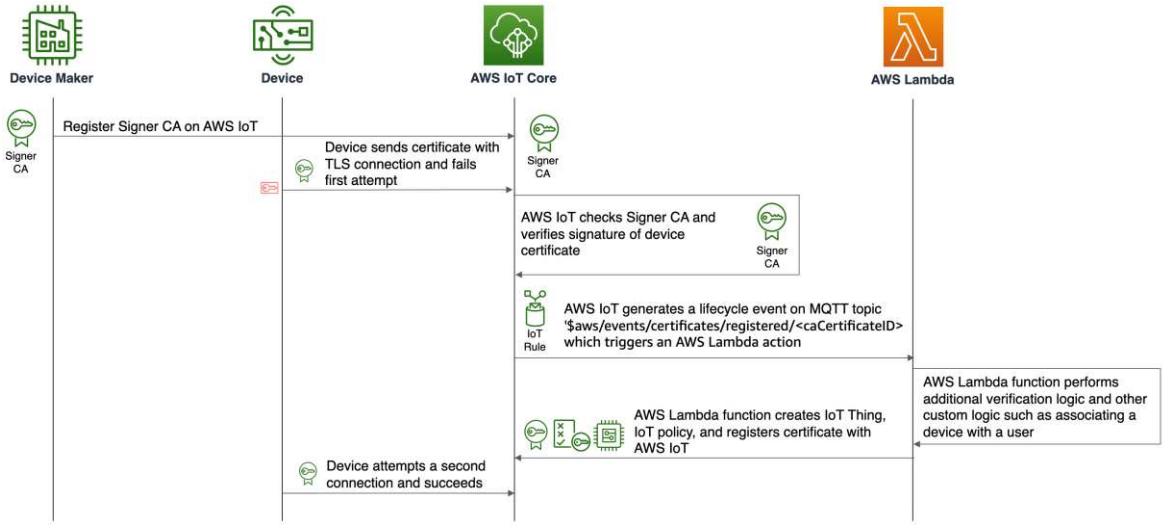


Slika 4.3 *JITP* (*just-in-time provisioning*) proces [15]

JITP omogućuje jednostavnost i automatizaciju prilikom registriranja uređaja bez potrebe za dodatnim koracima. Međutim, potrebno je osigurati pravilno registrirana certifikacijska tijela s registracijskim predloškom.

4.3.2. *JITR* (*just-in-time registration*)

JITR proces vrlo je sličan *JITP* procesu, ali pruža dodatnu fleksibilnost i implementaciju dodatne logike kroz *AWS Lambda* funkcije koje mogu obavljati dodatne provjere ili postupke prije konačne registracije uređaja. Prilikom pokušaja prvog spajanja na *AWS IoT*, na „`$aws/events/certificates/registered/<caCertificateID>`" MQTT temi generirat će se *„Lifecycle event"*, a zatim *AWS Lambda* funkcija ima zadaću obaviti dodatne provjere ili postupke, postaviti stanje certifikata u aktivno, kreirati „stvar", pridružiti politiku pristupa te oboje povezati certifikatom [15]. Proces *JITR* prikazan je slikom 4.4.



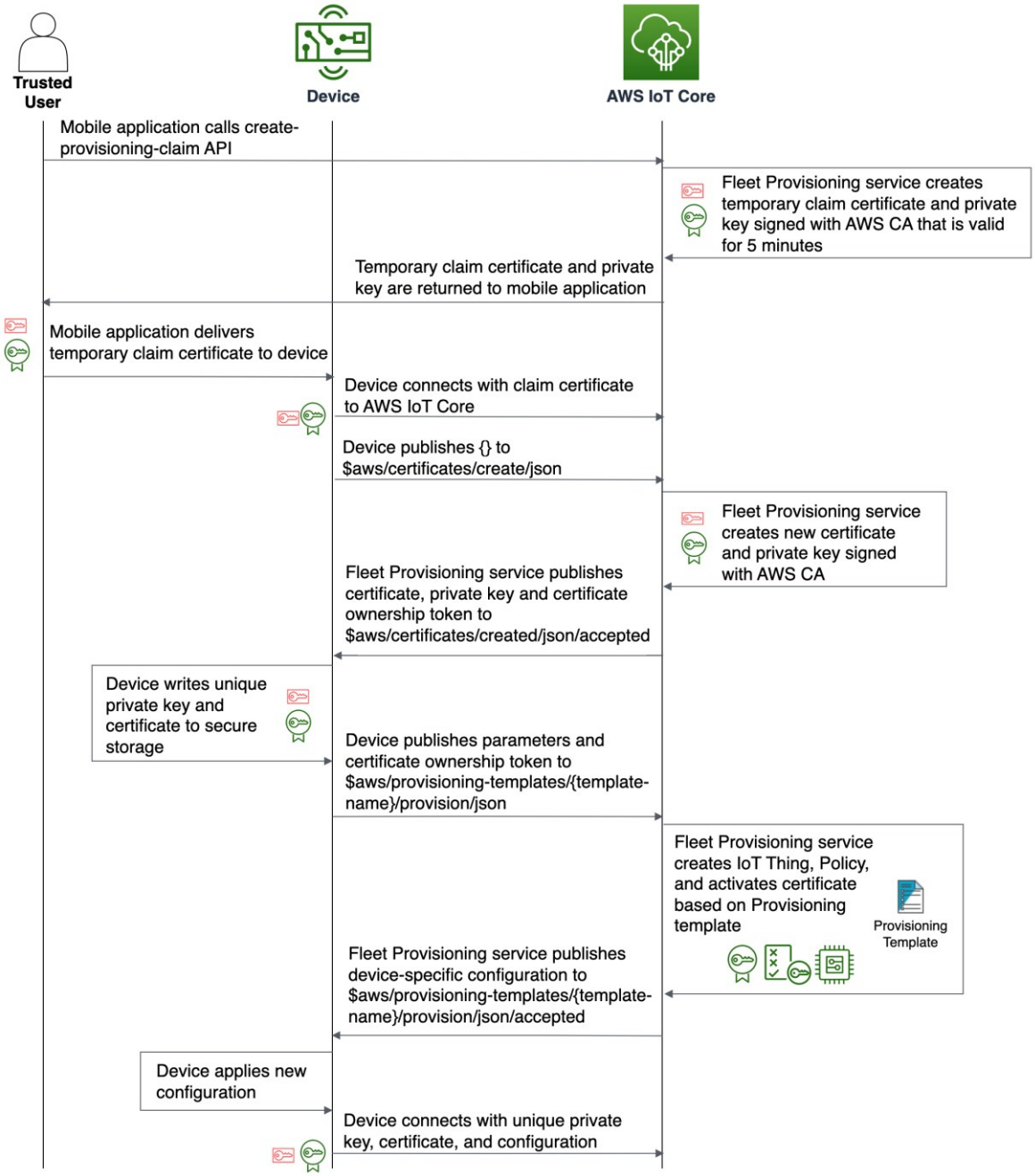
Slika 4.4 *JITR* (*just-in-time registration*) proces [15]

4.3.3. Registracija putem provjerenog korisnika (engl. *provisioning by trusted user*)

Postoje slučajevi gdje proces *provisioning*a jedinstvenih certifikata tijekom procesa proizvodnje nije moguć zbog tehničkih ograničenja, troška ili nekih drugi razloga. U takvim slučajevima jedan od mogućih procesa je *provisioning by trusted user*. Uređaj proizvodnju napušta bez jedinstvenih certifikata, a samo provjereni korisnici mogu *provision*ati uređaj jedinstvenim certifikatima. Primjena ovog procesa može se vidjeti u komercijalnim *IoT* uređajima koji dolaze s odgovarajućom mobilnom aplikacijom, a korištenjem aplikacije krajnji korisnik konfigurira uređaj putem *Wi-Fi* mreže te se tek tada uređaj po prvi puta spaja na *AWS IoT*.

AWS IoT Core pruža aplikacijsko korisničko sučelje (engl. *application programming interface*, *API*) koji omogućava mobilnim aplikacijama da pozivom sučelja `CreateProvisioningClaim` generiraju privremene certifikate i privatne ključeve koji vrijede pet minuta. Certifikat i privatni ključ se zatim prebacuju na uređaj putem mobilne aplikacije (korištenjem *Wi-Fi*-a, *BLE*-a, *USB*-a i sl.), uređaj se pomoću njih spaja na *AWS IoT* i mijenja privremene certifikate i ključ za jedinstveni X.509 certifikat s dužim trajanjem. Uređaj sigurno sprema dobiveni certifikat i pripadajući privatni ključ u svoju memoriju, a *AWS IoT* pomoću postavljenog *fleet provisioning* predloška registrira „stvar“, pripadajuću politiku pristupa i certifikat na *AWS račun*. *Fleet provisioning* predložak mora biti postavljen i održavan na *AWS IoT* platformi. Naposljetku, moguće je prije *provisioning*a dodati *AWS*

Lambda funkcije koje pružaju dodatni korak autentikacije prilikom procesa [15]. Dijagram *provisioning by claim user* prikazan je na slici 4.5.



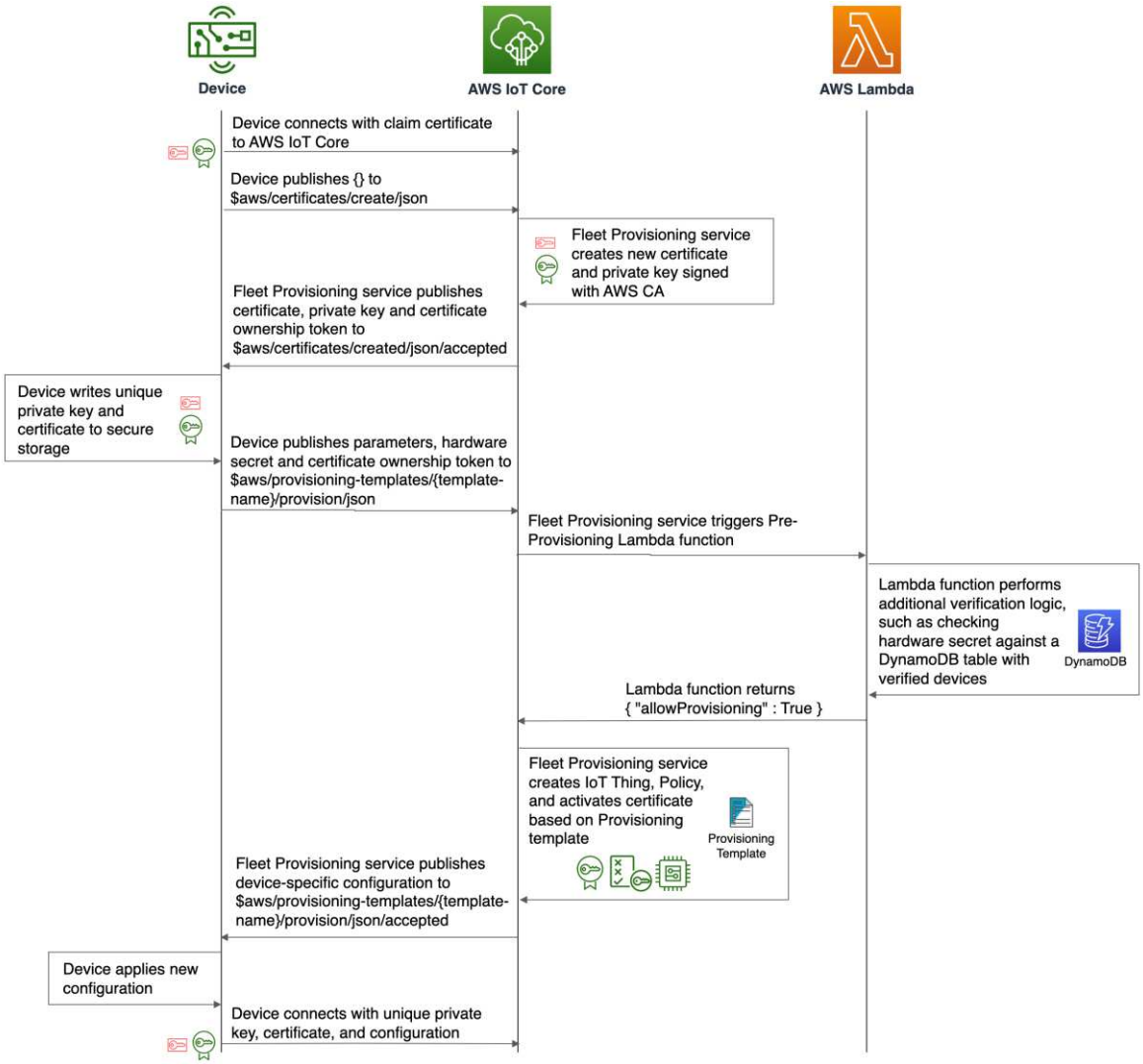
Slika 4.5 *Provisioning by claim user* proces [15]

4.3.4. Registracija putem zahtjeva (engl. *provisioning by claim*)

Registracija putem zahtjeva (*provisioning by claim*) je proces vrlo sličan registraciji putem provjerenog korisnika te se primjenjuje isto u situacijama kada proces *provisioning*a jedinstvenog certifikata nije moguć tijekom proizvodnje uređaja. Razlikuje se jedino po postupku dobivanja privremenih certifikata i privatnog ključa. Dok se privremeni certifikat

i ključ u procesu registracije putem provjerenog korisnika dobivaju korištenjem mobilne aplikacije, kod procesa *provisioning by claim* uređaji su proizvedeni s posebnim dijeljenim certifikatom (*claim certificate, bootstrap certificate*) sigurno pohranjenim u memoriji uređaja. Navedeni certifikat ima dopuštenje samo: spojiti se na *AWS IoT Core*, dokazati svoj identitet i zatražiti jedinstveni certifikat koji će zatim uređaju služiti za daljnje spajanje i komunikaciju s *AWS IoT*-om. Ako uređaj ima vlastiti privatni ključ, šalje se zahtjev za potpisom certifikata (engl. *Certificate Signing Request, CSR*) s *claim* certifikatom te ga *AWS IoT* zatim potpisuje [16]. Na slici 4.6 prikazan je *provisioning by claim* postupak. Dok je na slici prikazano korištenje *AWS Lambda* funkcije za dodatnu verifikaciju uređaja, ono nije nužno.

Claim certifikat trebao bi biti jedinstven po seriji uređaja kako bi se ograničio broj zahvaćenih uređaja u slučaju ugroženog privatnog ključa i povlačenja certifikata. Ako se *claim* certifikat povuče, uređaji koji dijele isti certifikat više neće biti u mogućnosti uspješno obaviti *provisioning* proces, no povlaćenje certifikata ne onesposobljava već registrirane uređaje [15].



Slika 4.6 *Provisioning by claim* proces [15]

4.4. AWS IoT Device Shadow

„Device Shadow“, *„shadow“* ili *„thing shadow“* predstavlja virtualnu kopiju stanja uređaja koja omogućava asinkronu komunikaciju između aplikacija i *IoT* uređaja. Time aplikacije mogu dohvatiti zadnje poznato stanje uređaja ili postaviti željeno stanje čak i kada uređaji nisu spojeni na *cloud*, a uređaji zatim svoje stanje sinkroniziraju prilikom ponovnog povezivanja na *cloud* platformu. *AWS IoT Device Shadow* je servis koji omogućava pohranu i dohvaćanje trenutnog stanja *IoT* uređaja te jednostavnije upravljanje uređajima u stvarnome vremenu. Svaki *shadow* ima rezervirane *MQTT* teme i *HTTP URL* koji podržavaju operacije dohvaćanja, ažuriranja i uklanjanja (*get, update* i *delete*) nad *shadow*ima.

*Shadow*i mogu biti kreirani, ažurirani i uklonjeni korištenjem *AWS IoT* konzole, *MQTT*-a i rezerviranih *MQTT* tema, *HTTP*-a i *REST API*-ja te *AWS* naredbenog sučelja. Nakon

kreiranja „stvari“, ne postoje njegovi odgovarajući *shadow*i sve dok se eksplicitno ne kreiraju. Integracija *Device Shadow*a s postojećim sustavima i aplikacijama je vrlo jednostavna s obzirom na to da se za komunikaciju koriste *MQTT* i *HTTP(S)* protokoli. U daljnjoj analizi *Device Shadow*a koristit će se pristup pomoću *MQTT* protokola jer većina *embedded IoT* uređaja upravo tako pristupa *shadow*ima [13].

4.4.1. *Named* i *unnamed shadow*

Servis podržava *named* (imenovane) i klasične, *unnamed* (bezimene) *shadow*e. „Stvar“ može imati više *named*, ali samo jedan *unnamed shadow* istovremeno. Pomoću imenovanih *shadow*a mogu se stvoriti različiti pogledi na stanja uređaja (različita svojstva logički grupirana po *shadow*-ima). S druge strane, korištenje bezimenih *shadow*a je jednostavnije i koristi se u rješenjima s manjim potrebama za *shadow*e, no dolazi s više ograničenja. *MQTT* teme za *named* i *unnamed shadow*e razlikuju se samo po prefiksu (nazvanim ***ShadowTopicPrefix***) prikazanim u tablici 4.1. Kako bi se stvorio prefiks potrebno je zamijeniti *„thingName“* s nazivom „stvari“, a *„shadowName“* s nazivom *shadow*a [13].

Tablica 4.1 Tablica prefiksa ovisno o vrsti *shadow*a

ShadowTopicPrefix	Vrsta *shadow*a
$aws/things/**thingName**/shadow	*Unnamed* (klasičan) *shadow*
$aws/things/**thingName**/shadow/name/**shadowName**	*Named* (imenovan) *shadow*

4.4.2. *Shadow* dokument

*Shadow*i koriste *Shadow* dokument u *JSON* formatu kako bi pohranjivali i dohvaćali podatke. *Shadow* dokument uobičajeno ima sljedeće odjeljke:

- ***state*** – odjeljak za pohranjivanje stanja uređaja, a može se sastojati od:
    - ***desired*** – željeno stanje uređaja. Aplikacije i drugi *cloud* servisi navode željena stanja svojstava uređaja ažurirajući *desired* odjeljak. U pravilu bi samo one trebale pisati u ovaj odjeljak.
    - ***reported*** – trenutno stanje uređaja. Uređaj prijavljuje svoje novo stanje ažurirajući *reported* odjeljak, a aplikacije pomoću ovog odjeljka dohvaćaju

zadnje poznato stanje uređaja. U pravilu bi samo uređaji trebali pisati u ovaj odjeljak.

- ***delta*** – virtualni tip koji sadrži razliku između *desired* i *reported* stanja uređaja, postavlja ga *AWS IoT*. Polja koja su u *desired*, a nisu u *reported* odjeljku, su uključena u deltu. Polja koja su u *reported*, ali nisu u *desired* odjeljku, nisu uključena u deltu.

- ***metadata*** – sastoji se od vremenskih oznaka (u *Epoch time*-u) svakog podatka iz *state* odjeljka, a služe kako bi se utvrdilo kada je koji podatak ažuriran.
- ***timestamp*** – informacija (vremenska oznaka) o tome kada je *AWS IoT* poslao poruku.
- ***clientToken*** – niz znakova jedinstven uređaju koji omogućuje povezivanja odgovora *AWS IoT*-a s poslanim zahtjevima.
- ***version*** – verzija dokumenta. Prilikom ažuriranja dokumenta, verzija se povećava se te tako osigurava da je verzija dokumenta koji se ažurira najnovija [13].

Primjer dokumenta prikazan je pomoću kôda 4.2, a u njemu se mogu vidjeti ranije spomenuti odjeljci.

```
{
  "state": {
    "desired": {
      "temperature": 22,
      "humidity": 50,
      "light": "on"
    },
    "reported": {
      "temperature": 21,
      "humidity": 48,
      "light": "off"
    },
    "delta": {
      "temperature": 1,
      "humidity": 2,
      "light": "on"
    }
  },
  "metadata": {
```

```
    "desired": {
      "temperature": {
        "timestamp": 1627384800
      },
      "humidity": {
        "timestamp": 1627384800
      },
      "light": {
        "timestamp": 1627384800
      }
    },
    "reported": {
      "temperature": {
        "timestamp": 1627384805
      },
      "humidity": {
        "timestamp": 1627384805
      },
      "light": {
        "timestamp": 1627384805
      }
    }
  },
  "clientToken": "137293",
  "version": 123,
  "timestamp": 1627384810
}
```

Kôd 4.2 Primjer *Shadow* dokumenta

4.4.3. Korištenje *shadow*-a

Device Shadow servis koristi posebne, rezervirane *MQTT* teme kojima omogućuje tri metode rada na *shadow*ima:

- ***update*** – kreira *shadow* ako ne postoji ili ažurira *state* odjeljak s podatcima iz poruke. Kada se promijeni stanje *shadow*a, *AWS IoT* šalje poruku na *„/delta“ MQTT* temu koja sadrži razliku između *desired* i *reported* odjeljka.
- ***get*** – dohvaća trenutni *shadow* dokument koji sadrži kompletno stanje uređaja.
- ***delete*** – uklanja *shadow* uređaja i njen sadržaj.

Shadow servis emulira zahtjev/odgovor model korištenjem *MQTT* modela objavi/pretplati. Svaka *shadow* metoda se sastoji od teme zahtjeva, a uspješan odgovor se prima preko *accepted* teme, dok se neuspješan odgovor prima preko *rejected* teme. *MQTT* teme za interakciju sa *shadow*ima su formirane kao prijašnje naveden ***ShadowTopicPrefix*** iza kojeg slijedi:

- ***/get*** – tema na koju se objavljivanjem prazne poruke dohvaća stanje uređaja (trenutni *shadow*).
- ***/get/accepted*** – tema na koju *AWS IoT* objavljuje *shadow* dokument s trenutnim stanjem uređaja.
- ***/get/rejected*** – tema na koju *AWS IoT* objavljuje dokument s opisom pogreške u slučaju da ne može dohvatiti *shadow* uređaja.
- ***/update*** – tema na koju se objavljuje djelomični *shadow* dokument sa *state* odjeljkom i *reported* ili *desired* stanjem. Primjer takvog dokumenta s novim željenim stanjem (*desired* odjeljkom) prikazan je kôdom 4.3.

```
{
  "state": {
    "desired": {
      "temperature": 21,
      "humidity": 46,
      "light": "on"
    }
  }
}
```

Kôd 4.3 Primjer dokumenta kojim se ažurira željeno stanje uređaja

- ***/update/accepted*** - tema na koju *AWS IoT* objavljuje poruku nakon prihvaćanja ažuriranja *shadow*a.
- ***/update/rejected*** - tema na koju *AWS IoT* objavljuje dokument s opisom pogreške u slučaju da *shadow* uređaja nije uspješno ažuriran.
- ***/update/delta*** – tema na koju *AWS IoT* objavljuje *shadow* dokument s *delta* odjeljkom u slučaju prihvaćanja ažuriranja *shadow*a te razlike između *desired* i *reported* odjeljka u *shadow* dokumentu.

- ***/update/documents*** – tema na koju *AWS IoT* objavljuje *shadow* dokument sa *state* odjeljkom kadgod je ažuriranje *shadow* dokumenta uspješno.
- ***/delete*** - tema na koju se objavljivanjem prazne poruke uklanja *shadow* dokument uređaja.
- ***/delete/accepted*** – tema na koju *AWS IoT* objavljuje poruku kada je *shadow* uređaja uspješno uklonjen.
- ***/delete/rejected*** – tema na koju *AWS IoT* objavljuje dokument s opisom pogreške u slučaju da *shadow* uređaja nije uspješno uklonjen.

Dok je uređaj spojen na *AWS IoT*, trebao bi primati poruke na *„**/update/delta**“* temu te usklađivati stanje uređaja tako da:

1. Obrađuje primljene poruke s *„**/update/delta**“* teme i sukladno tome usklađuje stanje uređaja.
2. Svaki puta kada se stanje uređaja promijeni, objavljuje poruku na *„**/update**“* temu sa *shadow* dokumentom koji u sebi sadrži *reported* odjeljak u kojem je zapisano trenutno stanje uređaja [13].

4.5. AWS IoT Jobs (OTA)

Daljinska operacija (engl. *remote operation*) je ažuriranje ili akcija koja se može izvoditi na fizičkom ili virtualnom uređaju bez potrebe za fizičkom prisutnosti operatora. Obavlja se koristeći *OTA* (*over-the-air*) ažuriranje. Služeći se *AWS IoT Jobs* servisom može se definirati set daljinskih operacija koji se šalju na izvođenje uređajima spojenima na *AWS IoT* platformu. Te daljinske operacije mogu obuhvaćati vraćanje uređaja na tvorničke postavke, resetiranje uređaja, zamjenu certifikata ili nadogradnju programske podrške putem daljinskog (*OTA*) ažuriranja. Zadatak (engl. *job*) je kreiran u *AWS* računalnom oblaku (putem *AWS* konzole, *API*-ja ili naredbenog retka) te poslan uređajima na izvođenje koristeći *MQTT* ili *HTTP* protokol [13].

4.5.1. Osnovni pojmovi

Osnovni pojmovi koje je potrebno poznavati prilikom korištenja *AWS IoT Jobs* servisa:

- ***Job* (zadatak)** – daljinska operacija koja se šalje na izvođenje uređajima spojenim na *AWS IoT.*
- ***Job document* (dokument zadatka)** – prije izrade zadatka mora se izraditi dokument zadatka. To je UTF-8 enkodirani *JSON* dokument koji opisuje željenu daljinsku operaciju te sadrži informacije potrebne uređaju da uspješno odradi zadatak. Dokument zadatka i u njemu navedene informacije (atributi) ovise o samoj implementaciji na *IoT* uređaju, no osnovni primjer je dan u kôdu 4.4. U njemu je navedena operacija i akcija (podoperacija) koju uređaj treba obaviti, u ovome slučaju je to nadogradnja programske potpore. Atribut *„firmwareUrl“* pokazuje na lokaciju odakle uređaj mora preuzeti novu programsku podršku.

```
{
  "jobDocument": {
    "operation": "update_firmware",
    "firmwareUrl": "https://your-bucket-
name.s3.amazonaws.com/firmware/esp32-firmware-v1.0.bin",
    "firmwareVersion": "1.0",
    "checksum": "5d41402abc4b2a76b9719d911017c592",  // MD5
checksum for integrity verification
    "action": "download_and_install"
  }
}
```

Kôd 4.4 Primjer dokumenta zadatka

- ***Target* (ciljani uređaji)** – lista uređaja ili grupa uređaja kojoj se šalje zahtjev za daljinskom operacijom.
- ***Deployment*** – definirani dokument zadatka i ciljani uređaji čine *deployment*. Dokument zadatka šalje se svim ciljanim uređajima.
- ***Job execution* (izvedba zadatka)** – instanca zadatka na ciljanom uređaju. Uređaj počinje izvedbu zadatka dohvaćajući dokument zadatka. Nakon toga, uređaj izvodi operacije navedene u dokumentu te tijekom izvođenja šalje izvještaj o stanju na *AWS IoT* platformu [13].

Postoje dvije vrste zadataka po načinu *deployment*-a:

- ***Snapshot job*** – zadatak se šalje svim uređajima koji su navedeni prilikom kreiranja zadatka. Nakon što oni izvijeste o stanju, zadatak je obavljen.

- ***Continuous job*** **(trajan zadatak)** – zadatak se također šalje svim uređajima koji su navedeni prilikom kreiranja posla, no zadatak se trajno nastavlja i šalje se svim uređajima koji su kasnije dodani u listu ciljanih uređaja [13].

4.5.2. Pojmovi prilikom konfiguriranja zadatka

Osnovni pojmovi koji pomažu prilikom konfiguriranja zadatka su:

- ***Rollouts*** – definira koliko uređaja primaju dokument zadatka svake minute.
- ***Scheduling*** – mehanizam koji omogućuje zakazivanje vremena *rollout*a dokumenta zadataka svim ciljanim uređajima.
- ***Abort*** – set uvjeta koji se moraju zadovoljiti kako se ne bi otkazali *rollout*i.
- ***Timeout*** – vremenski okvir u kojem uređaj mora poslati odgovor, inače se zadatak može označiti neuspješnim.
- ***Retry*** – mehanizam koji omogućava do 10 ponovnih izvođenja zadataka u slučaju isteka vremena (engl. *timeout*a), neuspješnog izvođenja, ili oboje [13].

4.5.3. Tijek rada uređaja

Uređaj može obavljati zadatke na dva načina:

- **Dohvaćanjem sljedećeg zadatka** (način korišten u razvijenom sustavu)
    1. Nakon spajanja na *AWS IoT*, uređaj se pretplaćuje na vlastitu *„notify-next“* temu.
    2. Pozivom `DescribeJobExecution` *MQTT API*-ja dohvaća se dokument zadatka i druge važne informacije.
    3. Pozivom `UpdateJobExecution` *MQTT API*-ja uređaj treba ažurirati svoj status (npr. *IN_PROGRESS*). Prošla i ova točka mogu se kombinirati u poziv `StartNextPendingJobExecution` funkcije.
    4. Uređaj izvodi dobiveni zadatak te pozivom `UpdateJobExecution` *MQTT API*-ja objavljuje napredak u izvođenju zadatka.

5. Uređaj pomoću `DescribeJobExecution` *MQTT API*-ja nadzire izvođenje zadatka u slučaju da je zadatak otkazan ili izbrisan te bi zbog tog slučaja trebao implementirati logiku nastavka normalnog rada.
6. Nakon obavljanja zadatka poziva se `UpdateJobExecution` *MQTT API* kako bi se javio status zadatka te uspješnost ili neuspješnost izvođenja spomenutog zadatka.
7. Na *„notify-next“* temu dolaze podatci za sljedeći zadatak, ako takav postoji.

- **Izbor između dostupnih zadataka**
  1. Nakon spajanja na *AWS IoT*, uređaj se pretplaćuje na vlastitu *„notify“* temu.
  2. Pozivom `GetPendingJobExecutions` *MQTT API*-ja dohvaća se lista dostupnih zadataka.
  3. Odabire se jedan od dohvaćenih zadataka, ako takav postoji.
  4. Izvođenje koraka 2.-6. iz prethodnog slučaja.
  5. Ako uređaj ostane spojen na *AWS IoT*, prima obavijest o svim dostupnim zadatcima.

Primjer poruke koju uređaji primaju na svojoj *„notify“ MQTT* temi može se vidjeti u kôdu 4.5. Kada je novi zadatak kreiran i poslan na izvođenje, *AWS IoT* servis objavljuje poruku na *„$aws/things/**thingName**/jobs/notify“* za svaki od ciljanih uređaja.

```
{
    "timestamp":1476214217017,
    "jobs":{
        "QUEUED":[{
            "jobId":"0001",
            "queuedAt":1476214216981,
            "lastUpdatedAt":1476214216981,
            "versionNumber" : 1
        }]
    }
}
```

Kôd 4.5 Poruka na *„notify“* temi

Nakon toga, uređaj dohvaća dokument zadatka i ostale informacije potrebne za izvođenje zadatka objavom poruke na *„$aws/things/**thingName**/jobs/**jobId**/get“ MQTT* temu. **JobId**

parametar može biti iz prethodne poruke ili može biti *„$next“,* čime se dohvaća sljedeći zadatak spreman za izvođenje.

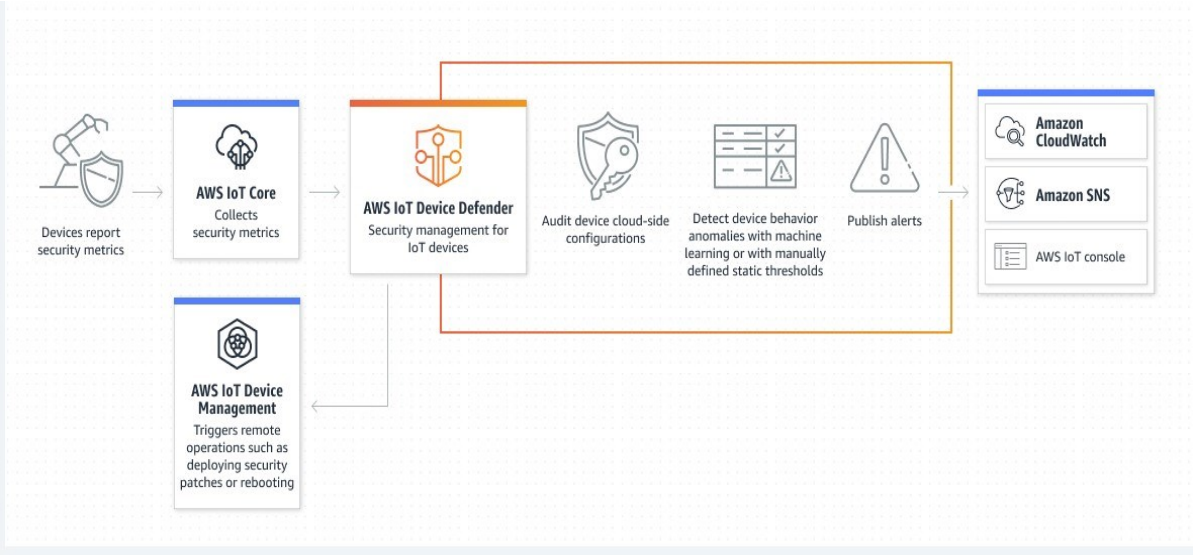
Ukratko, *AWS IoT* šalje ciljanim uređajima poruku kako bi ih informirao da je novi zadatak dostupan. Ciljani uređaji zatim započinju izvođenje zadataka dohvaćajući dokument zadatka, obavljanjem zadanih operacija te prijavom svog napretka *AWS IoT*-u. Kada se zadatak počne izvoditi, status zadatka prelazi u *„IN_PROGRESS“.* Prilikom izvođenja zadatka, uređaji inkrementalno prijavljuju napredak dok se zadatak ne obavi uspješno, neuspješno ili dok ne dođe do *timeout*a zadatka [13].

4.5.4. Sigurnost prilikom nadogradnje programske podrške

AWS IoT koristi *„Code Signing for AWS IoT“* servis kako bi automatski potpisao i time osigurao programe namijenjene uređajima. *„Code Signing for AWS IoT“* koristi certifikate i privatni ključ koji trebaju biti postavljeni u *AWS IoT*-u. Za potrebe testiranja taj certifikat ne mora biti potpisan od strane certifikacijskog tijela, no inače je preporučeno. Ako uređaj koristi uslugu potpisivanja programa, nakon preuzimanja programa mora potvrditi dobiveni potpis. Potpis se nalazi u dokumentu zadatka pod atributom *„codesign“* [13]. Mora se napomenuti da blokovi podataka novog programa dolaze na *„$aws/things/**thingName**/streams/**StreamId**/data/json“ MQTT* temu.

4.6. AWS IoT Device Defender

AWS IoT Device Defender je potpuno upravljan sigurnosni i nadzorni servis, a cilj servisa je pomoći korisnicima u prepoznavanju i reagiranju na potencijalne sigurnosne prijetnje uređaja spojenih na *AWS IoT* pružajući im dodatan sloj sigurnosti. Servis kontinuirano auditira konfiguracije uređaja kako bi osigurao kako njihove konfiguracije ne odstupaju od preporučenih i zadanih vrijednosti. Ako do toga dođe, servis ima mogućnost poslati obavijest korisniku putem drugih *AWS* servisa. Također, servis dopušta korisnicima nadzor sigurnosnih značajki na uređajima i *AWS IoT Core* servisu. Korisnici mogu definirati prigodno ponašanje i dopuštene sigurnosne značajke za svoje uređaje ili to mogu prepustiti strojnom učenju (engl. *machine learning, ML*) koji izrađuje model baziran na povijesnim podatcima uređaja. Ako se primijeti da uređaj može imati sigurnosni propust, servis ima mogućnost poslati alarm kako bi korisnici mogli izbjeći ili reagirati na problem (slika 4.7) [17].



Slika 4.7 *AWS IoT Device Defender* servis [17]

Korištenjem kombinacije metrika s *cloud* (*AWS IoT*) strane i metrika s uređaja mogu se detektirati:

- promjene u obrascima spajanja
- komunikacija uređaja s nepoznatim ili neautoriziranim krajnjim točkama
- promjene u obrascima odlaznog i dolaznog prometa

4.6.1. Značajke

Glavne značajke *Device Defender* servisa su:

- **Audit (revizija)** – provjerava se konfiguracija uređaja s preporučenim sigurnosnim praksama *AWS*-a kako bi se utvrdile stvari kao što su nedovoljno ograničavajuće politike pristupa. Auditi mogu biti pokrenuti na zahtjev ili se obavljati periodički. Razlikujemo i različite stupnjeve zabrinutosti zbog nađenih problema: kritična, visoka, srednja i niska.
- **Detekcija pravilima** – postavljanjem pravila specificira se normalno ponašanje uređaja. *Device Defender* nadzire i analizira svaki prijavljeni podatak s uređaja kako bi mogao detektirati anomaliju.
- **Detekcija strojnim učenjem** – automatskim generiranjem modela i svakodnevnim treniranjem, servis korištenjem strojnog učenja nadzire i analizira sigurnosne rizike i anomalije.

- **Upozorenja** – objavljivanje upozorenja (alarma) na druge *AWS* servise poput *AWS IoT* konzole, *Amazon CloudWatch*a te *Amazon SNS*-a.
- **Ublažavanje** – prilikom upozorenja mogu se koristiti ugrađene akcije za ublažavanje posljedica, a one mogu biti: dodavanje „stvari“ u grupe „stvari“, zamjena politike pristupa ili ažuriranja certifikata uređaja.
- **Sigurnosni profil** – definira set ponašanja. Svako ponašanje sadrži metriku koja precizira normalno ponašanje uređaja. Ponašanja se razdvajaju u dvije kategorije: detekcija pravilima i detekcija strojnim učenjem. Ponašanja u kategoriji detekcije pravilima definiraju normalno ponašanje pomoću ručno postavljenih statičnih pravila, dok se ponašanja u kategoriji detekcije strojnim učenjem definiraju korištenjem povijesnih podataka uređaja i modelom treniranim nad tim setom podataka. Pouzdanost modela strojnog učenja je iskazana u tri razine: visokoj, srednjoj i niskoj [17].

4.6.2. Detekcija strojnim učenjem (engl. *machine learning, ML*)

Detekcija strojnim učenjem se može koristiti u slučajevima kada je teško postaviti statična pravila za definiranje normalnog ponašanja uređaja. Jedan od takvih primjera je broj odspajanja s *AWS IoT* platforme, s obzirom na to da nije jasno koji je točan broj koji bi se smatrao prihvatljivim.

Dodatni primjer korištenja je nadzor nad ponašanjima koji se mijenjaju dinamički u ovisnosti o vremenu. Model strojnog učenja takva ponašanja uči periodički te trenira model s tim podatcima. Primjer toga je broj poslanih poruka prema *AWS IoT* platformi gdje uređaj može u određeno vrijeme slati veći broj poruka od uobičajenog broja.

Za prvotno treniranje modela strojnog učenja potrebno je 14 dana i minimalno 25 tisuća podatkovnih točaka po metrici za detekciju. Nakon toga, model se ponovno trenira svaki dan nad podatcima iz posljednjih 14 dana. Ako minimalni zahtjev od 25 tisuća podatkovnih točaka nije zadovoljen, model se pokušava izgraditi svaki naredni dan sljedećih 30 dana prije nego što se odustane od modela. Prikupiti 25 tisuća podatkovnih točaka tijekom 14 dana je ekvivalent sljedećim scenarijima:

- 60 uređaja s aktivnošću na *AWS IoT*-u svakih 45 minuta.
- 40 uređaja s aktivnošću na *AWS IoT*-u svakih 30 minuta.

- 15 uređaja s aktivnošću na *AWS IoT*-u svakih 10 minuta.
- 7 uređaja s aktivnošću na *AWS IoT*-u svakih 5 minuta [17].

4.6.3. Metrike (s uređaja i *cloud* strane)

Prilikom kreiranja sigurnosnog profila moguće je zadati normalno ponašanje uređaja putem metrike s uređaja i metrike s *cloud* (*AWS IoT*) strane. Pomoću tablice 4.2 prikazane su metrike koje je moguće koristiti, a pored njih je precizirano o čijoj se metrici radi te koje su podržane metode detekcije pojedine metrike.

Tablica 4.2 Tablica metrika [17]

Metrika	***Cloud/device – side* metrika**	**Podržane metode**
Broj poslanih bajtova	*Device*	Pravila + strojno učenje
Broj primljenih bajtova	*Device*	Pravila + strojno učenje
Broj poslanih paketa	*Device*	Pravila + strojno učenje
Broj primljenih paketa	*Device*	Pravila + strojno učenje
Odredišna *IP* adresa	*Device*	Pravila
Broj otvorenih *TCP* priključaka	*Device*	Pravila + strojno učenje
Broj otvorenih *UDP* priključaka	*Device*	Pravila + strojno učenje
Lista otvorenih *TCP* priključaka	*Device*	Pravila
Lista otvorenih *UDP* priključaka	*Device*	Pravila

Broj uspostavljenih *TCP* veza	*Device*	Pravila + strojno učenje
Veličina poruke	*Cloud*	Pravila + strojno učenje
Broj poslanih poruka	*Cloud*	Pravila + strojno učenje
Broj primljenih poruka	*Cloud*	Pravila + strojno učenje
Maksimalni broj neuspjelih autentikacijskih pokušaja	*Cloud*	Pravila + strojno učenje
Izvorišna *IP* adresa	*Cloud*	Pravila
Broj pokušaja spajanja na *AWS IoT*	*Cloud*	Pravila + strojno učenje
Broj odspajanja s *AWS IoT*-a	*Cloud*	Pravila + strojno učenje
Dužina trajanja nespojivosti uređaja	*Cloud*	Pravila

4.7. DynamoDB

DynamoDB baza podataka u razvijenom sustavu korištena je za pohranu primljenih podataka sa senzora temperature i vlažnosti zraka te njihovo dohvaćanje za potrebe prikaza podataka preko *web* aplikacije.

4.7.1. Opće informacije

Amazon DynamoDB je *AWS*-ov potpuno upravljan servis *NoSQL* baze podataka bez jedinstvenog poslužitelja (engl. *serverless*) koji pruža brze i predvidive performanse s neometanom skalabilnošću. Prikladan je za širok spektar primjena, uključujući mobilne pozadinske sustave, *web* aplikacije i Internet stvari (*IoT*) [18].

4.7.2. Arhitektura

DynamoDB koristi distribuiranu arhitekturu pri čemu se osigurava horizontalna skalabilnost i otpornost na kvarove. Podatci se pohranjuju u tablicama koje su zbirke stavki, pri čemu svaka stavka može sadržavati razne atribute. Primarni ključ jedinstveno identificira svaku stavku unutar tablice, a primarni ključ može se sastojati samo od particijskog ključa ili particijskog ključa i ključa za sortiranje.

- **Tablica** – glavna organizacijska jedinica za pohranu podataka.
- **Stavka** – zapis u tablici, sadrži atribute. Slična redovima u relacijskoj bazi podataka (*SQL*)
- **Atributi** – polja unutar stavke. Slični stupcima u relacijskoj bazi podataka.

4.7.3. Ključne značajke

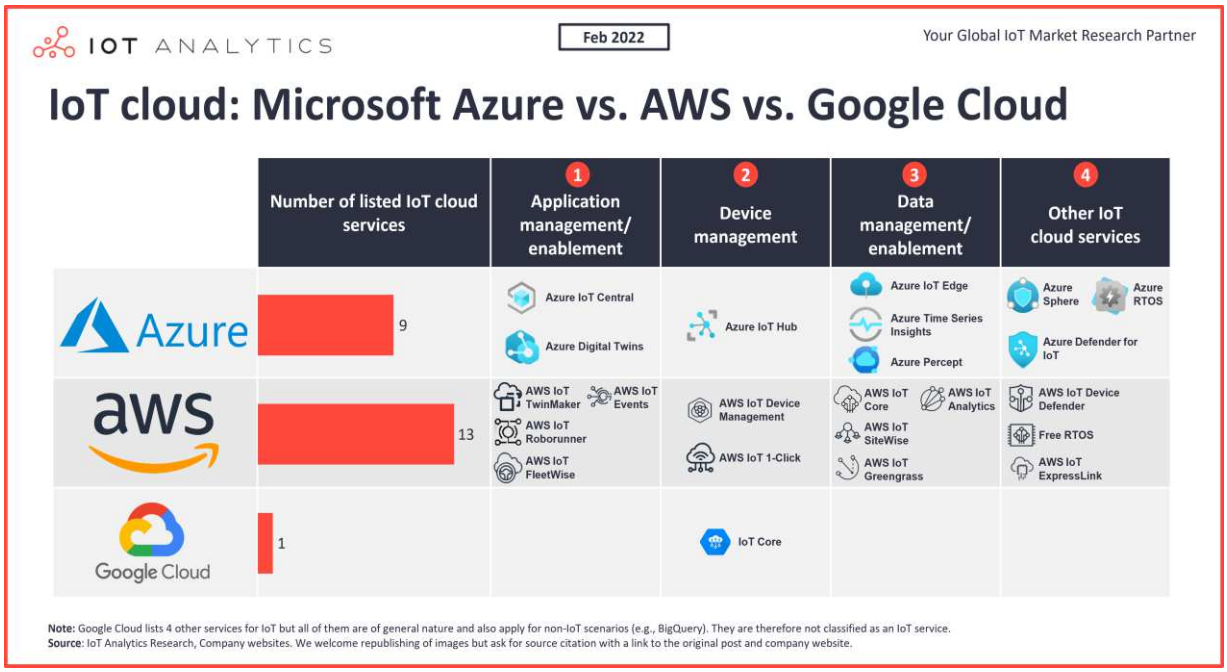
- **Skalabilnost** – *DynamoDB* baza je podataka bez jedinstvenog poslužitelja (engl. *serverless*) s distribuiranom arhitekturom koja automatski dijeli podatke na više poslužitelja omogućujući rukovanje velikim količinama podataka i visokim stopama zahtjeva bez ikakve potrebe za održavanjem. *DynamoDB* se automatski skalira ovisno o klijentovim potrebama.
- **Performanse** – s vremenom odgovora ispod deset milisekundi, optimiziran je za aplikacije sa zahtjevom visokih performansi. Brze operacije čitanja i pisanja postignute su korištenjem *SSD* memorije, učinkovitim indeksiranjem i slojem predmemoriranja u memoriji (*DAX*).
- **Visoka dostupnost i otpornost na kvarove** – repliciranjem podataka preko više dostupnih zona unutar *AWS* regija osigurana je visoka dostupnost i trajnost podataka, dok se u slučaju kvarova podatci automatski repliciraju.

- **Potpuno upravljan servis (engl. *fully managed*)** – *DynamoDB* brine o stvaranju, konfiguraciji, održavanju, dostupnosti, sigurnosti i nadzoru baze podataka omogućujući klijentima iznimnu jednostavnost korištenja [18].

4.8. Usporedba s ostalim *IoT cloud* platformama

Iz istraživanja tržišta 2022. godine tri najpopularnije *IoT cloud* platforme bile su u vlasništvu *AWS*-a, *Microsoft*a te *Google Cloud*a te su tada obuhvaćale više od 80 % tržišta. S obzirom na to da *Google Cloud* nikada nije bio potpuno posvećen *IoT* području, pogotovo ne kao konkurenti *AWS* i *Microsoft*, prestao je s podrškom za svoju *IoT* platformu. Njihova platforma (*Google IoT Core*) još je uvijek dostupna za korištenje, no s obzirom na oskudnost usluga koje nudi, nije najbolji izbor za razvoj pravih projekata. Tijekom zadnjih godina, pojavili su se novi konkurenti: *IBM Watson IoT Cloud*, *Cisco IoT Cloud Connect* i *Samsung SmartThings*, ali i platforme otvorenog koda (engl. *open-source*) kao što su *OpenMTC*, *FIWARE* i *SiteWhere*. Usprkos tomu, *AWS* i *Microsoft Azure* ostali su daleko ispred ostalih po svojem tržišnom udjelu i popularnosti. Stoga, usporedba će biti provedena između njih.

Na slici 4.8 mogu se vidjeti dostupni servisi za *AWS* i *Microsoft Azure IoT* platforme, a jasno je da *AWS* prednjači u tom segmentu sa svojih 13 *IoT* servisa naspram *Azure*ovih 9. Usprkos tomu, za dosta projekta obje platforme će zadovoljiti svojim dostupnim servisima i funkcionalnostima. Primjerice, za traženi sustav u sklopu ovoga rada, obje platforme nude svoje servise koji omogućuju razvoj takvog sustava. *AWS*-ov *Device Shadow* servis odgovara *Azure*ovom *Device Twins* servisu, *Fleet Provisioning* je sličan *Device Provisioning* servisu, a *Azure Device Update* nudi funkcionalnosti sukladne AWS *IoT Jobs (OTA)* servisu. Osim toga, obje platforme omogućavaju komunikaciju putem najpopularnijih protokola (*MQTT* i *HTTP*), nude biblioteke za velik broj programskih jezika (*C, Java, Python*, i tako dalje) te pružaju izvrsnu skalabilnost i sigurnost.



Slika 4.8 Usporedba dostupnih servisa [19]

Zbog svega toga, jedne od važnih stvari za uzeti u obzir prilikom izbora *IoT* platforme su mogućnosti integracije dostupnih servisa i usluga za potrebe projekta te cijena korištenja platformi.

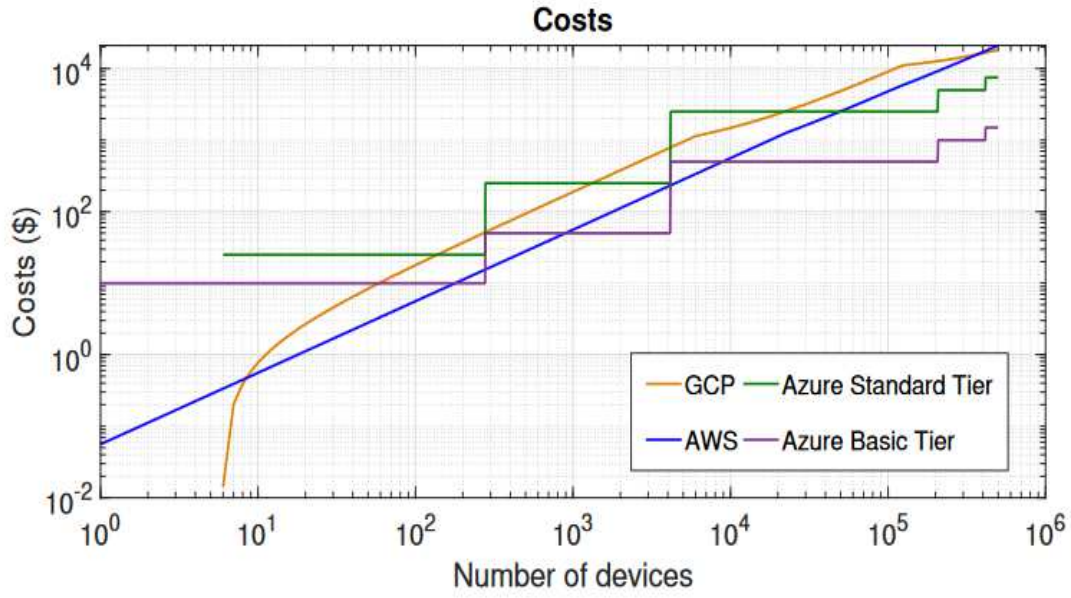
Microsoft Azure IoT ima veliku prednost kod velikih poduzeća zbog svoje integracije s ostalim *Microsoft* proizvodima: *Microsoft Windows, Microsoft 365 (Office)* i *PowerBI*. Najsnažniji fokus njihove platforme je na industrijskim aplikacijama i sustavima te imaju nadmoć kod proizvodnih i industrijskih projekata.

AWS IoT ističe se po svojim raznovrsnim i specijaliziranim servisima, a pruža odličnu integraciju s ostalim *AWS* uslugama kao što su *AWS Lambda, S3 Bucket* i alati za strojno učenje. *AWS* također prednjači i u domeni strojnog učenja i računarstva na rubu (engl. *edge computing*) sa svojim *AWS Greengrass* servisom. Dodatno, može se pohvaliti najjačom programerskom zajednicom, opširnom i temeljitom dokumentacijom te brojnim vodičima i resursima.

Što se tiče cijene, *AWS IoT Core* odvojeno naplaćuje korištenje usluga spajanja i slanja/prima poruka, *device shadow*a, operacije registra „stvari“ i korištenje mehanizma pravila (engl. *rules engine*). Cijena ovih usluga razlikuje se po svakoj regiji, a poruke se mjere u inkrementu od 5 kB. *AWS* nudi kalkulator pomoću kojega se može izračunati približna cijena korištenja platforme.

Azure IoT Hub nudi dvije razine pretplate: osnovnu i standardnu. Razlikuju se po broju značajki koje podržavaju. Standardna razina omogućuje sve značajke, dok osnovna razina omogućuje samo podskup značajki i namijenjena je za *IoT* sustave čija komunikacija ide isključivo iz smjera uređaja prema *cloud*u [20].

Kako bi se analizirale i usporedile cijene korištenja platformi, razmatran je slučaj u kojem uređaji svake minute šalju jednu poruku veličine 1 kB. Na slici 4.9 prikazan je graf u kojem jedna os predstavlja cijenu korištenja platforme, a druga os predstavlja broj spojenih uređaja. Graf je u logaritamskoj skali zbog lakšeg prikaza [20]. Plavom bojom (najdebljom linijom) prikazano je kretanje cijene *AWS*-a, zelenom i ljubičastom (srednje debelom linijom) prikazane su cijene *Azure* platforme (standardni plan je uvijek skuplji od osnovnog) te narančastom bojom (najtanjom linijom) prikazana je cijena *Google Cloud* platforme, no ona trenutno nije u razmatranju.



Slika 4.9 Funkcija odnosa cijene (u dolarima) i rastućeg broja uređaja [20]

Iz grafa se može zaključiti da je *AWS* platforma isplativija do 50 tisuća istovremeno spojenih uređaja, a nakon toga *Azure IoT* platforma postaje isplativija. No, treba se uzeti u obzir kako i u kojoj regiji se koristi *AWS IoT* platforma, budući da odvojeno naplaćuje usluge.

5. Implementacija sustava

Programiranje mikrokontrolerske strane sustava napravljeno je korištenjem C programskog jezika, FreeRTOS operacijskog sustava te ESP-IDF-a v5.4.

Što se tiče *web* aplikacije, za serverski poslužitelj korišten je *Spring Boot framework* te Java programski jezik, a korisničko sučelje izrađeno je pomoću Vue.js programskog jezika.

5.1. Konfiguracija *AWS IoT* platforme

5.1.1. Definiranje pravila pristupa (engl. *policies*)

Prva korak koji treba napraviti je kreirati politike pristupa kako bismo omogućili ili uskratili „stvarima“ pristup određenim resursima. U sklopu razvijenog sustava potrebna je jedna politika pristupa za *IoT* uređaje, jedna za *web* aplikaciju te jedna za *fleet provisioning* postupak. Za *IoT* uređaje i *web* aplikaciju koristit ćemo istu politiku pristupa koja im omogućuje spajanje i pristup bilo kojem resursu (kôd 5.1). U uređajima za proizvodnju ne preporučuje se ovakav pristup, no u pokaznom sustavu u redu je ovakav pristup. Za *fleet provisioning* postupak koristit ćemo posebnu politiku pristupa koja omogućuje pretplaćivanje, primanje i objavu poruka samo na *MQTT* teme potrebne za *fleet provisioning* (kôd 5.2). Spomenute politike pristupa nužno je kreirati u *AWS IoT Core* sučelju navigiranjem do *„Security“ -> „Policies“ -> „Create policy“* te zatim odabirom na *JSON* format unosa i kopiranjem jedne od navedenih politika pristupa (kôd 5.1, kôd 5.2) ili kreiranjem posebne politike pristupa.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

Kôd 5.1 Politika pristupa s omogućenim pristupom svakom resursu

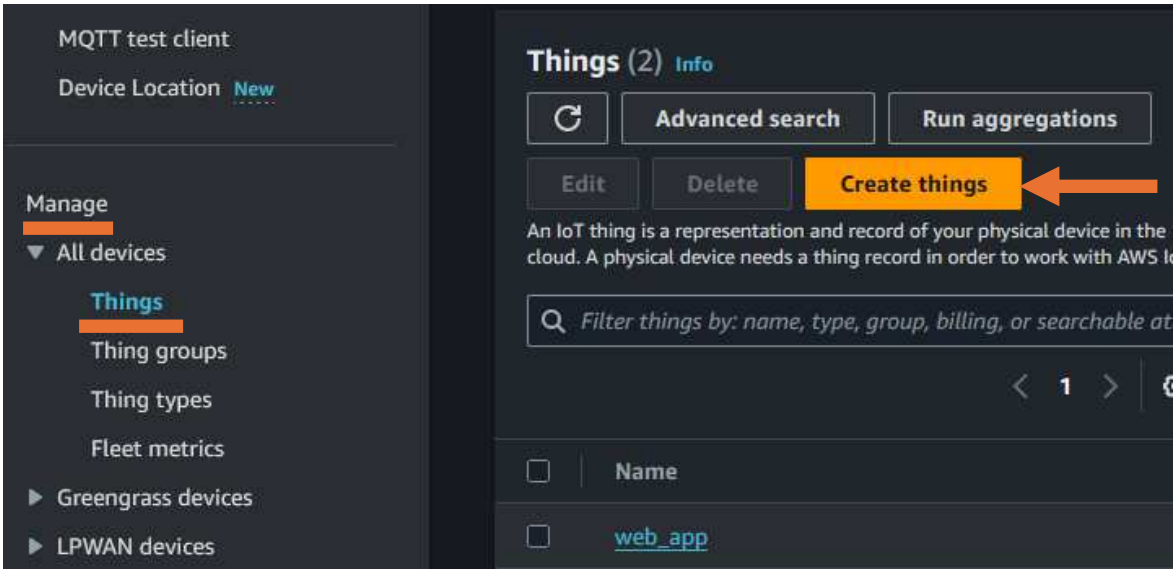
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:eu-central-
1:186460055589:topic/$aws/certificates/create/*",
        "arn:aws:iot:eu-central-
1:186460055589:topic/$aws/provisioning-
templates/ClaimProvisioningTemplate/provision/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:eu-central-
1:186460055589:topicfilter/$aws/certificates/create/*",
        "arn:aws:iot:eu-central-
1:186460055589:topicfilter/$aws/provisioning-
templates/ClaimProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

Kôd 5.2 Politika pristupa za *fleet provisoning*

5.1.2. Kreiranje „stvari“ (engl. *things*)

S obzirom na to da je kod *web* aplikacije za interakciju s *Device Shadow* servisom korišten *MQTT* protokol, kreirat ćemo „Stvar“ koja će predstavljati *web* aplikaciju na *AWS IoT* platformi. Osim *web* aplikacije, „stvari“ će biti i *IoT* uređaji, no za njih ne moramo ručno kreirati „stvari“, već će se za to pobrinuti *fleet provisioning* servis.

Za kreiranje „stvari“, potrebno je otvoriti *AWS IoT* sučelje, navigirati do opcije *„All devices“* -> *„Things“* -> *„Create things“* (slika 5.1).



Slika 5.1 Kreiranje "stvari"

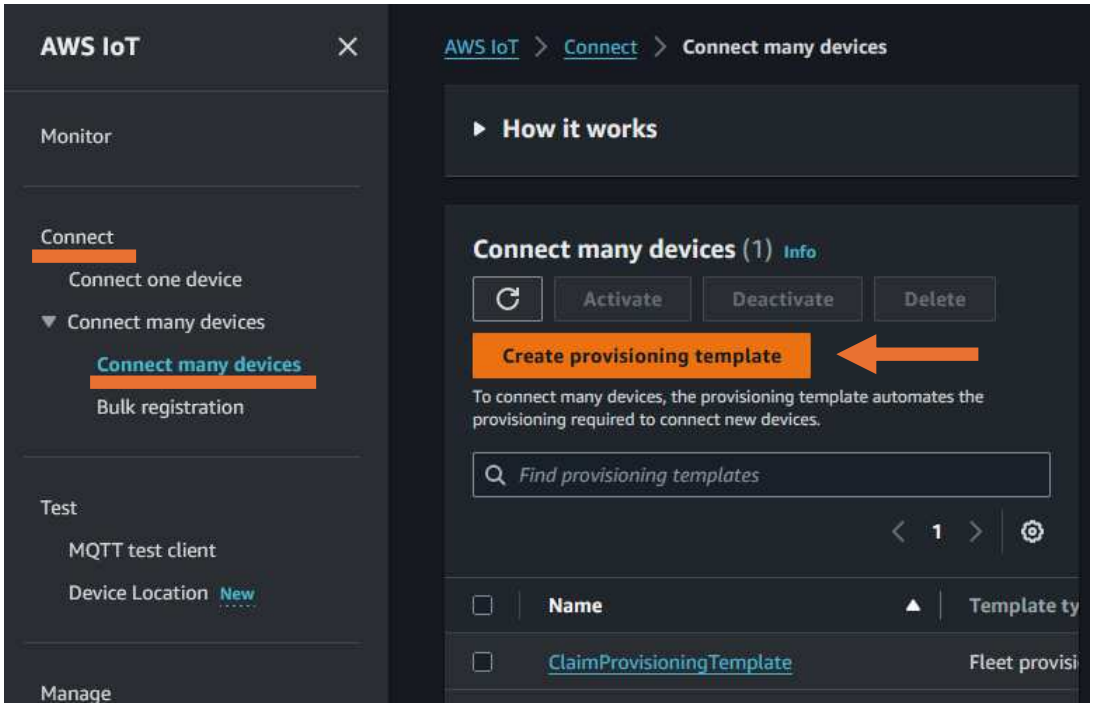
Dovoljno je kreirati jednu instancu „stvari“ s proizvoljnim imenom i bez *Device Shadow*a. Ponudit će se novi zaslon s izborom certifikata. Na tom zaslonu odabrati opciju auto generiranje novog certifikata. Zatim je potrebno asocirati politiku pristupa (kôd 5.1) s auto generiranim certifikatom. Otvorit će se zaslon s mogućnošću preuzimanja novog certifikata i privatnog ključa. Potrebno ih je preuzeti te kopirati u *„resources“* mapu u projektu za *back-end* programsku podršku.

5.1.3. Kreiranje grupe „stvari“

Za lakše upravljanje automatski dodanim *IoT* uređajima, potrebno je stvoriti grupu „stvari“ proizvoljnog imena, a *fleet provisioning* servis će brinut o tome da novokreirane „stvari“ budu dodane u pripadajuću grupu „stvari“.

5.1.4. Postavljanje *fleet provisioning* servisa

Za postavljanje *fleet provisioning* servisa koji je zaslužan za automatizirano registriranje „stvari“, potrebno je kreirati *fleet provisioning* predložak. Za to je potrebno navigirati do *„Connect“* -> *„Connect many devices“* te odabrati opciju *„Create provisioning template“* (slika 5.2).



Slika 5.2 Kreiranje *fleet provisioning* predloška

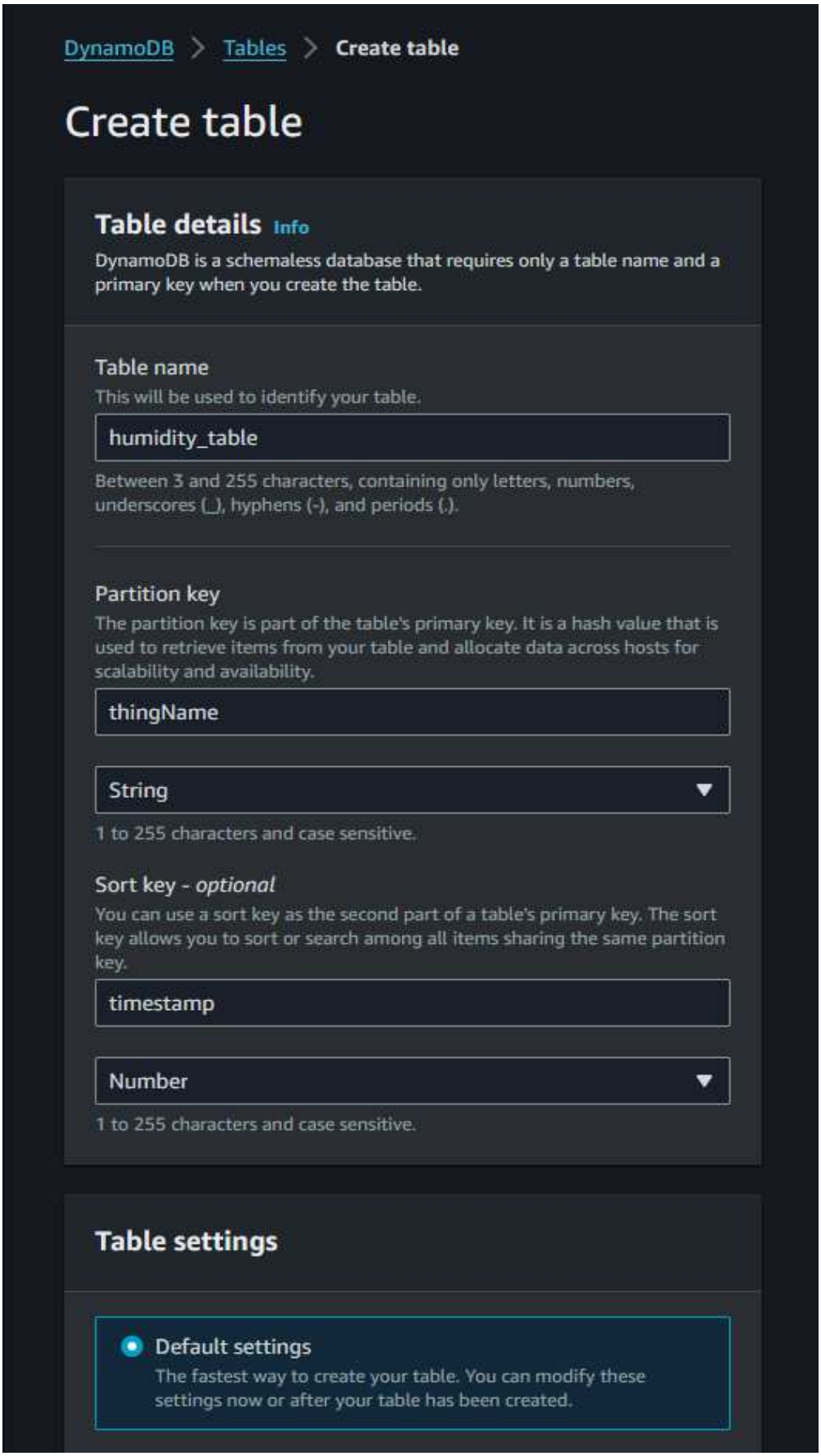
Kao metodu *provisioning*a potrebno je odabrati *„provisioning devices with claim certificates“*. Na sljedećem zaslonu potrebno je postaviti predložak kao aktivan i dati mu proizvoljni naziv. Potrebno će biti i kreirati novu ulogu (engl. *role*) u *IAM* servisu te joj asocirati već napravljeni predložak politike pristupa naziva *„AWSIoTThingRegistration“* koji će omogućiti registraciju novih uređaja na *AWS IoT*. Nakon što je nova uloga kreirana, odabrati politiku pristupa iz poglavlja 5.1.1 (kôd 5.2). Na sljedećem zaslonu, dovoljno je odabrati opciju da se ne obavlja nikakva akcija prije *provisioning*a. Osim toga, odabrati opciju automatskog kreiranja „stvari“ prilikom *provisioning*a uređaja te postaviti prefiks za novokreirane „stvari“ i asocijaciju s nedavno kreiranom grupom „stvari“ *IoT* uređaja. Naposljetku, ponovno odabrati politiku pristupa za *fleet provisioning* (kôd 5.2). Tim korakom se *fleet provisioning* predložak kreira i omogućuje uređajima automatsku registraciju na *AWS IoT* platformu.

5.1.5. Generiranje novog X.509 certifikata

Posebni certifikat je potreban kako bi *IoT* uređaj mogao pristupiti *MQTT* brokeru te na njega slati poruke pri *fleet provisioning* postupku (sukladno poglavlju 4.3.4). Generiranje novog certifikata radi se odabirom *„Security“ -> „Certificates“ -> „Add certificate“ -> „Create certificate“*. Politika pristupa pridružena certifikatu trebala bi biti ista kao i za *fleet provisioning* (kôd 5.2). Dobiveni certifikat i privatni ključ potrebno je postaviti u *„components/aws/certs“* podmapu u mapi koja sadrži potrebnu programsku podršku za mikrokontrolere.

5.1.6. Izrada tablica u bazi podataka

Za pohranu senzorskih podataka odabrana je *DynamoDB* baza podataka (poglavlje 4.7). Potrebno je izraditi dvije tablice za pohranjivanje senzorskih podataka, jednu za vrijednosti temperature, a drugu za vrijednosti vlažnosti zraka. Traženjem *DynamoDB* servisa te zatim odabirom *„Tables“ -> „Create table“* dolazi se do zaslona prikazanog na slici 5.3. Slika 5.3 prikazuje podatke za izradu tablice vlažnosti zraka, a za izradu tablice temperature potrebno je samo promijeniti naziv tablice. Kao particijski ključ odabran je naziv „stvari“ kako bi razlikovali koji senzorski podatak je stigao s kojeg *IoT* uređaja.



Slika 5.3 Izrada *DynamoDB* tablice

5.1.7. Kreiranje pravila (engl. *rules*)

Kako bi *AWS* platforma pohranjivala senzorske podatke dobivene s *MQTT* tema (*„sensor/humidity“* i *„sensor/temperature“*) u bazu podataka, potrebno je kreirati dva pravila (jedno za temperaturu, drugo za vlažnost) koja će se izvoditi prilikom stizanja poruke na navedene teme. Pravilo (*rule*) kreira se tako da se u izborniku odabere *„Message routing“ -> „Rules“ -> „Create rule“.* Nakon izbora imena pravila, kao *SQL* upit potrebno je staviti `SELECT topic(1) as thingName, timestamp, humidity FROM '+/sensor/humidity'` za vlažnost ili `SELECT topic(1) as thingName, timestamp, temperature FROM '+/sensor/temperature'` za temperaturu.

Ovime se u bazu podataka stavljaju podatci o nazivu „stvari“ (particijski ključ), vremenu i vrijednosti podatka sa senzora. Za akciju, koja će se izvoditi prilikom aktivacije pravila, staviti *„DynamoDBv2“* te odabrati odgovarajuću tablicu iz baze podataka u koju će se podatci spremati.

5.2. Struktura mikrokontrolerske strane

Stablo projekta programske podrške za mikrokontrolersku obitelj ESP32 koji prikazuje najvažnije datoteke te dodatno objašnjava njihovu svrhu:

```
.
├── CMakeLists.txt
├── Makefile
├── bsp            // podrška za hardverske komponente
│   ├── CMakeLists.txt
│   ├── esp_devkit  // ESP specifična podrška
│   │   ├── config
│   │   │   └── bsp_config.h
│   │   └── src
│   │       ├── button_driver.c   // Driver za tipku
│   │       ├── humidity_driver.c // Driver za senzor vlažnosti
│   │       ├── led_driver.c      // Driver za svjetleću diodu
│   │       ├── nvs_driver.c      // Driver na NVS
│   │       └── temperature_driver.c  // Driver za senzor temperature
│   └── inc
│       └── bsp
├── components
│   ├── MQTT
│   │   ├── CMakeLists.txt
│   │   ├── mqtt_agent.c      // MQTT podrška
│   │   └── mqtt_agent.h
│   ├── aws
│   │   ├── CMakeLists.txt
│   │   ├── OTA
│   │   │   ├── ota_mqtt.c    // OTA podrška
│   │   │   └── ota_mqtt.h
│   │   ├── aws.c             // logika upravljanja cloud stranom
```

```
│   │   ├── aws.h
│   │   ├── certs       // certifikati potrebni za pristup AWS IoT-u
│   │   │   ├── aws-root-ca.pem
│   │   │   ├── aws_code_sign.crt
│   │   │   ├── certificate.pem.crt  // certifikat iz poglavlja 5.1.5
│   │   │   ├── endpoint.txt
│   │   │   └── private.pem.key   // privatni ključ iz poglavlja 5.1.5
│   │   ├── device_shadow
│   │   │   ├── device_shadow.c     // upravljanje Device Shadow servisom
│   │   │   └── device_shadow.h
│   │   ├── fleet_provisioning
│   │   │   ├── fleet_provision.c  // upravljanje Fleet provisioningom
│   │   │   ├── fleet_provision.h
│   │   │   ├── fleet_provision_serializer.c
│   │   │   └── fleet_provision_serializer.h
│   │   ├── publisher
│   │   │   ├── publisher.c  // objavljivanje senzorskih podataka
│   │   │   └── publisher.h
│   │   └── subscriber
│   │       ├── subscriber.c // pretplaćivanje na MQTT teme (nekorišteno)
│   │       ├── subscriber.h
│   │       ├── subscriber_manager.c  // upravitelj i handler pretplata
│   │       └── subscriber_manager.h
│   ├── board
│   │   ├── CMakeLists.txt
│   │   ├── Kconfig
│   │   ├── board.c     // high-level podrška korištenih HW komponenti
│   │   ├── board.h
│   │   ├── button
│   │   │   ├── button.c      // high-level podrška za tipke
│   │   │   └── button.h
│   │   ├── led
│   │   │   ├── led.c         // high-level podrška za svjetleće diodu
│   │   │   └── led.h
│   │   └── sensors
│   │       ├── sensor.c      // high-level podrška za senzore
│   │       └── sensor.h
│   ├── error_handling
│   ├── lib
│   │   └── CMakeLists.txt
│   ├── logging
│   ├── ntp                   // NTP klijent za sinkronizaciju vremena
│   ├── qrcode                // generator QR koda za Wi-Fi provisioning
│   ├── randomizer
│   ├── reset                 // podrška resetiranja pločice i memorije
│   ├── storage               // podrška za memoriju sustava
│   └── wifi_cm               // menadžer Wi-Fi sučelja
├── lib
│   ├── cbor
│   └── esp-aws-iot           // AWS specifična podrška za ESP platformu
├── main
│   ├── CMakeLists.txt
│   ├── component.mk
│   └── main.c          // glavni program, zadužen za pokretanje  sustava
├── partitions.csv      // tablica particija
├── sdkconfig-esp32     // konfiguracija za ESP-WROOM-32 mikrokontroler
├── sdkconfig-esp32c3   // konfiguracija za ESP32-C3 mikrokontroler
└── sdkconfig
```

Kôd 5.3 Stablo projekta mikrokontrolerske strane

Programska podrška podržava ESP-WROOM-32 i ESP32-C3 mikrokontroler, a kopiranjem odgovarajuće konfiguracije (*sdkconfig-esp32* ili *sdkconfig-esp32c3*) u datoteku *sdkconfig* se omogućava kompilacija projekta za željeni mikrokontroler.

5.3. Implementacija *NVS driver*-a

NVS memorija služi kako bi se trajno mogli sačuvati podatci za spajanje na *Wi-Fi* mrežu te sve vjerodajnice potrebne za pristup *AWS IoT* platformi (uključujući certifikat, privatni ključ i naziv „stvari“ dobivene iz postupka *fleet provisioning*a). *Wi-Fi* podatci se spremaju automatski prilikom procesa *Wi-Fi provisioning*a, dok je isječak spremanja *AWS* vjerodajnica prikazan kôdom 5.4. `Storage_write_blob` je funkcija koja predstavlja apstrakciju za pisanje u *NVS* memoriju ESP mikrokontrolera.

```
...
if(STATUS_OK == status)
{
    LOGI(TAG, "Received thing name: %.*s", aws_creds-
>device_thing_name_len, aws_creds->device_thing_name);
    aws_creds->device_certificate_len++; // so it includes
null termination
    aws_creds->device_private_key_len++; // so it includes
null termination
    status = aws_save_device_creds(aws_creds);
}
...
err_status_t aws_save_device_creds(aws_creds_t *p_dev_creds)
{
    if(NULL == p_dev_creds)
    {
        LOGE(TAG, "Invalid device creds, not saving creds.");
        return STATUS_FAIL;
    }
    LOGI(TAG, "Device certficateID: %.*s", p_dev_creds-
>device_certificate_id_len, p_dev_creds-
>device_certificate_id);
    return storage_write_blob(AWS_CREDS_LOCATION,
p_dev_creds, sizeof(aws_creds_t));
}
```

Kôd 5.4 Spremanje *AWS* vjerodajnica u *NVS*

5.4. *Wi-Fi manager*

Wi-Fi sučelje koristi FreeRTOS zadatak (engl. *task*) kako bi prvotno inicijalizirala *Wi-Fi* sučelje, pokrenula ga te zatim konstantno procesirala događaje vezane uz *Wi-Fi*.

5.4.1. Inicijalizacija

Inicijalizacija se obavlja na način prikazan u kôdu 5.5. Funkcija `esp_netif_init()` postavlja internu strukturu podataka za upravljanje mrežnim sučeljima, dok `esp_netif_create_default_wifi_sta` postavlja *Wi-Fi* sučelje u *Station* način rada (tako da se ESP spaja na *Wi-Fi* pristupnu točku). Potom se odabire uobičajena (zadana) konfiguracija te se napokon inicijalizira *Wi-Fi* sučelje funkcijom `esp_wifi_init()`.

```
esp_netif_init();
ESP_ERROR_CHECK(esp_event_loop_create_default());
esp_netif_create_default_wifi_sta();
wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
ESP_ERROR_CHECK(esp_wifi_init(&cfg));
```

Kôd 5.5 Inicijalizacija *Wi-Fi* sučelja

5.4.2. *Wi-Fi handlers*

Osim postavljanja konfiguracije, zadaju se i funkcije (*handler*s) koje će obrađivati *Wi-Fi* događaje (kôd 5.6). Na ovakav način, odvojeni su događaji povezani s *Wi-Fi* mrežom (pokretanjem, spajanjem i odspajanjem) od *Wi-Fi provisioning* događaja te događaja povezanih s dobivanjem *IP* adrese.

```
ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT,
ESP_EVENT_ANY_ID, (esp_event_handler_t)_wifi_event_cb,
NULL));

ESP_ERROR_CHECK(esp_event_handler_register(WIFI_PROV_EVENT,
ESP_EVENT_ANY_ID, (esp_event_handler_t)_wifi_prov_event_cb,
NULL));
    ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT,
IP_EVENT_STA_GOT_IP, (esp_event_handler_t)_ip_event_cb,
NULL));
```

Kôd 5.6 Registracija *handlera* za *Wi-Fi* događaje

Funkcija `_wifi_event_cb` zadužena je za signalizaciju spajanja ili odspajanja *Wi-Fi* mreže slanjem događaja, a ti događaji zatim ili pokreću *timer* za ponovno spajanje ili postavljaju zastavicu u memoriji da je *Wi-Fi* spojen i da su podatci za spajanje spremljeni u memoriji.

Funkcija `_wifi_prov_event_cb`, koja obrađuje događaje vezane uz *Wi-Fi provisioning*, zadužena je za ispis primljenih podataka za spajanje (*SSID* i lozinke), ispis u slučaju neuspjelog *provisioning*a te spremanje primljenih podataka u memoriju i pokretanje povezivanja na *Wi-Fi* mrežu.

Naposljetku, `_ip_event_cb` obrađuje samo događaj uspješnog dobivanja *IP* adrese te, u tom slučaju, postavlja zastavicu da je *Wi-Fi* uspješno spojen i inicijalizira *NTP* klijent koji služi za sinkronizaciju stvarnog vremena mikrokontrolera kako bi se olakšao postupak kasnijeg spajanja putem *TLS*-a (kôd 5.7).

```
case IP_EVENT_STA_GOT_IP:
{
    ip_event_got_ip_t* event =
(ip_event_got_ip_t*)event_data;
    LOGI(TAG, "Connected with IP Address:" IPSTR,
IP2STR(&event->ip_info.ip));
    xEventGroupSetBits(wifi_event_group,
WIFI_CONNECTED_EVENT);
    _wifi_conn_retry_timer_delete();
    storage_provisioned_set(true);
    ntp_client_init_sntp();
break;
}
```

Kôd 5.7 Uspješno dobivena *IP* adresa

5.4.3. Pokretanje *Wi-Fi* sučelja

Nakon inicijalizacije, *Wi-Fi* sučelje se pokreće na način da se prvo provjeri zastavica u memoriji koja signalizira jesu li u memoriji već pohranjeni pristupni podatci s kojima se uređaj uspješno bio spojio na *Wi-Fi* mrežu. Ako jesu, *Wi-Fi* modul se stavlja u *Station* način rada te se pokreće spajanje na *Wi-Fi* mrežu. Ako nisu, sustav pokreće postupak *provisioning*a putem *Bluetooth Low Energy*ja. Metoda kojom je ostvarena takva funkcionalnost prikazana je putem kôda 5.8.

```
static void _wifi_start(void)
{
    if(storage_provisioned_get())
    {
        ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));

        ESP_ERROR_CHECK(esp_wifi_start());
    }
    else
    {
        _provision_start_ble();
    }
}
```

Kôd 5.8 Pokretanje *Wi-Fi* sučelja

5.4.4. *Wi-Fi provisioning*

Spajanje uređaja na *Wi-Fi* mrežu izvodi se pomoću *Wi-Fi provisioning* postupka (poglavlje 3.4). Prije pokretanja samog postupka *provisioning*a, potrebno je konfigurirati postavke samog servisa (kôd 5.9). U konfiguracijskoj strukturi odabran je *BLE* način. Prilikom *provisioning*a, uređaj svoje dostupne servise oglašava preko *BLE*-a, a razlikovanje tih servisa postiže se postavljanjem posebnog identifikacijskog broja servisa (`custom_service_uuid`). `Service_name` varijabla dohvaća *MAC* adresu uređaja kako bi se osigurala jedinstvenost postupka za svaki uređaj.

```
wifi_prov_mgr_config_t config = {
        .scheme = wifi_prov_scheme_ble,
        .scheme_event_handler =
WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM,
        .app_event_handler    = WIFI_PROV_EVENT_HANDLER_NONE,
    };
uint8_t custom_service_uuid[] =
{ 0x21, 0x43, 0x65, 0x87, 0x09, 0xba, 0xdc, 0xfe, 0xef, 0xcd,
0xab, 0x90, 0x78, 0x56, 0x34, 0x12 };
wifi_prov_scheme_ble_set_service_uuid(custom_service_uuid);
char service_name[12];
_get_device_service_name(service_name, sizeof(service_name));
```

Kôd 5.9 Inicijalizacija *Wi-Fi provisioning*a

Zatim je potrebno pozvati funkciju koja će inicijalizirati *Wi-Fi provisioning* menadžera: `wifi_prov_mgr_init(config).`

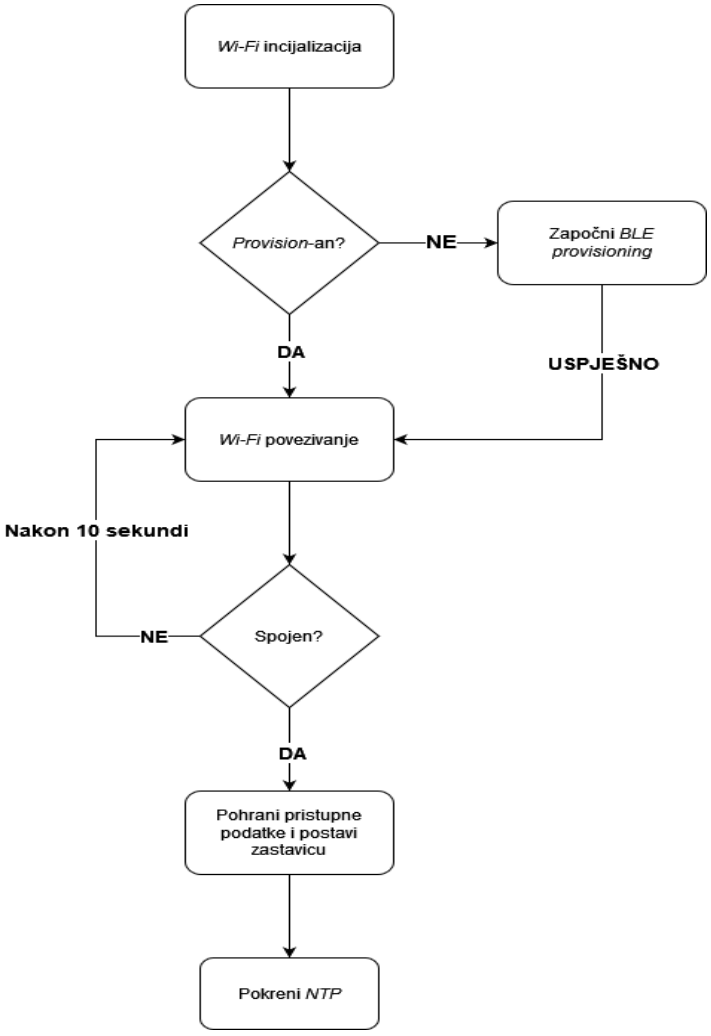
Naposljetku, *provisioning* proces se započinje te se na serijsko sučelje mikrokontrolera ispisuje *QR* kod kojeg je moguće skenirati putem mobilne aplikacije kako bi se korisniku pojednostavio čitav proces (kôd 5.10). Niz znakova koji dokazuje posjedovanje uređaja (engl. *Proof-of-Possession, PoP*) nije jedinstven za svaki uređaj, već je identičan niz znakova: „**abcd1234**“.

```
ESP_ERROR_CHECK(wifi_prov_mgr_start_provisioning(security,
p_pop, service_name, p_service_key));

_print_qr(service_name, p_pop, "ble");
```

Kôd 5.10 Pokretanje *Wi-Fi provisioning*a

Cjelokupni dijagram koji sažima proces *Wi-Fi* servisa prikazan je na slici 5.4.



Slika 5.4 Dijagram *Wi-Fi* sučelja

5.5. Spajanje na *MQTT* brokera

Za spajanje na *MQTT* brokera potrebno je prvo otvoriti *TLS* vezu između uređaja i krajnje točke (engl. *endpoint*). Krajnja točka na koju se spaja ovisi o *AWS* računu te se može pronaći na *AWS* sučelju. Za otvaranje *TLS* veze potrebno je brokeru predati važeći certifikat i privatni ključ. Ako je uređaj registriran na *AWS* platformu, predaje certifikate i privatni ključ dobiven prilikom *fleet provisioning*a. Ako uređaj nije registriran, koristi certifikat i privatni ključ iz poglavlja 5.1.5 kako bi otvorio *TLS* vezu, spojio se na *MQTT* brokera i registrirao na *AWS* platformu. Ostala konfiguracija identična je za oba slučaja. Postavljanje certifikata i privatnog ključa za oba slučaja prikazan je u kôdu 5.11. U slučaju da uređaj još nije registriran certifikat i privatni ključ dohvaćaju se iz datoteka koji su ugrađene u uređaj, dok se u slučaju ako je registriran dohvaćaju iz memorije (a spremljeni su nakon *fleet provisioning* postupka).

```
if(false == b_client_mode)
{
    p_network_context->pcClientCert =
certificate_pem_crt_start;
    p_network_context->pcClientCertSize =
certificate_pem_crt_end - certificate_pem_crt_start;
    p_network_context->pcClientKey = private_pem_key_start;
    p_network_context->pcClientKeySize = private_pem_key_end
- private_pem_key_start;
}
else
{
    aws_load_device_creds(&aws_creds);
    p_network_context->pcClientCert = (const char *)
aws_creds.device_certificate;
    p_network_context->pcClientCertSize = (uint32_t)
aws_creds.device_certificate_len;
    p_network_context->pcClientKey = (const char *)
aws_creds.device_private_key;
    p_network_context->pcClientKeySize = (uint32_t)
aws_creds.device_private_key_len;
}
```

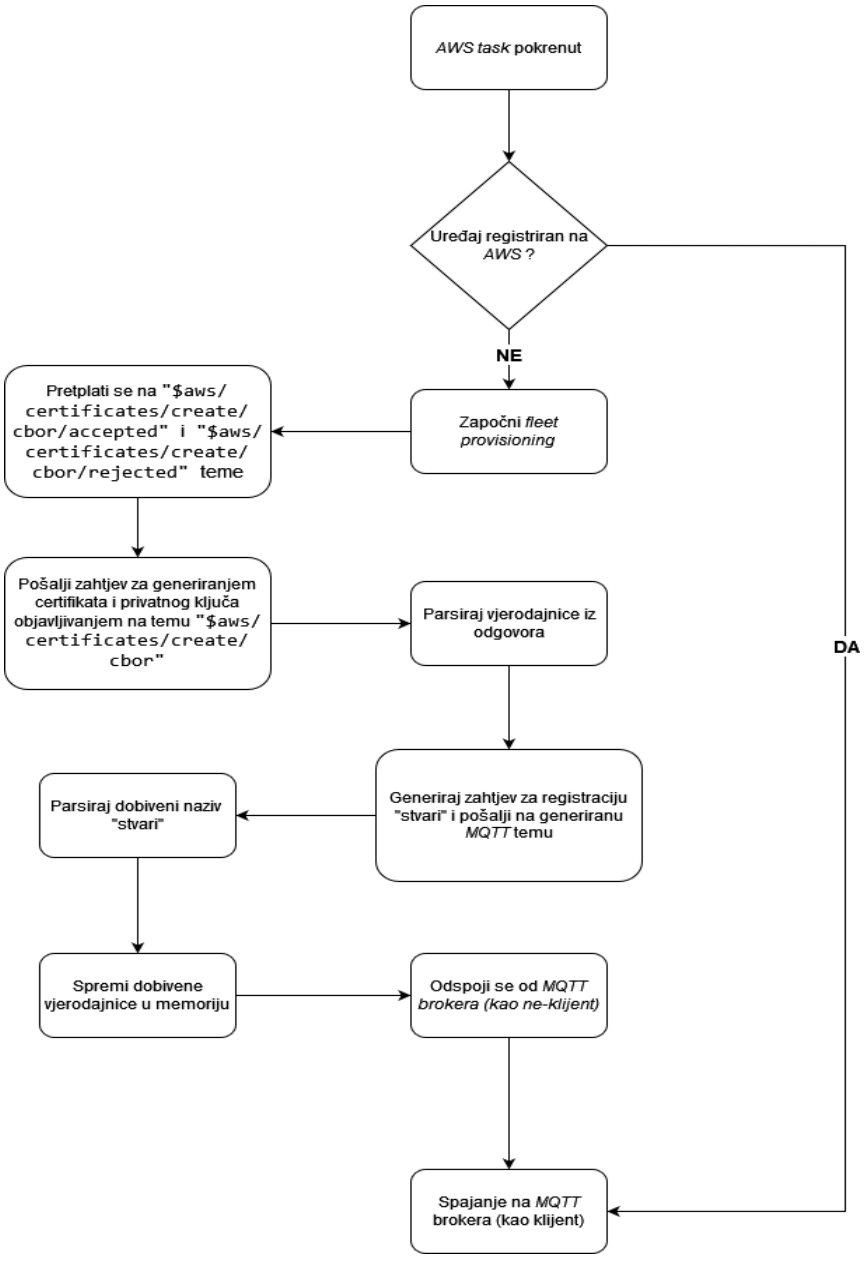
Kôd 5.11 Postavljanje certifikata i privatnog ključa

Otvaranje *TLS* veze pokreće se pozivom `xTlsConnect(p_network_context)`.

Nakon uspostave *TLS* veze uređaj se pokušava spojiti na *MQTT* brokera pomoću metode `MQTT_Connect`.

5.6. *Fleet provisioning* implementacija

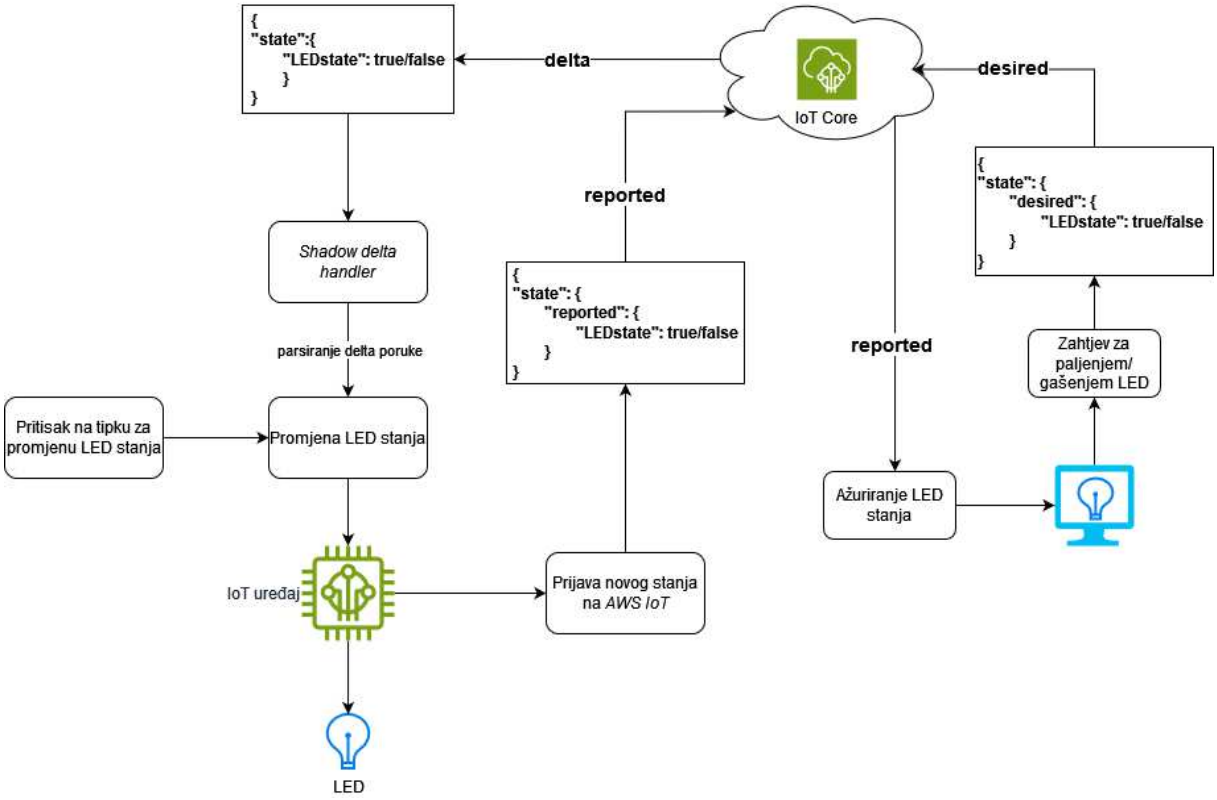
Pomoću dijagrama (slika 5.5) prikazan je implementirani postupak *fleet provisioning*a sa strane mikrokontrolera. Umjesto *JSON* poruka pri generiranju certifikata i ključa korišten je *CBOR* format poruke zbog svoje manje veličine.



Slika 5.5 Dijagram *fleet provisioning* implementacije

5.7. Upravljanje *LED*-om (*Device Shadow* servis)

Upravljanje svjetlećom diodom moguće je kratkim pritiskom *„BOOT“* tipke na razvojnom sustavu ili korištenjem *web* aplikacije, a napravljeno je koristeći *Device Shadow* servis. Dijagram i razmijenjene poruke prikazane su na slici 5.6.



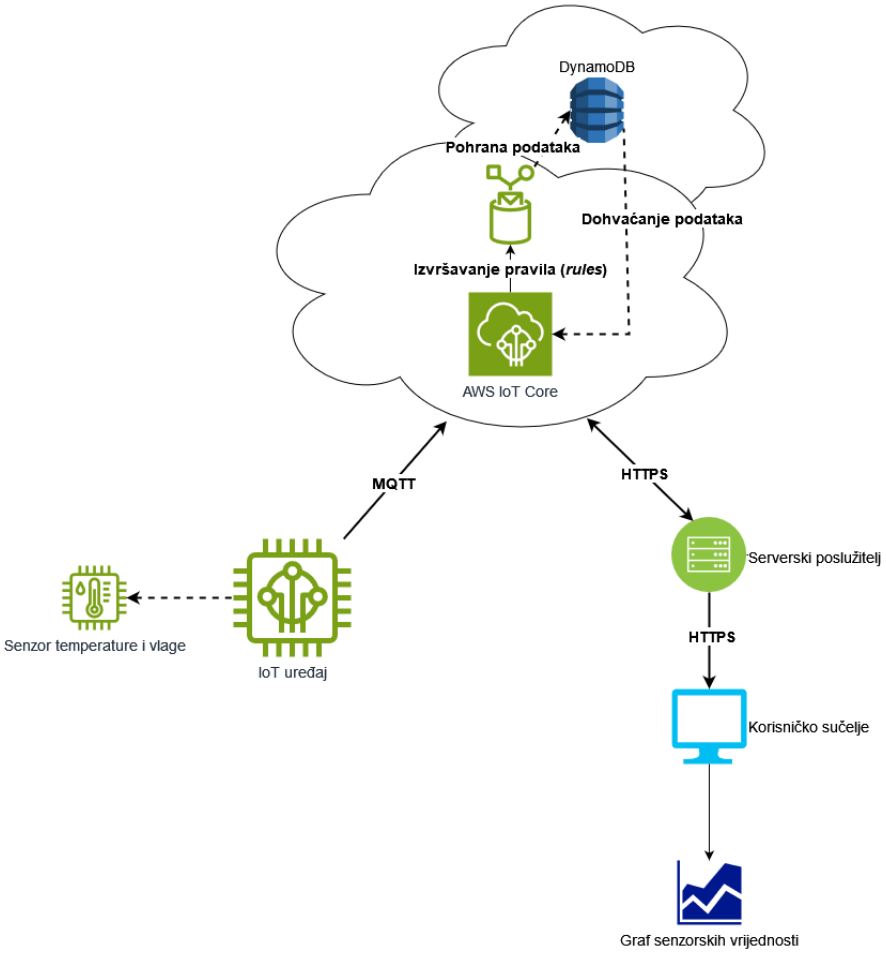
Slika 5.6 Upravljanje *LED*-om

Web aplikacija na zahtjev promjene stanja svjetleće diode generira *„desired“ shadow* poruku te ju objavljuje na *AWS IoT*. Ako postoji razlika između prijavljenog stanja i željenog stanja, *AWS IoT* objavljuje *„delta“* poruku u kojoj je sadržana razlika. Funkcija zadužena za upravljanje *„delta“* porukama mijenja stanje svjetleće diode te prijavljuje novo stanje (koje je sada isto kao i željeno) na *AWS IoT*. Primanjem novog prijavljenog stanja, *web* aplikacija ažurira svoje sučelje kako bi prikazao novo stanje svjetleće diode.

5.8. Senzorski podatci

Potpuni ciklus senzorskih podataka prikazan je na slici 5.7. *IoT* uređaj periodički prikuplja senzorske podatke sa (simuliranog) senzora temperature i vlage te ih pomoću *MQTT* protokola šalje *MQTT* brokeru koji se nalazi na *AWS IoT Core*u. Ranije definiranim

pravilima (engl. *rules*), *AWS IoT* obavlja pohranu primljenih senzorskih podataka u *DynamoDB* bazu podataka. Ti podatci se zatim dohvaćaju putem *HTTPS* protokola na zahtjev serverskog poslužitelja, koji te podatke dalje šalje korisničkom sučelju. Naposljetku, korisničko sučelje prikazuje graf vrijednosti u ovisnosti o vremenu za sve pohranjene senzorske podatke.



Slika 5.7 Ciklus senzorskih podataka

5.9. Udaljena nadogradnja programske podrške (*FOTA*)

Udaljena nadogradnja programske podrške ima vrlo jednostavnu implementaciju s obzirom na pripremljenu biblioteku *AWS IoT* programske podrške s *OTA* agentom. Prilikom implementacije najvažnije je *OTA* agentu (njegovom sučelju) pridružiti *MQTT* operacije za pretplaćivanje, objavljivanje i odjavu pretplate (kôd 5.12), a nakon toga inicijalizirati samog *MQTT* agenta (kôd 5.13) te u zasebnom *FreeRTOS* zadatku (engl. *task*) pozvati funkciju zaduženu za procesiranje daljnjih primljenih *OTA* događaja (kôd 5.14).

```
/* Initialize the OTA library MQTT Interface.*/
p_ota_interfaces->mqtt.subscribe = _ota_mqtt_subscribe;
p_ota_interfaces->mqtt.publish = _ota_mqtt_publish;
p_ota_interfaces->mqtt.unsubscribe = _ota_mqtt_unsubscribe;
```

Kôd 5.12 Pridruživanje MQTT operacije OTA agentu

```
if(OtaErrNone != OTA_Init(&otaBuffer, &ota_interfaces,
                          (const uint8_t *) aws_thing_name,
                           _ota_event_handler))
{
      LOGE(TAG, "Could not initialize OTA");
      status = STATUS_FAIL;
}
```

Kôd 5.13 Inicijalizacija *OTA* agenta

```
static void _ota_mqtt_task(void *pParam)
{
    /* Calling OTA agent task. */
    OTA_EventProcessingTask( pParam );
    LOGI(TAG, "OTA TASK stopped.");
    vTaskDelete(NULL);
}
```

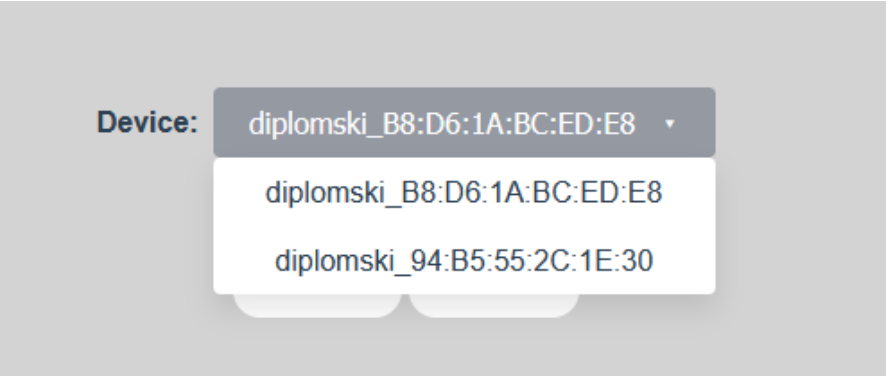
Kôd 5.14 *FreeRTOS* zadatak koji poziva funkciju za procesiranje *OTA* događaja

FOTA (engl. *Firmware over-the-air*) podrška ima u sebi ugrađenu uslugu potpisivanja programske podrške (engl. *code signing*) kako bi uređaji mogli potvrditi da je primljena programska podrška objavljena od autoriziranog korisnika te kako se nije mijenjala otkako je potpisana. Prilikom izrade novog *FOTA* zadatka, na *AWS* sučelju bit će potrebno kreirati novi profil za potpisivanje programske podrške. U sklopu kreiranja novog profila, morat će se generirati i novi certifikat za potpisivanje programske podrške te ispuniti putanja na kojoj će se certifikat nalaziti na uređaju. Putanja može biti proizvoljno odabrana (na primjer *„OTA Code Verify Key“*), no važno je da se podudara s definicijom:

```
#define pkcs11configLABEL_CODE_VERIFICATION_KEY.
```

6. Pokazni sustav

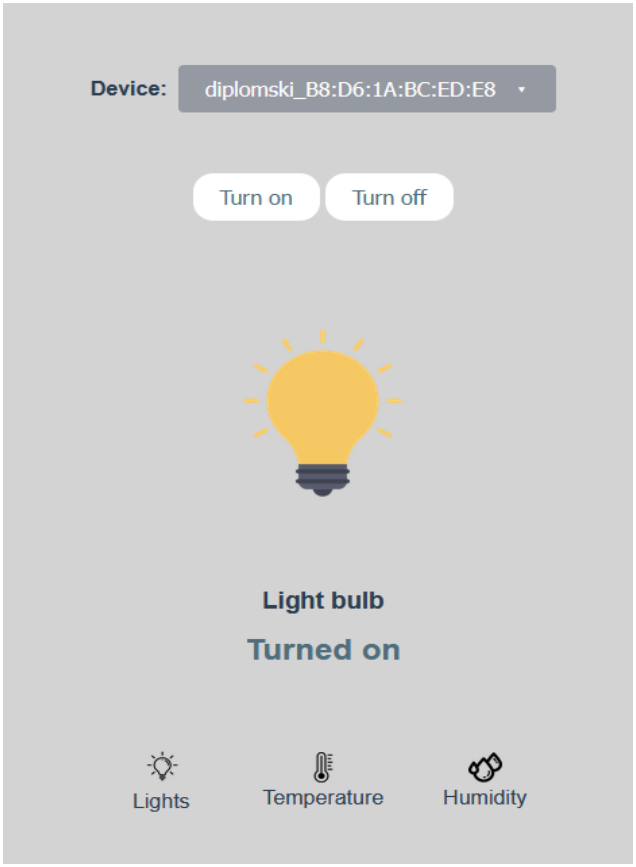
Rezultat sustava je *web* aplikacija koja pomoću svojega sučelja omogućuje korisnicima upravljanje svjetlećom diodom te prikaz izmjerenih senzorskih vrijednosti za određeni uređaj. Na slici 6.1 prikazan je izbornik pomoću kojega se može odabrati uređaj za upravljanje i prikaz podataka. Dostupna su dva uređaja s obzirom na to da su se koristila dva ESP32 mikrokontrolera koji su zatim registrirani na *AWS IoT fleet provisioning* postupkom.



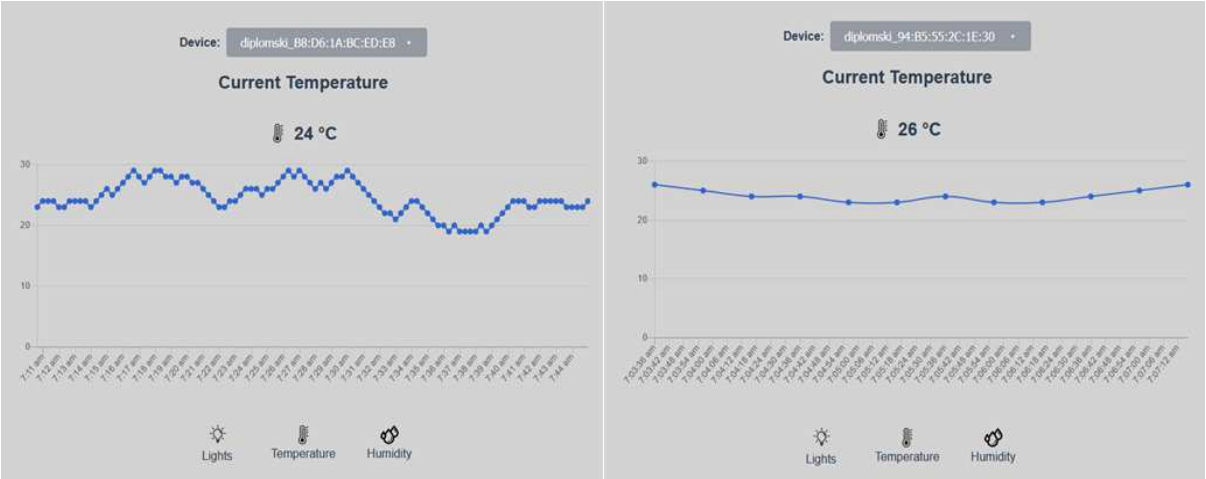
Slika 6.1 Izbornik uređaja

Na slici 6.2 prikazano je sučelje za upravljanje svjetlećom diodom. Žarulja svijetli jer je zadnje prijavljeno stanje svjetleće diode na uređaju bilo upaljeno. Pritiskom na tipke *„Turn on“* i *„Turn off“* upravlja se stanjem svjetleće diode tako da se šalje *„desired“ device shadow* poruka na *AWS IoT* platformu. Kada uređaj prijavi da je stanje svjetleće diode promijenjeno, korisničko sučelje također ažurira stanje svjetleće diode.

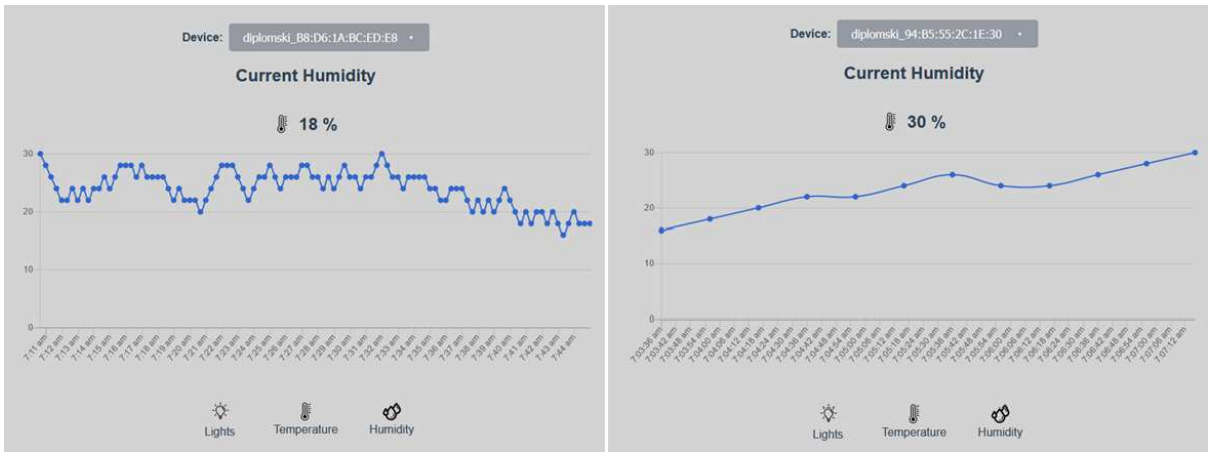
Na slikama 6.3 i 6.4 prikazani su grafovi izmjerenih temperatura i vlažnosti zraka u ovisnosti o vremenu za oba registrirana uređaja. Vidljivo je kako su grafovi i prikazane vrijednosti različite, a to se može zahvaliti particijskim ključevima koji primljene *MQTT* poruke raspoređuju ovisno o nazivu „stvari“.



Slika 6.2 Upravljanje svjetlećom diodom



Slika 6.3 Graf temperature u ovisnosti o vremenu za oba uređaja



Slika 6.4 Graf vlažnosti zraka u ovisnosti o vremenu za oba uređaja

Zaključak

U okviru rada prezentirana je arhitektura ESP32 mikrokontrolera i predstavljane su porodice koje su korištene u realizaciji pokaznog sustava. ESP32 mikrokontroleri u današnje vrijeme predstavljaju optimalan izbor za implementaciju IoT sustava zahvaljujući ugrađenoj podršci za *Wi-Fi* i *Bluetooth* (*BLE*) protokole te cijeni koja je značajno niža od konkurentskih proizvoda. Dodatnu prednost predstavlja podrška za najvažnije module i protokole za komunikaciju, programska razvojna okolina i biblioteke za apstrakciju sklopovlja, razmjerno velike *RAM* i *flash* memorije, vlastita *AWS IoT* biblioteka, izvrsna dokumentacija i brojna zajednica koja koristi ovu platformu.

AWS IoT platforma i njezini servisi detaljno su prezentirani kroz rad te su pokazane njezine različite mogućnosti: registracija uređaja na platformu (*AWS IoT Fleet Provision* servis), udaljena nadogradnja programske potpore (*AWS IoT jobs/OTA*), upravljanje aktuatorima (*AWS IoT Device Shadow*) i pohranjivanje senzorskih podataka u bazu podataka (*AWS IoT MQTT* broker, *rules engine* i integracija s *DynamoDB* bazom podataka). *AWS IoT* platforma pokazala se jednostavnom za implementaciju zbog svojih biblioteka koje se mogu koristiti u raznim programskim jezicima i na više operacijskih sustava. S druge strane, konfiguracija platforme nije jednostavna s obzirom na to da se nudi puno različitih usluga i servisa. Ipak, iznimno dobra dokumentacija i podrška zajednice olakšavaju cjelokupnu implementaciju i konfiguraciju sustava i platforme.

Kroz rad je razvijen pokazni sustav za koji je prikazan izgled korisničkog sučelja, korištenje *web* aplikacije koja omogućuje daljinsko upravljanje svjetlećom diodom te prikaz izmjerenih senzorskih vrijednosti putem grafa za sve registrirane uređaje na *AWS IoT* platformu. Pokazni sustav demonstrirao je značajke i servise *AWS IoT* platforme potrebne za razvoj umreženih uređaja koji se koriste u domeni Interneta stvari i pametnih kuća, a razvijena programska potpora sustava pogodna je za daljnji razvoj pametnih uređaja i uvođenje novih senzora i aktuatora.

Literatura

[1] Kocer S., Dundar O., Butuner R. *Programmable Smart Microcontroller Cards*. (2021). Poveznica: https://www.isres.org/books/Programlanabilir%20Ak%C4%B1ll%C4%B1%20Mikrodenetleyici%20Kartlar_01_16-12-2021.pdf; pristupljeno 21. svibnja 2024.

[2] A. Maier, A. Sharp and Y. Vagapov. *Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things*. Internet Technologies and Applications (ITA), Wrexham, UK, (2017).

[3] Atif, Muhammad & Shapna, Muralidharan & Ko, Heedong & Yoo, Byounghyun. *Wi-ESP—A tool for CSI-based Device-Free Wi-Fi Sensing (DFWS)*. Korea Institute of Science and Technology, Seoul, (2020).

[4] *ESP-IDF Security Features - Flash Encryption*, [*Online*]. Poveznica: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/security/flash-encryption.html; pristupljeno 26. svibnja 2024.

[5] *ESP-IDF API Guides - Partition Tables*, [*Online*]. Poveznica: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/partition-tables.html; pristupljeno 26. svibnja 2024.

[6] *ESP32 Hardware Reference - Chip Series Comparison*, [*Online*]. Poveznica: https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32/hw-reference/chip-series-comparison.html#; pristupljeno 26. svibnja 2024.

[7] *ESP-IDF API Guides - Memory Types*, [*Online*]. Poveznica: https://docs.espressif.com/projects/esp-idf/en/v5.2.2/esp32/api-guides/memory-types.html; pristupljeno 26. svibnja 2024.

[8] Inamdar, A. *Memory Availability Comparison between ESP32 and ESP32-C3,* (2021, travanj). Poveznica: https://blog.espressif.com/memory-availability-comparison-between-esp32-and-esp32-c3-10f2e65f055e; pristupljeno 26. svibnja 2024.

[9] Čuljak, M. *Wi-Fi provisioning*. Seminar 1. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2023.

[10] *HTTP*, [*Online*]. Poveznica: https://en.wikipedia.org/wiki/HTTP; pristupljeno 29. svibnja 2024.

[11] EMQX Team. *Memory Availability Comparison between ESP32 and ESP32-C3,* (2024, ožujak). Poveznica: https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt; pristupljeno 29. svibnja 2024.

[12] Hermanudin, Aldwin & Ekadiyanto, F. & Sari, Riri. *Performance Evaluation of CoAP Broker and Access Gateway Implementation on Wireless Sensor Network*. TENCONSpring, Sydney, (2018).

[13] *What is AWS IoT?*, [*Online*]. Poveznica: https://docs.aws.amazon.com/iot/latest/developerguide/index.html; pristupljeno 1. lipnja 2024.

[14] *Internet of Things (IoT)*, [*Online*]. Poveznica: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/internet-of-things-services.html; pristupljeno 1. lipnja 2024.

[15] *Provisioning identity in AWS IoT Core for device connections*, [*Online*]. Poveznica: https://docs.aws.amazon.com/whitepapers/latest/device-manufacturing-provisioning/provisioning-identity-in-aws-iot-core-for-device-connections.html; pristupljeno 2. lipnja 2024.

[16] *How to automate onboarding of IoT devices to AWS IoT Core at scale with Fleet Provisioning*, [*Online*]. Poveznica: https://aws.amazon.com/pt/blogs/iot/how-to-automate-onboarding-of-iot-devices-to-aws-iot-core-at-scale-with-fleet-provisioning/; pristupljeno 2. lipnja 2024.

[17] *What is AWS IoT Device Defender?*, [*Online*]. Poveznica: https://docs.aws.amazon.com/iot-device-defender/latest/devguide/index.html; pristupljeno 4. lipnja 2024.

[18] *What is Amazon DynamoDB?*, [*Online*]. Poveznica: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html; pristupljeno 5. lipnja 2024.

[19] Hasan, M. *The IoT cloud: Microsoft Azure vs. AWS vs. Google Cloud,* (2022, veljača). Poveznica: https://iot-analytics.com/iot-cloud/; pristupljeno 10. lipnja 2024.

[20] Pierleoni, Paola & Concetti, Roberto & Belli, Alberto & Palma, Lorenzo. *Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison*. Universita Politecnica delle Marche, Ancona, Italija, (2019, prosinac).

Sažetak

Sustav za udaljeni nadzor pametne kuće temeljen na platformi ESP32-C3 i servisima AWS IoT

U okviru rada proučena je arhitektura ESP32 mikrokontrolera te je provedena usporedba porodica ESP-WROOM-32 i ESP32-C3. Napravljen je pregled važnijih mogućnosti i protokola koji se koriste u realizaciji *IoT* sustava. Proučeni su i objašnjeni važni servisi u sklopu *AWS IoT cloud* platforme: *AWS IoT Jobs (OTA)*, *AWS IoT Device Shadow*, *AWS IoT Fleet Provisioning* i *AWS IoT Device Defender*. Služeći se navedenim *AWS* servisima, razvijen je pokazni sustav temeljen na ESP32(-C3) mikrokontroleru s funkcijom spajanja na lokalnu pristupnu točku, automatskom registracijom novih senzorskih čvorova na *AWS* infrastrukturu, mogućnošću udaljene nadogradnje programske potpore odabranih čvorova te pristupom posljednjem poznatom stanju uređaja. Dodatno, izrađena je pokazna *web* aplikacija s mogućnošću upravljanja *IoT* uređajima i grafičkom vizualizacijom prikupljenih senzorskih podataka dohvaćenih iz *DynamoDB*-a, *AWS cloud* baze podataka.

Ključne riječi: Internet stvari, pametna kuća, senzorska mreža, ESP32, FreeRTOS, AWS IoT

Summary

System for remote management of a smart home based on ESP32-C3 platform and AWS IoT services

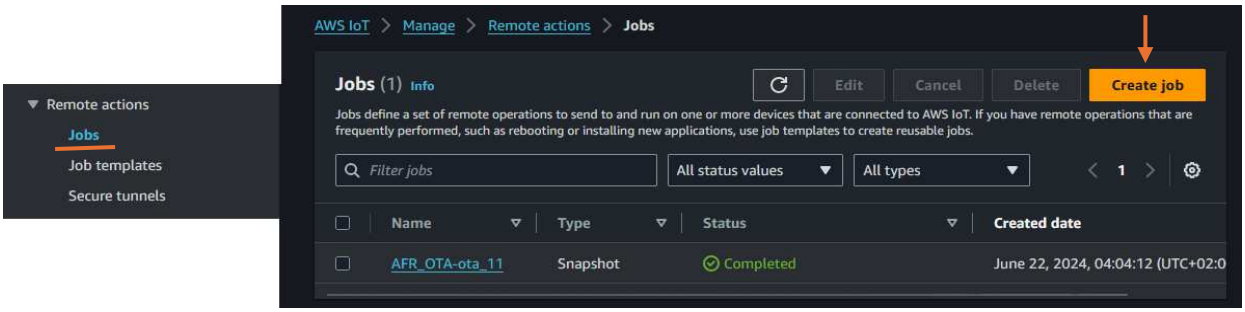
In this thesis the architecture of the ESP32 microcontrollers was presented and the capabilities of ESP WROOM-32 and ESP32-C3 families were compared. An overview of the most important features and protocols used in IoT systems was provided. The most important services within the AWS IoT cloud platform were presented: AWS IoT Jobs (OTA), AWS IoT Device Shadow, AWS IoT Fleet Provisioning, and AWS IoT Device Defender. Using these AWS services, a demonstration system based on ESP32(-C3) microcontroller was developed, with features such as: provisioning a device to a local access point, automatic registration of new sensor nodes on AWS infrastructure, firmware upgrade over-the-air capability for selected nodes, and access to the latest known state of devices. Additionally, a demo web application was developed, with the ability to manage IoT devices and visualize acquired sensor data retrieved from DynamoDB, an AWS cloud database.

Keywords: Internet of Things, smart home, connected sensors, ESP32, FreeRTOS, AWS IoT

Privitak

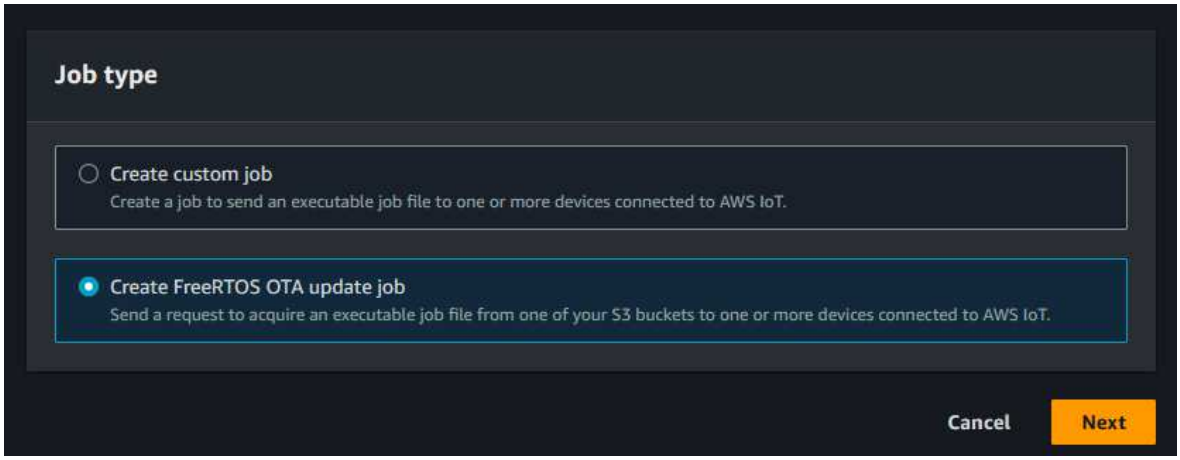
Upute za pokretanje udaljene nadogradnje programske podrške (*FOTA*)

Za nadogradnju programske podrške pomoću *AWS IoT Jobs/OTA* servisa potrebno je slijediti sljedeće upute. Na zaslonu prikazanom slikom 0.1 odabrati *„Jobs“* odjeljak te *„Create job“* tipku.



Slika 0.1 Kreiranje novog *OTA* zadatka

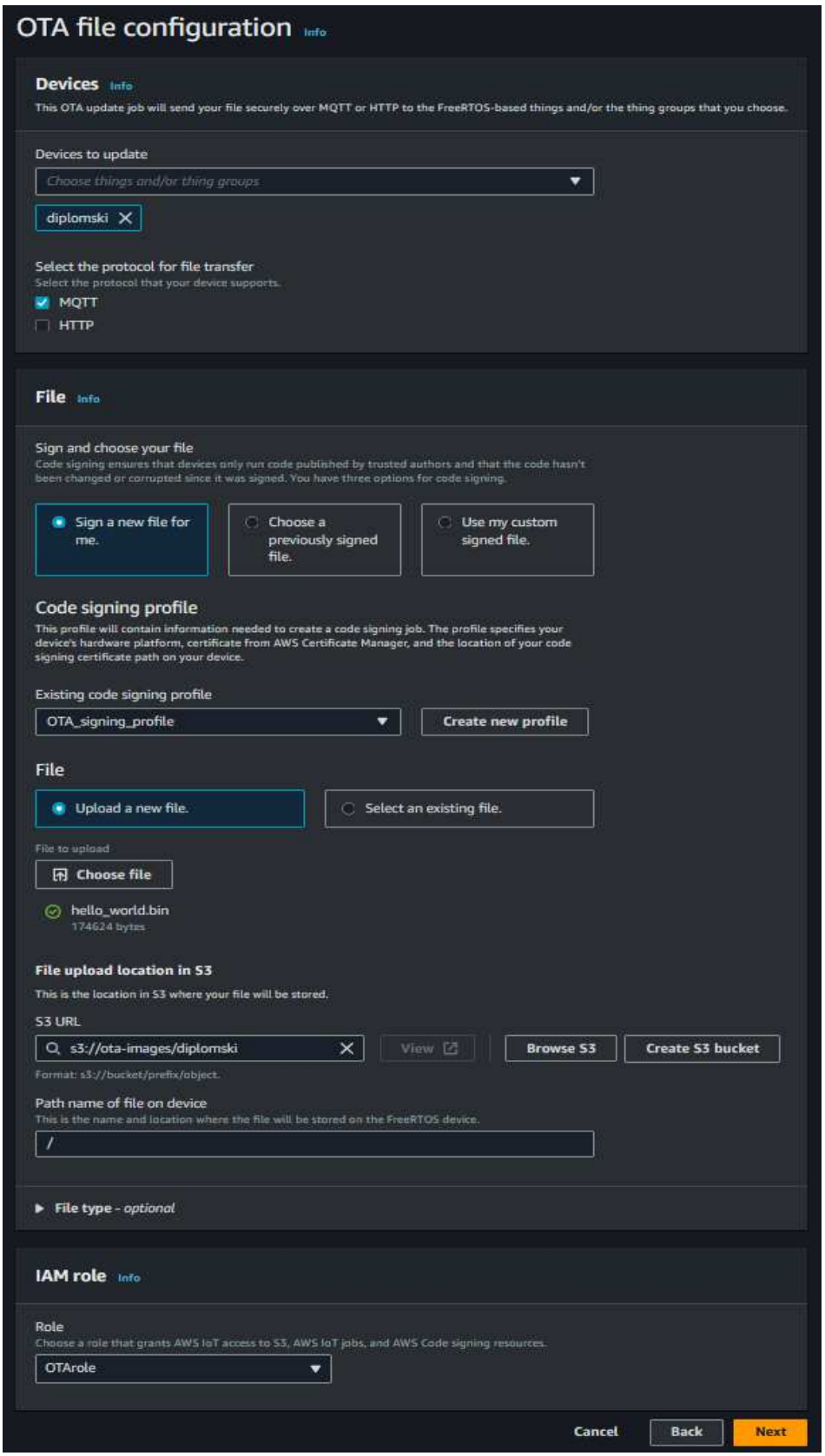
Kao vrstu zadatka odabrati *„FreeRTOS OTA Update job“* (slika 0.2) te na sljedećoj stranici unijeti proizvoljni naziv zadatka.



Slika 0.2 Odabir vrste zadatka

Glavna stranica konfiguracije *OTA* zadatka prikazana je slikom 0.3. Potrebno je odabrati „stvari“ ili grupu „stvari“ nad kojom se želi izvesti nadogradnja programske podrške. Zatim odabrati *MQTT* protokol i za potpisivanje programske podrške odabrati jednu od ponuđenih opcija. Prvi puta će biti potrebno odabrati *„Sign a new file for me“*. Odabrati kreiran profil potpisivanja programske podrške i učitati novu „sliku“ u *.bin* formatu. Za parametar *„Path name of file on device“* dovoljno je staviti „/“, a za ulogu odabrati ulogu prikladnu *OTA* postupku. Naposljetku, potrebno je izabrati *snapshot* ili *continuous* vrstu zadatka objašnjenu

u poglavlju 4.5.1 te pritisnuti tipku za kreiranje zadatka, čime će se izvođenje zadatka pokrenuti.



Slika 0.3 Glavna konfiguracija *OTA* zadatka