






DOKUMENTACIJA PROJEKTA IZ UGRADBENIH SISTEMA




Smart Home

Taner BAJROVIĆ, Edwin GRACA, Ajna MUJKIĆ

Sadržaj

1	Uvod	1
2	Dijagram stanja	1
3	Kod	1
3.1	 AlarmState	1
3.2	 Keypad	2
3.3	 Mqtt	2
3.4	 Main	3
3.5	 Buzzer	5

Korištene oznake

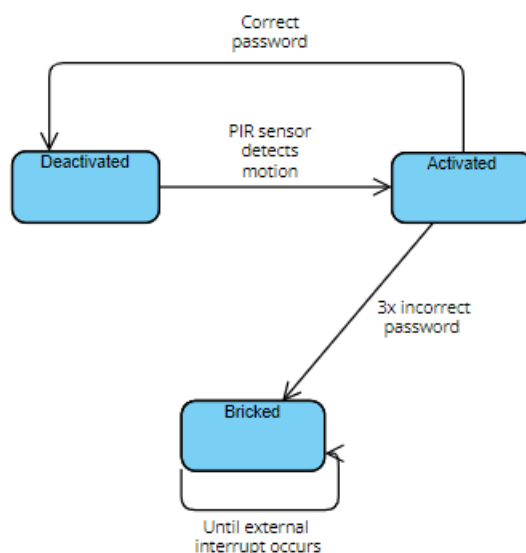
 Klasa	Oznaka za klasu
 Modul	Oznaka za modul
host/tema	Oznaka za temu
 Stanje	Oznaka za stanje

1 Uvod

Kao projekat na predmetu ‘Ugradbeni Sistemi’ implementirali smo alarm koji se može integrirati u neki Smart house sistem. Sklop omogućava da se određeni prostor pokrije senzorima za detekciju pokreta, aktivira/deaktivira alarm te unese lozinka za njegovo isključivanje. Unos lozinke se izvršava putem displeja i tastature. Ako je detektovano kretanje u zaštićenoj zoni, vlasniku se šalje poruka na mobitel putem MQTT protokola, a on može i isključiti alarm putem aplikacije koja podržava MQTT.

2 Dijagram stanja

Prvobitno je alarm u stanju `Deactivated`. Ako se osoba pojavi u zaštićenoj zoni aktivira se alarm i sistem prelazi u stanje `Activated` i u njemu ostaje sve dok korisnik ima preostalih pokušaja za unos lozinke. Ukoliko se netačna lozinka unese 3 puta sistem ulazi u stanje `Bricked` i u njemu ostaje dok ima napajanja.



Slika 1. State diagram

3 Kod

3.1 `AlarmState`

Klasa `AlarmState`, koja se nalazi unutar modula `Alarm`, sadrži metode za prijelaze između različitih stanja u kojima se sistem može nalaziti. U nastavku ćemo ukratko proći kroz metode ove klase te objasniti ulogu svake:

```
hasCode(self)-> bool
```

Provjerava da li je postavljena lozinka.

```
isLocked(self)-> bool
```

Provjerava da li je alarm uključen.

```
unlock(self, code: str)
```

Otključava, odnosno deaktivira alarm. Nakon poziva ove metode, objekat klase `cls AlarmState` se nalazi u `Deactivated` stanju.

```
lock(self)
```

Zaključava, odnosno aktivira alarm. Nakon poziva ove metode klase, objekat `cls AlarmState` se nalazi u `Activated` stanju.

```
setCode(self, newCode: str)
```

Postavlja novu lozinku koja je proslijeđena kao argument `newCode`.

Konstruktor ima sljedeći oblik:

```
__init__(self, locked: bool)
```

Konstruktor inicijalizira `cls AlarmState` objekat sa specificiranim stanjem u kojem se alarm već nalazi putem `locked` argumenta.

3.2 `MDL Keypad`

Modul `MDL Keypad` se koristi za očitavanje unesenog karaktera sa tastature. Njena jedina metoda je `Keypad4x4Read(cols, rows)`, prikazana u nastavku:

```
Keypad4x4Read(cols, rows)-> str
```

Metoda čita unos sa tastature te vraća uneseni karakter koji je tipa `str`. Implementacijom debouncing-a smo osigurali da se uvijek vrati tačno jedan karakter za svaki pritisak na taster.

3.3 `MDL Mqtt`

Modul `MDL Mqtt` uspostavlja Wi-Fi konekciju, povezuje se na MQTT broker i upravlja buzzer-om na osnovu MQTT poruka. Ovaj modul omogućava semi-duplex komunikaciju koristeći sljedeće teme:

- `ushome/alarmCommands` Primanje kontrolnih komandi

- `ushome/alarmNotifications` Objavljivanje notifikacija

Navedeni modul implementira sljedeće metode:

`sub()`

Poziva se po prijemu MQTT poruke. Upravlja porukama koje dolaze sa teme `ushome/alarmCommands`. Ako poruka glasi 'Stop beeping', duty cycle buzzer-a se postavlja na 0. U suprotnom se ispisuje primljena poruka.

`publish()`

Objavljuje string koji je proslijeđen kao parametar na temu `ushome/alarmNotifications`.

3.4 `MDL Main`

U `MDL Main` modulu se odvija glavna logika programa. U ovom dijelu su definirane metode za različite izgled displeja, te za specificiranje ponašanja alarma ovisno da li se nalazi u `Activated`, `Deactivated`, ili `Bricked` stanju.

`motionDetected()`

Prekidna rutina koja upravlja ponašanjem PIR senzora. Poziva se kada se detektuje pokret. Ako je alarm u stanju `Activated`, aktivira `cs Buzzer` i šalje notifikaciju putem MQTT.

`displayMessageSlowly(message, duration=0.1)`

Ispisuje poruku na TFT displej karakter po karakter uz definiranu pauzu između karaktera.

`showStartupMessage()`

Ispisuje prvu poruku na ekranu koja se sastoji od pozdrava, naziva i verzije uređaja.

`showWaitScreen()`

Ispisuje 'loading' ekran na displej.

`showUnlockedScreen()`

Upravlja logikom za slučaj kada alarm nije aktivan. Ako lozinka nije postavljena, od korisnika se zatraži postavljanje te potvrda lozinke, a u suprotnom se zatraži aktivacija alarma.

showLockedScreen()

Upravlja logikom za slučaj kada je alarm u `Activated` stanju. Od korisnika traži deaktivaciju alarma unosom ispravne lozinke.

showBrickedScreen()

Ispisuje poruku da je alarm u `Bricked`, te aktivira karakterističan zvuk na buzzer-u.

beginCountdown(duration)

Ipisuje poruku o čekanju i odbrojava onoliko vremena koliko je navedeno u parametru funkcije.

inputSecretCode()

Čita karaktere koje korisnik unosi na tastaturi, spaja ih u string i vraća tako formirani string.

setNewCode()

Traži od korisnika da unese i potvrdi novu lozinku. Ako su obje unesene lozinke identične, ažurira postojeću lozinku i ispisuje poruku o uspješnoj izmjeni. U suprotnom, ispisuje poruku o grešci.

alarmLockedLogic()

Implementira logiku za slučaj alarma u `Activated` stanju. Traži od korisnika unos lozinke, provjerava njenu ispravnost, te zavisno od nje ili otključava alarm ili okida blokiranje alarma s obzirom na broj preostalih pokušaja unosa lozinke. U nastavku je prikazan kod koji implementira logiku aktivnog alarma:

```
# Alarm logic when locked (turned on).
def alarmLockedLogic() -> None:
    global attemptsLeft, signalSent
    secretCode = inputSecretCode()
    unlockedSuccessfully = alarmState.unlock(
        secretCode)

    if unlockedSuccessfully:
        buzzer.stopBeeping()
        signalSent = False
        showUnlockedScreen()
        attemptsLeft = 3
    else:
```

```

publish("Incorrect password entered")
attemptsLeft -= 1
if attemptsLeft == 0:
    showBrickedScreen()
display.erase()
display.set_pos(LEFT_MARGIN, TOP_MARGIN + 40)
display.print(f"Incorrect password. You have
               {attemptsLeft} attempt(s) left.")

```

alarmUnlockedLogic()

Implementira logiku za slučaj alarma u `Deactivated` stanju. Čita unos sa tastature i vrši akcije u ovisnosti od unesenog karaktera (# ili *) - aktivacija alarma ili postavljanje nove lozinke, respektivno. U nastavku je prikazan kod koji implementira logiku deaktiviranog alarma:

```

# Alarm logic when unlocked (turned off).
def alarmUnlockedLogic() -> None:
    key = Keypad4x4Read(col_list, row_list)
    if key == "#":
        beginCountdown(10)
        alarmState.lock()
        showLockedScreen()
    elif key == "*":
        setNewCode()

```

3.5 `cls` Buzzer

Klasa `cls` Buzzer nudi interfejs za upravljanje buzzer komponentom koristeći širinsko-impulsnu modulaciju (PWM). Omogućava postavljanje duty cycle-a i frekvencije PWM signala u svrhu kontrolisanja zvuka koji proizvodi buzzer. Ovo je postignuto sljedećim statičkim atributima, čije vrijednosti odgovaraju duty-cycle vrijednosti:

- LOW = 0
- MID = 32000
- HIGH = 64000

Klasa implementira sljedeće metode:

```
setDuty(self, pin_number: int)
```

Postavlja duty cycle buzzer-a na vrijednost specificiranu parametrom.

setFrequency()

Postavlja frekvenciju buzzer-a na vrijednost specificiranu parametrom.

stopBeeping()

Postavlja duty cycle buzzer-a na minimalnu vrijednost.

startBeeping()


Postavlja duty cycle buzzer-a na neku srednju vrijednost definiranu konstantom iz globalnog opsega.

startScreaming()

Postavlja duty cycle buzzer-a na maksimalnu vrijednost.

Konstruktor klase ima sljedeći oblik:

`__init__(self, pin_number: int)`

Konstruktor inicijalizira  Buzzer objekat sa specificiranim brojem pin-a na koji je buzzer spojen. Također postavlja inicijalnu frekvenciju na 400Hz i duty cycle na minimalnu vrijednost.