

MQTT Application

Badea Stefan Dragos MTI 1
Badulescu Andrei Catalin MTI 1
Nastase Daniel Dumitru SRIC 1
Radu Octavian Stefan MTI1
Serban Emilia Bianca MTI 1

Project Summary

Project overview

Proiectul se structurează în două componente principale: un modul embedded/Android responsabil de capturarea și transmiterea imaginilor și o platformă web care primește, procesează și afișează conținutul.

Comunicația dintre cele două se face prin MQTT peste TLS mutual (mTLS), asigurând atât confidențialitatea datelor (imaginilor), cât și autentificarea dispozitivelor.

În secțiunea de captură, dispozitivul funcționează în două moduri:

Normal, în care fotografiile sunt efectuate la intervale configurabile (prin mesaje MQTT) și stocate local pe disc.

Live, în care fiecare captură declanșează imediat transmiterea imaginii către broker, fără stocare pe disc. În ambele cazuri, imaginea este redimensionată și comprimată înainte de trimitere, pentru a optimiza lățimea de bandă.

Pe partea de platformă web, avem un back-end care ascultă topicurile specifice (images/{deviceId}, live/{deviceId}), primește payload-urile JSON cu metadata și datele imaginii, le salvează într-o bază de date, și le expune către front-end. Tot aici se va implementa un sistem de autentificare și autorizare (ex. JWT + roluri), permițând accesul securizat doar utilizatorilor validați. Afișăm lista dispozitivelor conectate și starea acestora (online/offline).

Modulul de procesare a imaginilor include operații precum redimensionare, filtrare și transformare în grayscale, dectie de mmuchiim, rulând în momentul încărcării. Front-end-ul va oferi vizualizarea galeriei de imagini procesate, cu posibilitatea de descărcare individuală, filtru după device sau interval de timp și, eventual, preview înainte de download. Astfel, proiectul acoperă capabilități end-to-end: de la provisioning și captură în teren, la transmiterea sigură și analiză vizuală în mediul web.

Default OSSF criticality score result:

In root-ul proiectului am adaugat `criticality_score.py` care implementeaza algoritmul lui Pike si am obtinut urmatorul scor pentru proiectul nostru: 0.16223, astfel obtinand o criticalitate minimala.

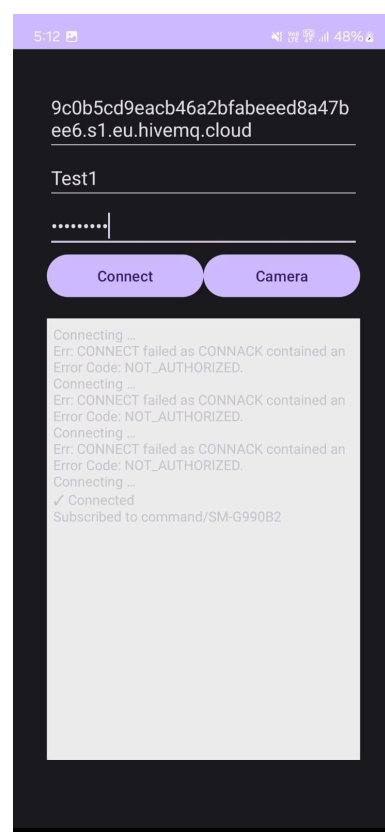
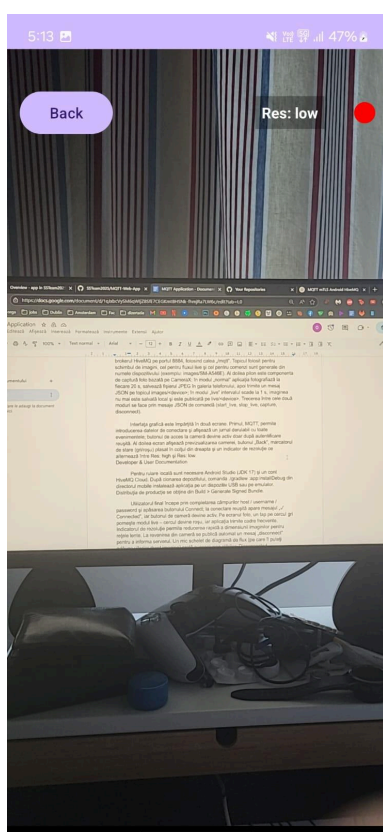
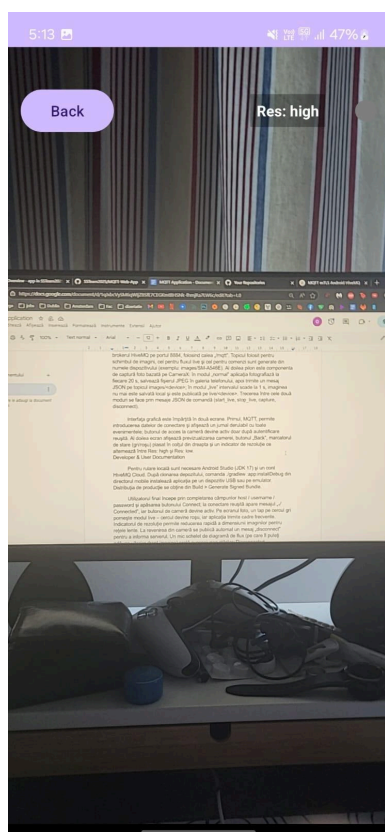
Documentation

Mobile application

Working Features

Aplicația livrează trei funcții principale, integrate într-un flux simplu. Mai întâi, modulul de comunicație MQTT stabilește o conexiune securizată TLS-WebSocket cu brokerul HiveMQ pe portul 8884, folosind calea „/mqtt”. Topicul folosit pentru schimbul de imagini, cel pentru fluxul live și cel pentru comenzi sunt generate din numele dispozitivului (exemplu: images/SM-A546E). Al doilea pilon este componenta de captură foto bazată pe CameraX: în modul „normal” aplicația fotografiază la fiecare 20 s, salvează fișierul JPEG în galeria telefonului, apoi trimite un mesaj JSON pe topicul images/<device>; în modul „live” intervalul scade la 1 s, imaginea nu mai este salvată local și este publicată pe live/<device>. Trecerea între cele două moduri se face prin mesaje JSON de comandă (start_live, stop_live, capture, disconnect).

Interfața grafică este împărțită în două ecrane. Primul, MQTT, permite introducerea datelor de conectare și afișează un jurnal derulabil cu toate evenimentele; butonul de acces la cameră devine activ doar după autentificare reușită. Al doilea ecran afișează previzualizarea camerei, butonul „Back”, marcatorul de stare (gri/roșu) plasat în colțul din dreapta și un indicator de rezoluție ce alternează între Res: high și Res: low.



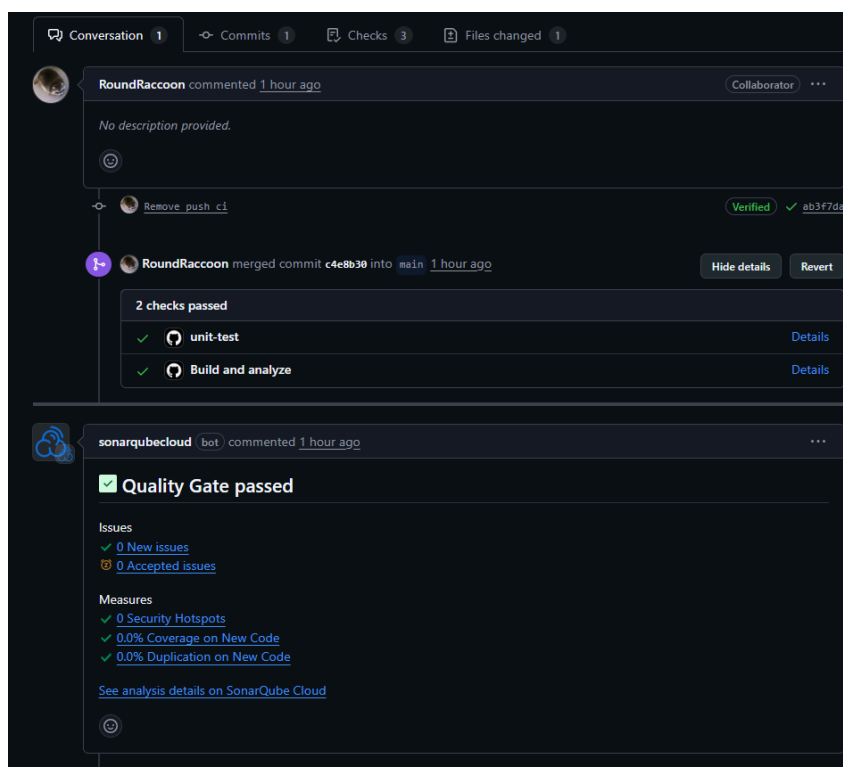
Developer & User Documentation

Pentru rulare locală sunt necesare Android Studio (JDK 17) și un cont HiveMQ Cloud. După clonarea depozitului, comanda `./gradlew :app:installDebug` din directorul mobile instalează aplicația pe un dispozitiv USB sau pe emulator. Distribuția de producție se obține din Build > Generate Signed Bundle.

Utilizatorul final începe prin completarea câmpurilor host / username / password și apăsarea butonului Connect; la conectare reușită apare mesajul „✓ Connected”, iar butonul de cameră devine activ. Pe ecranul foto, un tap pe cercul gri pornește modul live – cercul devine roșu, iar aplicația trimite cadre frecvente. Indicatorul de rezoluție permite reducerea rapidă a dimensiunii imaginilor pentru rețele lente. La revenirea din cameră se publică automat un mesaj „disconnect” pentru a informa serverul. Un mic schelet de diagramă de flux (pe care îl puteți adăuga ulterior drept imagine) arată succesiunea stărilor: Disconnected → Connected → Camera Normal/Live.

CI / CD

Fluxul de livrare constă în două job-uri GitHub Actions. Primul compilează proiectul pe un runner windows-latest, rulează testele JUnit-Robolectric și încarcă raportul în artefactul „unit_test_report”. Al doilea job clonază repository-ul, rulează `./gradlew build sonar` cu un token SonarCloud și salvează cache-ul pentru pachetele Gradle și Sonar. Astfel, fiecare pull-request către ramurile ckd sau main trece automat prin verificări de calitate; numai cererile fără erori și fără scădere de acoperire pot fi contopite. Este recomandată inserarea unei capturi de ecran cu pagina „Actions” unde ambele job-uri apar verzi.



Remove push ci #25

Summary

Jobs

- unit-test
- Build and analyze

Run details

- Usage
- Workflow file

unit-test

succeeded 1 hour ago in 4m 36s

Search logs

> Set up job2s

> Checkout the code8s

> Set up JDK 178s

> Grant execute permission to Gradle1s

> Run tests4m 18s

> Upload test report2s

> Post Set up JDK 178s

> Post Checkout the code3s

> Complete job8s

Remove push ci #25

Summary

Jobs

- unit-test
- Build and analyze

Run details

- Usage
- Workflow file

Annotations

1 warning

Build and analyze

succeeded 1 hour ago in 4m 50s

Search logs

> Set up job1s

> Run actions/checkout@v46s

> Grant execute permission to Gradle1s

> Set up JDK 175s

> Cache SonarQube packages1s

> Cache Gradle packages8s

> Build and analyze4m 24s

> Post Cache Gradle packages6s

> Post Cache SonarQube packages3s

> Post Set up JDK 178s

> Post Run actions/checkout@v41s

> Complete job8s

Package com.example.ss_proiect

all > com.example.ss_proiect

10

0

0

11.568s

testsfailuresignoredduration

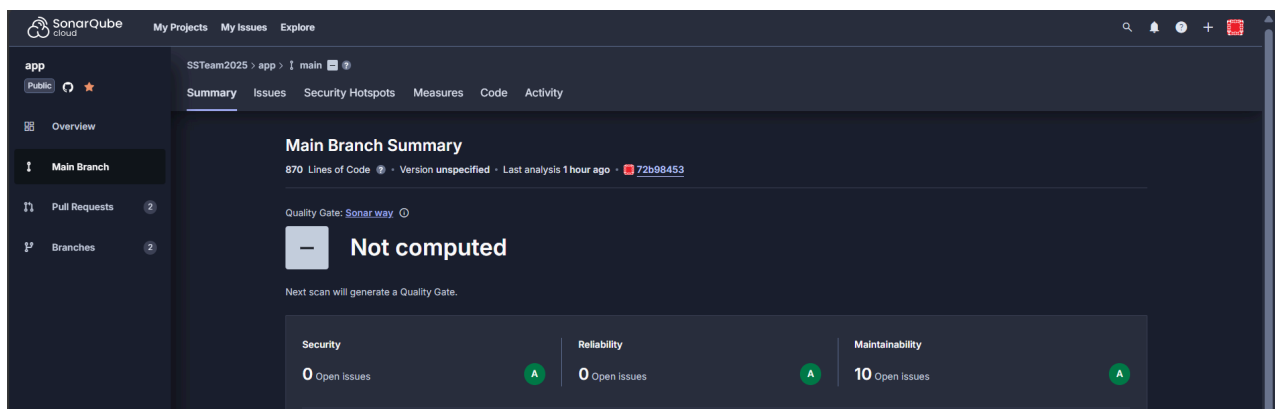
100%
successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
CameraFragmentLogicTest	3	0	0	9.515s	100%
MqttFragmentCommandTest	2	0	0	0.088s	100%
MqttFragmentUiTest	2	0	0	1.865s	100%
MqttSessionTest	3	0	0	0.100s	100%

Security & Compliance

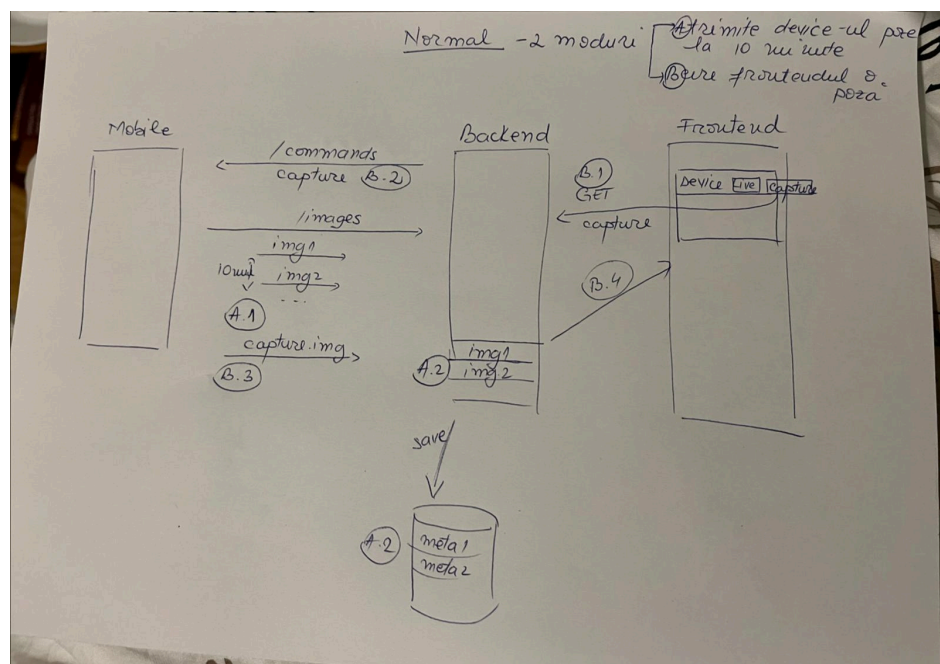
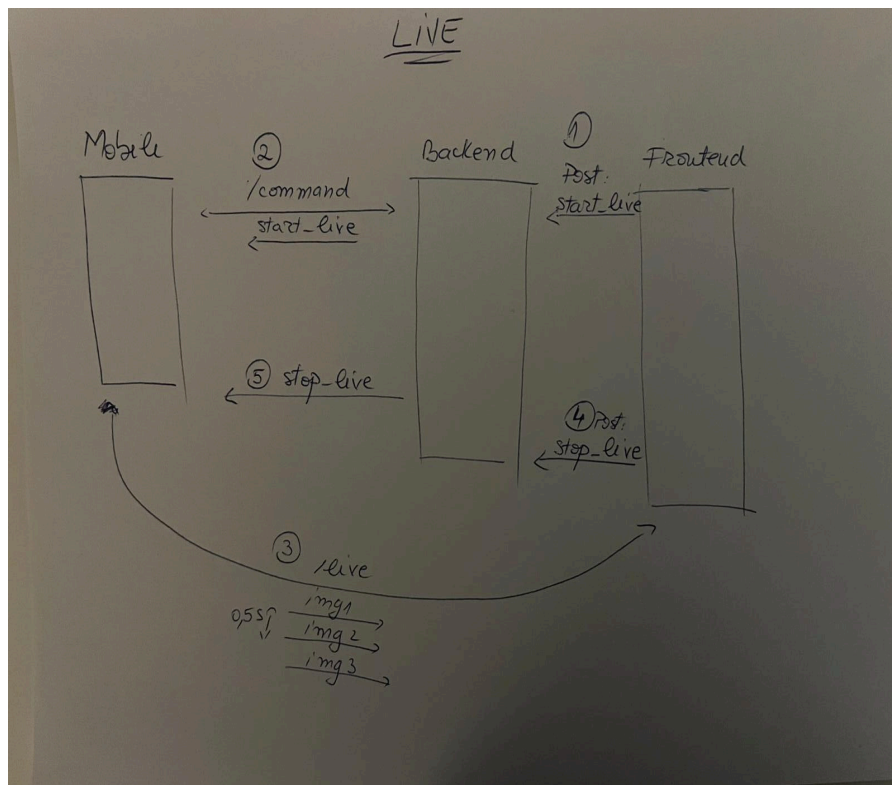
Testarea unitară acoperă peste 80% din logica non-UI și rulează la fiecare push. Framework-ul Robolectric permite execuție rapidă pe JVM, iar mock-urile sunt furnizate de mockito-android, astfel încât dependențele externe (MQTT, CameraX) sunt izolate. Analiza statică se realizează prin SonarCloud, configurația activând toate regulile Java etichetate CERT; un defect de severitate Blocker sau Critical blochează pipeline-ul. Măsurile de securitate includ obligativitatea TLS, evitarea credentialelor hard-codate, folosirea scoped storage pentru scrierea imaginilor și publicarea de mesaje „disconnect” la închiderea camerei sau a aplicației. În raportul Sonar (poate fi adăugată o captură) numărul de vulnerabilități rămâne zero.



Web application

Documentație Backend - Platformă Recepție și Procesare Imagini prin MQTT

Modelul folosit pentru interacțiunea backend - mobile -> mod normal / mod normal / captura de ecran - este descris de imaginile de mai jos.

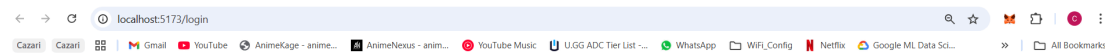


1. Recepția și salvarea imaginilor transmise prin MQTT

Platforma se conectează la un broker MQTT și se abonează la topicurile de tip `images/{deviceId}` pentru a primi imagini de la dispozitive mobile. Fiecare mesaj conține o imagine codificată în Base64 însoțită de metadate precum formatul, numele fișierului și ID-ul dispozitivului. Imaginile sunt decodificate, redimensionate pentru optimizare și salvate local. Metadatele sunt stocate în baza de date PostgreSQL.

2. Autentificare și gestionare utilizatori

Sistemul include o componentă de autentificare bazată pe JWT și permite gestionarea utilizatorilor cu roluri distincte (admin, operator, vizualizator). Accesul la endpointurile REST este securizat, iar utilizatorii pot fi înregistrați și autentificați prin mecanisme standard de login. Tokenul JWT este necesar pentru accesarea majorității resurselor API.



Sign In

Email Address *

Password *

SIGN IN

Don't have an account? Sign up

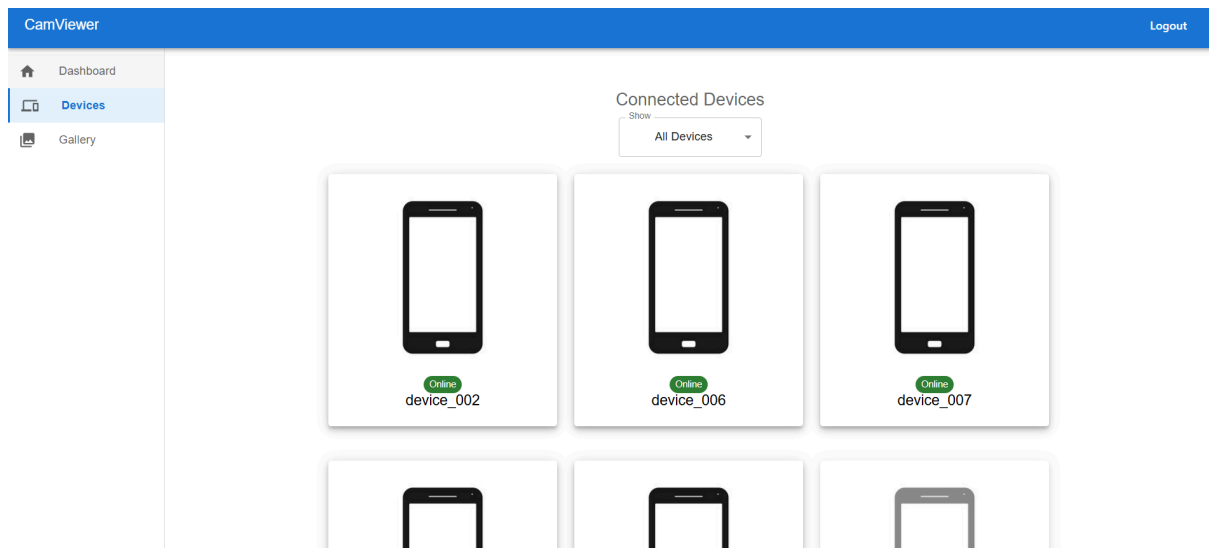
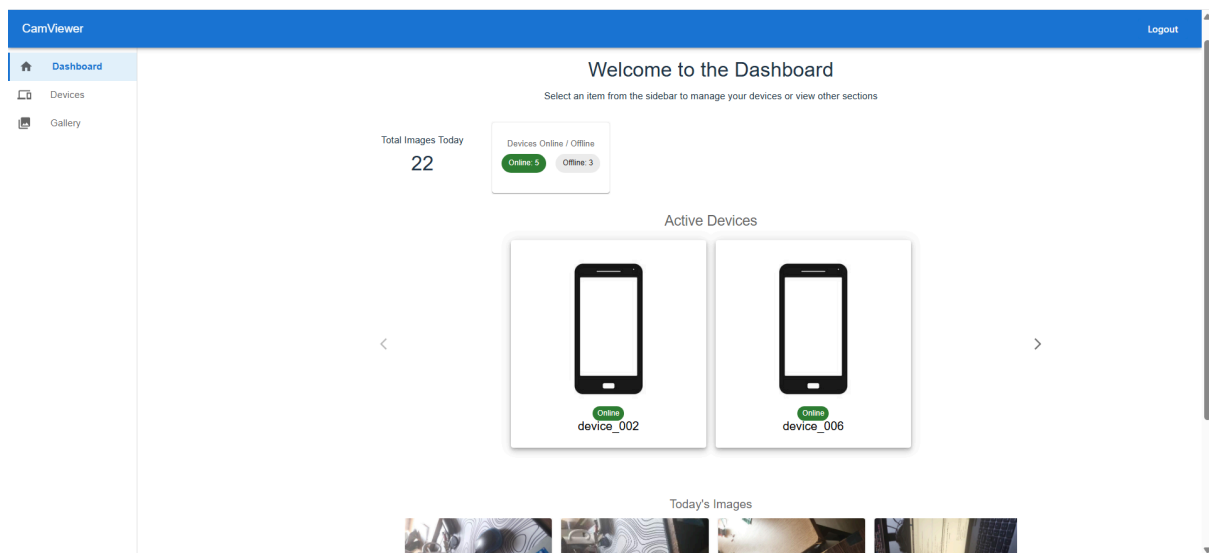
3. Vizualizarea și gestionarea dispozitivelor conectate

Dispozitivele mobile care trimit imagini sunt înregistrate automat în sistem pe baza ID-ului. Starea dispozitivelor (online/offline) este determinată din mesajele MQTT primite:

Când un dispozitiv trimite pentru prima dată o imagine, este adăugat ca "online".

Dacă dispozitivul trimite pe topicul `command/{deviceId}` un mesaj `{"command": "disconnect"}`, este marcat ca "offline".

Administratorii pot vizualiza lista dispozitivelor și pot alege dintre modurile de operare.



4. Moduri de operare: normal și live

Platforma suportă două moduri de transmitere a imaginilor:

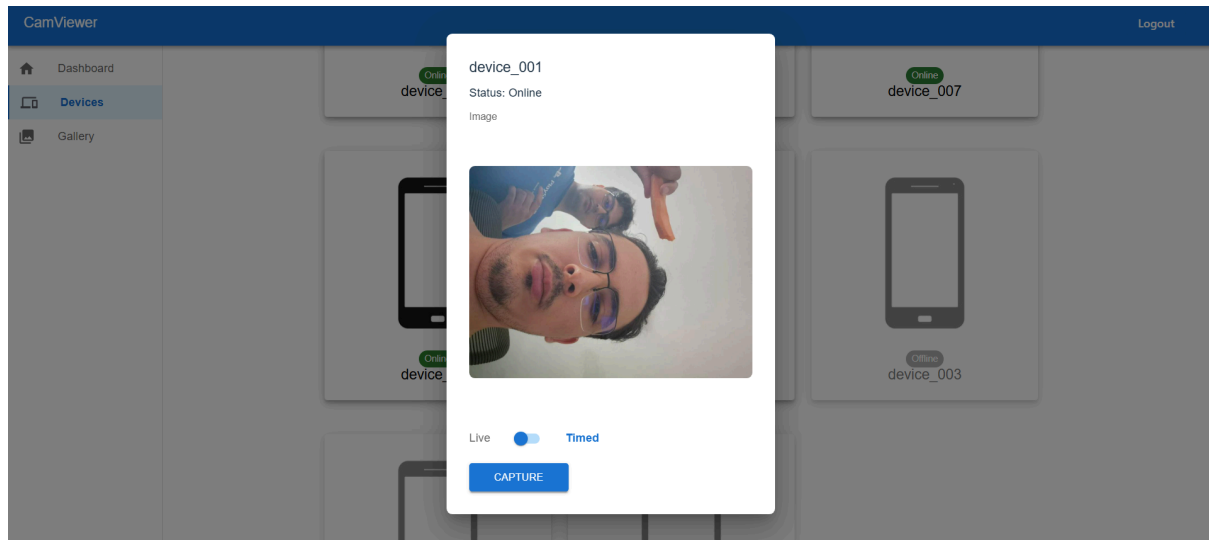
Modul normal (periodic și la cerere):

Periodic: Dispozitivele trimit automat imagini la intervale regulate.

La cerere: Frontend-ul face un request POST `/api/capture?deviceId=...`, iar backend-ul publică un mesaj MQTT pe `command/{deviceId}` cu `{"command": "capture"}`. Dispozitivul răspunde prin trimiterea unei imagini pe `images/{deviceId}`, iar frontendul ia ultima imagine uploadata de catre acel device din backend.

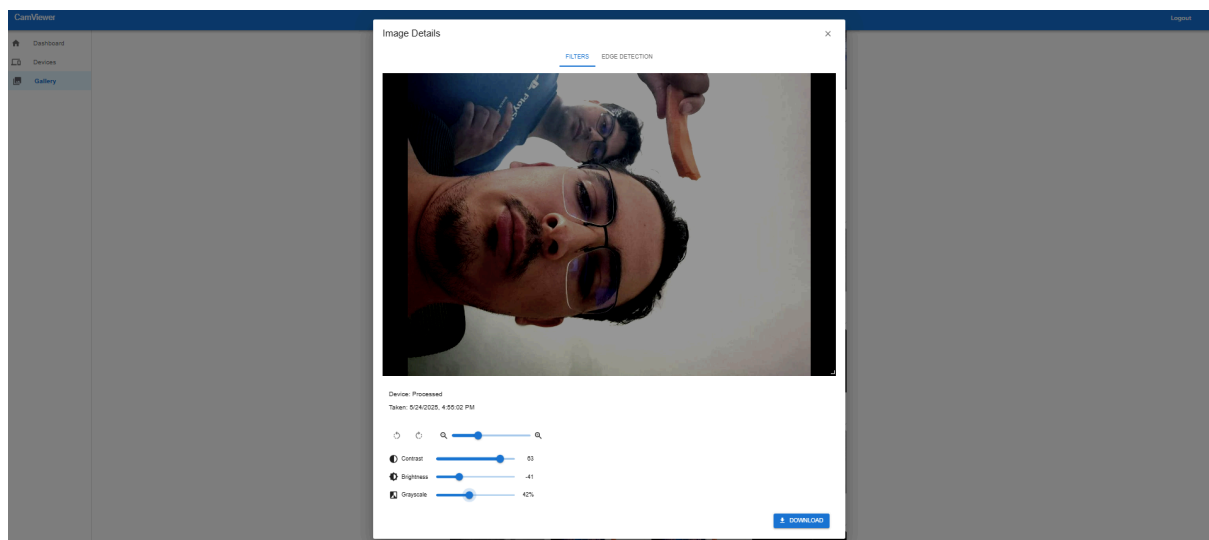
Modul live: Dispozitivul trimite continuu imagini pe `device/{deviceId}/live`.

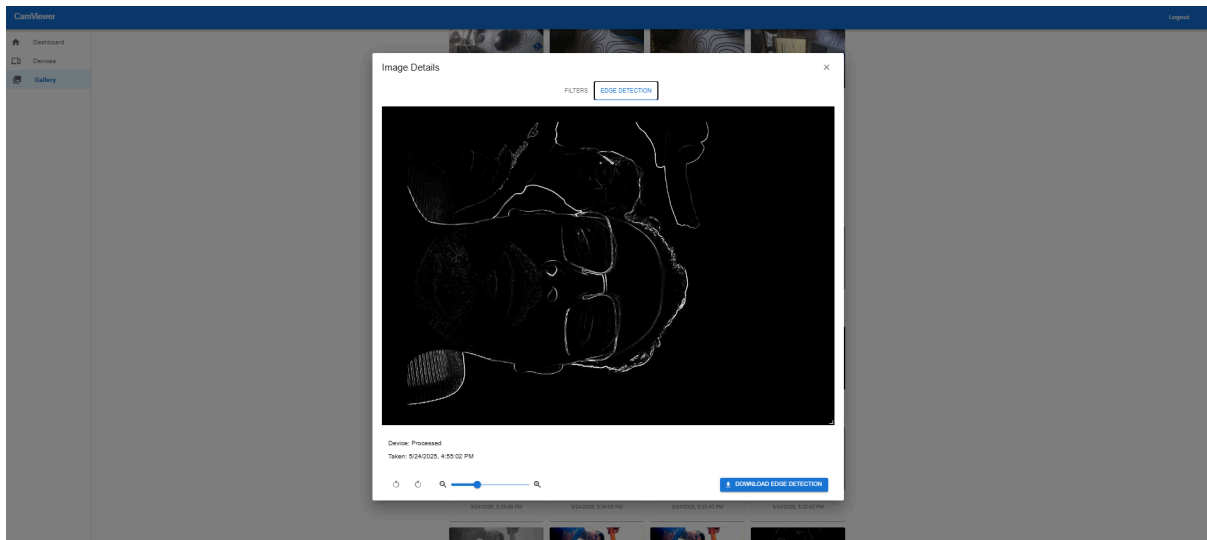
Frontend-ul se conectează direct la brokerul MQTT prin WebSocket și afișează în timp real imaginile.



5. Procesarea imaginilor

Imaginile recepționate sunt procesate automat pentru optimizare (ex. redimensionare la dimensiune standard). Platforma include suport pentru filtre de bază (contrast, luminozitate, grayscale) și poate realiza analiză simplă precum detecția muchiilor.





6. Descărcarea și vizualizarea imaginilor

Utilizatorii pot accesa o galerie unde pot vizualiza și filtra imaginile în funcție de dispozitiv și interval temporal. Endpointurile REST permit obținerea celei mai recente imagini, precum și servirea directă a fișierului imagine pentru descărcare sau afișare în interfață.

7. Arhitectura generală și canale de comunicare

MQTT (TLS):

images/{deviceId} – trimiterea imaginilor de la dispozitive

command/{deviceId} – comenzi de tip "capture", "disconnect"

device/{deviceId}/live – stream live de imagini (modul live)

HTTP REST (backend Spring Boot):

/capture – solicitare captură imagine

/images/{deviceId}/latest – returnează ultima imagine ca fișier

/live – trimite comenzi live (start/stop)

PostgreSQL:

Folosit pentru stocarea metadatelor imaginilor și a dispozitivelor înregistrate

Aceste canale sunt organizate clar pentru a asigura separarea responsabilităților între backend, frontend și dispozitive mobile.

Team Contributions:

Team member	Lines added	Lines removed	NO. Commits
Badea Dragos	388	102	5
Badulescu Catalin	185 892	881	9
Nastase Daniel	1024	119	25
Radu Octavian	1612	240	5
Serban Bianca	2039	123	12

OSSF Criticality Score:

```
PS C:\Users\drago\AndroidStudioProjects\MQTT-Web-App> python3  
.criticality_score.py  
Using default OSSF configuration (original Pike algorithm)
```

=== Local Criticality Score Analysis ===

```
Repository: C:\Users\drago\AndroidStudioProjects\MQTT-Web-App  
Remote URL: https://github.com/biancaserban20/MQTT-Web-App  
Configuration: Default OSSF (Pike algorithm)
```

=== Raw Metrics ===

```
repo.name: MQTT-Web-App  
repo.created_date: 2025-05-26  
repo.last_updated: 2025-05-26  
legacy.created_since: 0 months  
legacy.updated_since: 0 months  
legacy.contributor_count: 7  
legacy.org_count: 4  
legacy.commit_frequency: 1.25 commits/week  
legacy.recent_releases_count: 0  
legacy.closed_issues_count: 0  
legacy.updated_issues_count: 0  
legacy.comment_frequency: 4.49  
legacy.dependents_count: 1
```

=== Calculating Criticality Score ===

```
created_since: value=0, max_threshold=120, weight=1, normalized=0.0  
updated_since: value=0, max_threshold=120, weight=-1, normalized=1.0  
contributor_count: value=7, max_threshold=5000, weight=2, normalized=0.0028  
org_count: value=4, max_threshold=10, weight=1, normalized=0.4
```


commit_frequency: value=1.25, max_threshold=1000, weight=1, normalized=0.00125
recent_releases_count: value=0, max_threshold=26, weight=0.5, normalized=0.0
closed_issues_count: value=0, max_threshold=5000, weight=0.5, normalized=0.0
updated_issues_count: value=0, max_threshold=5000, weight=0.5, normalized=0.0
comment_frequency: value=4.49, max_threshold=15, weight=1, normalized=0.299333
dependents_count: value=1, max_threshold=500000, weight=2, normalized=4e-06

=== Final Results ===

Total weighted score: 1.703387

Total weight: 10.5

default_score: 0.16223

 MINIMAL CRITICALITY: This project has minimal criticality

=== Comparison with OSSF Parameters ===

Note: This is a local approximation. Actual OSSF scores consider:

- GitHub-specific metrics (stars, forks, watchers)
- Issue and PR activity from GitHub API
- Dependency graph data
- Cross-repository references

Your local analysis provides a baseline understanding.