**Laborator 13 - IoT – documentatie**

**Aplicatia propusa**

Creati, utilizând Python si Flask, o aplicatie noua astfel încât sa introduceti suport pentru autentificare si autorizare pe roluri, astfel:

* **guest/anonim**
  + este rolul implicit, alocat oricarui utilizator care nu a trecut de pasul de *autentificare*
  + acest rol nu are nici un drept (nu poate citi datele unui senzor, nu poate crea/actualiza parametri de configurare ai senzorului respectiv)
* **owner**
  + rolul se asigneaza utilizatorului care poate citi datele unui senzor, dar nu poate crea/actualiza configurarile acestuia
* **admin**
  + rolul cel mai „puternic” din sistem
  + acest rol poate citi datele unui senzor si poate crea/actualiza configurarile respective

Aceasta aplicatie va rula pe un dispozitiv extern ESP32 (calculatorul de lucru din laborator, laptopul personal, etc.). Pentru a implementa aceasta functionalitate, este necesar sa creati o noua resursa virtuala. Aceasta resursa poate fi expusa sub forma unui URL relativ semnificativ, de exemplu http://localhost/auth. Maparea metodelor HTTP si semnificatia acestora este:

* POST/auth DELETE/auth/jwtStore
  + realizeaza functionalitatea de tip login
  + INPUT: nume de utilizator si parola, transmise în reprezentare JSON în corpul de mesaj
  + OUTPUT: token-ul JWT creat cu succes, daca numele de utilizator si parola sunt validate cu succes; mesaj de eroare altfel
* GET/auth/jwtStore
  + verifica valabilitatea unui token
  + INPUT:o cerere directa, simpla, fara corp de mesaj
  + token-ul JWT care trebuie validat va fi inclus în header-ul Authorization
  + OUTPUT: raspuns 200 OK daca token-ul este valid si corpul de mesaj va include rolul utilizatorului valid; 404 Not Found daca token-ul nu este regasit în sistem, eroare generica altfel (token invalid)
* DELETE/auth/jwtStore
  + realizeaza functionalitatea de tip logout prin invalidarea token-ului
  + INPUT:o cerere directa, simpla, fara corp de mesaj; token-ul JWT care trebuie validat va fi inclus în header-ul Authorization
  + Nota: !!! abordarea propusa nu este una 100% standard, dar ofera avantajul simplitatii!!
  + OUTPUT: raspuns 200 OK daca token-ul este valid, 404 Not Found daca token-ul nu este re gasit în sistem, eroare generica altfel (token invalid)

Pentru a gestiona token-urile JWT în cadrul unei aplicatii Python, se poate utiliza pachetul Flask-JWT-Extended. Pentru simplitate, lista de utilizatori (nume de utilizator, parola de acces si rol) poate fi mentinuta în memorie, într-un dictionar, dupa modelul urmator:

|  |  |  |
| --- | --- | --- |
| username | password | role |
| User1 | Parola1 | Admin |
| User2 | Parola2 | Owner |
| User3 | Parola3 | Owner |

Suita de servicii RESTful implementata pe ESP32 trebuie modificata astfel încât fiecare *endpoint* protejat la acces va localiza si extrage valoarea *header*-ului HTTP Authorization:

* daca acest *header* nu exista sau daca include o valoare eronata (i.e. JWT-ul nu este marcat ca fiind de tip Bearer), atunci utilizatorul va fi redirectat catre resursa de autentificare
* daca se localizeaza un JWT, atunci se va invoca serviciul RESTful python pentru validarea token-ului si pentru determinarea rolului utilizatorului curent:
  + daca rolul permite realizarea actiunii, atunci se va oferi raspuns utilizatorului
  + daca rolul nu permite realizarea actiunii sau token-ul este invalid, atunci utilizatorul va fi redirectat catre resursa de autentificare

**Rezolvare**

**Structura proiect**

proiect /

│

├── flask\_authetification\_server/        ← Serverul Flask

│   ├── app.py                ← Codul principal Flask

│   ├── users.py              ← Dicționarul cu utilizatori

│   ├── requirements.txt      ← Dependențe (Flask, flask-jwt-extended)

| ├── static/

│ └── index.html ← Pagina HTML

│

├── esp\_client/             ← Codul ESP32

│   ├── lib/

│   ├── src/

│   │   └── main.cpp          ← Codul principal ESP32

│   ├── platformio.ini        ← Configurare proiect PlatformIO

**Aplicatia Flask**

1. Autentificare utilizator (POST /auth)

Acest endpoint permite autentificarea unui utilizator pe baza unui username și a unei parole transmise în format JSON și returnează un token JWT în cazul validării cu succes.

* Valid input

A screenshot of a login form

AI-generated content may be incorrect.

* Invalid input

A screenshot of a login screen

AI-generated content may be incorrect.

1. Validare token (GET /auth/jwtStore)

Acest endpoint verifică validitatea unui token JWT trimis în header-ul Authorization și, dacă tokenul este valid, returnează rolul și numele utilizatorului asociat.

* Valid input

A screenshot of a computer

AI-generated content may be incorrect.

* Invalid input

A screenshot of a computer

AI-generated content may be incorrect.

1. Logout / Invalidează token (DELETE /auth/jwtStore)

Acest endpoint permite utilizatorului autentificat să se delogheze prin invalidarea tokenului JWT curent, eliminându-l din sistemul de stocare temporară a tokenurilor active.

* Valid input

A screenshot of a computer

AI-generated content may be incorrect.

* Invalid input

A screenshot of a computer

AI-generated content may be incorrect.

Dupa ce am sters tokenul pentru utilizatorul respectiv, am incercat sa il validez din nou, dar validarea a esuat, ceea ce inseamna ca stergerea tokenului de mai sus a avut loc cu succes.

A screenshot of a computer

AI-generated content may be incorrect.

**Aplicatia ESP**

Programul din ESP32 se conectează la o rețea WiFi, face autentificare către un serverul Flask pentru a obține un token JWT, apoi trimite date despre un senzor PIR (care este conectat la placuta) la server, folosind tokenul obținut.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.