Security of Systems – Project Assignment

Authors

Mihnea-Andrei Blotiu

Stefan-Darius Iordache

Stefan-Dorin Jumarea

Roxana Popa

May 26, 2025

Contents

1 System Architecture Overview

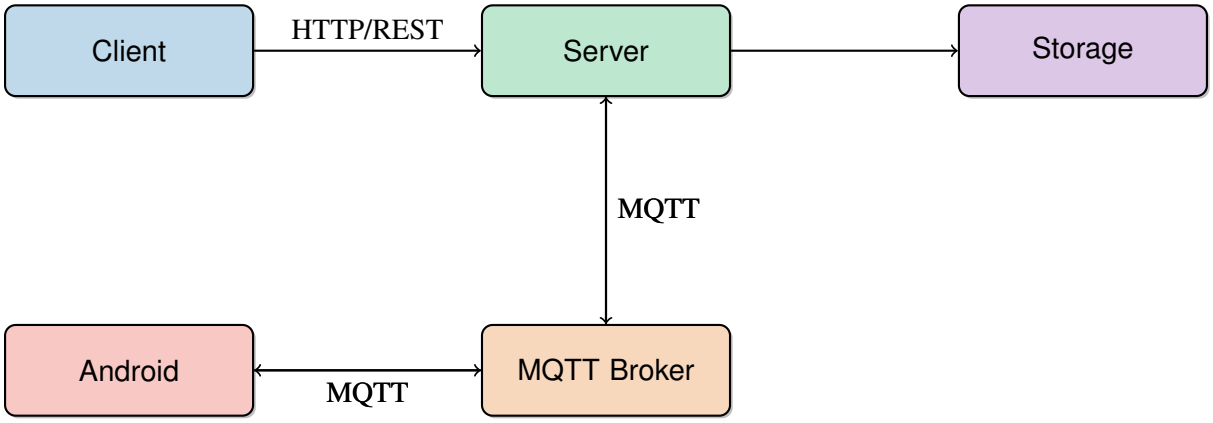


Figure 1: System Architecture of Security of Systems - First Force

The project represents a sophisticated distributed security platform that seamlessly integrates multiple cutting-edge technologies to deliver comprehensive surveillance capabilities. At its core, the architecture follows a modern microservices approach as it can be seen in Figure 1 with real-time communication channels between components. The architecture implements a hybrid communication model where REST APIs handle administrative operations and configuration, while MQTT provides the real-time messaging.

The system architecture consists of five primary components working in concert:

- **Client Application**: A responsive React-based web interface that provides administrators and security personnel with an intuitive dashboard for monitoring, search operations, and system configuration.
- **Server Backend**: Powered by Go for maximum performance and concurrency, the server handles authentication, device management, image processing, and serves as the central coordination point for the entire system.
- **Android Application**: A native Kotlin mobile client that extends monitoring capabilities to on-the-go scenarios, allowing remote system control from the UI.
- **MQTT Broker**: Facilitates lightweight, real-time bidirectional communication between system components using the publish-subscribe pattern, enabling status updates with minimal latency.
- **Storage Layer**: Combines MongoDB's flexible document storage for metadata and event information with S3-compatible object storage for efficient management of captured images.

1.1 Core Components & Technology Stack

Component	Technologies	Responsibilities
Client	React, TypeScript, TailwindCSS	User interface, photos search, device settings management
Server	Go, AWS SDK, MongoDB	Backend, coordination, authentication
Android App	Kotlin	Mobile interface, real-time and on-command photos
MQTT Broker	Mosquitto	Real-time communication between server and devices

1.2 Key Features & Workflows

The system operates through several interconnected workflows:

- **Real-time Monitoring:** Live view of device feeds with mode switching.
- **Text-based Search:** Search photos by extracted text content.
- **Multi-platform Access:** Web interface and Android application.
- **Image Capture:** Photos triggered manually or periodically (live / normal mode).
- **Advanced Search:** Filter photos by date, device, text content.
- **Role-based Access:** Different permission levels in UI (user / admin).
- **Secure Communication:** Encrypted data transfer via TLS.

1.3 Deployment Architecture

The system is deployed using Docker containers orchestrated with Docker Swarm as it can be seen in Figure 2, allowing for easy scaling of individual components based on load requirements. The deployment can be customized for cloud environments or on-premises installation.

1.4 Development Workflow

The project follows a modern development workflow with continuous integration and deployment:

- Feature branch development with pull request reviews
- Automated testing through GitHub Actions workflows
- Dependency management with Renovate (configured limit: 5 PRs)
- Automated Docker image builds on successful merges to main
- Automatic versioned deployment with rollback capabilities

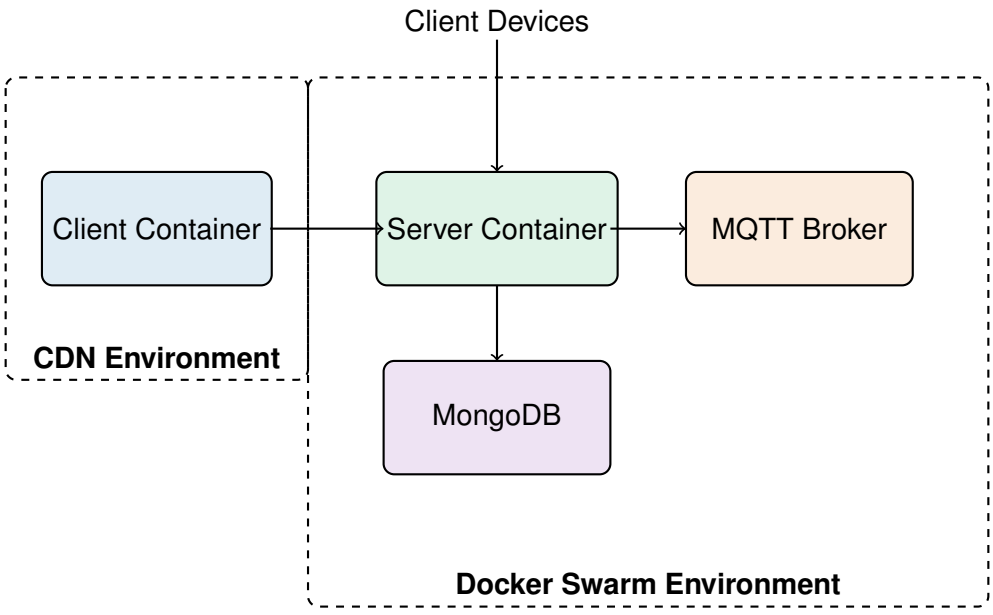


Figure 2: Container-based Deployment Architecture

2 Functionality, Documentation, Execution

This section outlines the core functionality of the system, provides documentation highlights, and explains both user and development documentation.

2.1 Working features (e.g., screenshots, sample usage)

The system is functionally divided into three primary components, each serving a distinct purpose while working in concert to deliver a complete solution. This section provides an overview of the implemented features across the web client, server backend, and mobile application, demonstrating how they fulfill the system's objectives.

2.1.1 Android Application

Key Features

- **Camera Integration:** Uses the Android CameraX API to preview and capture images.
- **MQTT Communication:** Integrates with the Eclipse Paho MQTT client to publish images to a topic.

- **Secure Communication:** Supports SSL/TLS connections using CA, certificate, and private key files stored in the `res/raw` directory.
- **User Interaction:**
  - Button to capture and send image manually.
  - Buttons to change the application mode (live/manual) and to start or stop transmission.
- **Preferences:** Stores a unique device identifier (UUID) or integer values persistently using SharedPreferences.

MQTT Messages

- **Register Message:** At the start of the application, a message is sent to the broker with the device ID, in order to let the server know that the application is connected.
- **Photos:** All photos are sent via mqtt, to the photos/ID topic.
- **Manual/Live Mode:** The server can send messages to the application via the device/setup/ID topic, which will change the mode the application is running (Live or Manual)
- **Disconnect Message:** When the application is closed, a disconnect message is sent to the server.

Application Architecture

- **MainActivity:** Handles camera setup, MQTT connection, user interactions, and UI updates.
- **Image Processing:** Converts `ImageProxy` to `Bitmap`, compresses to JPEG, and encodes in Base64.
- **MQTT Logic:** Publishes on a separate thread to avoid blocking the UI, and subscribes to topics with message callbacks.
- **Threading:** Uses Handler and Runnables for periodic UI updates.

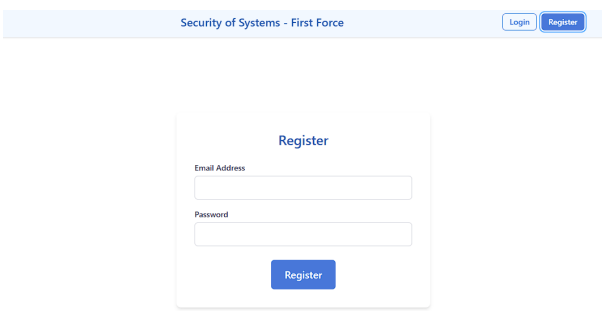
Figure 3: Main screen of the application showing camera preview and control buttons.

2.1.2 Frontend Client

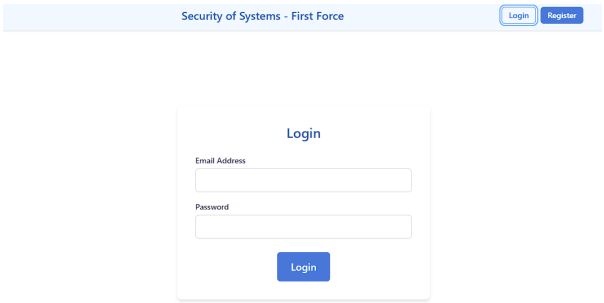
As it can be seen in Figure 4, the user meets with a beautifully-designed landing page.

Figure 4: Landing page of the UI

Then, the app has both register and login support as stated in Figures 5a and 5b below:



(a) Register page

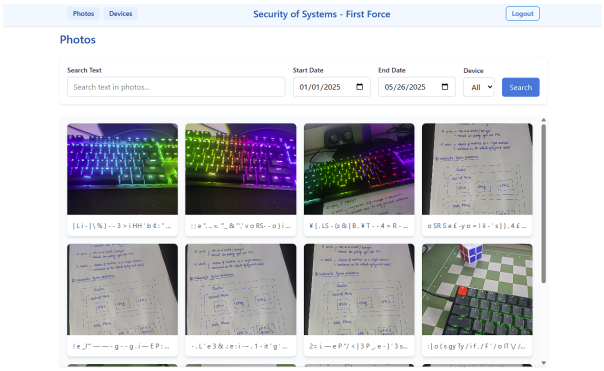


(b) Login page

Figure 5: Register and login pages

If a user registers and then logs into the application, the UI will offer two possibilities:

- A photos page with an example in Figure 6a. It offers possibilities for seeing photos taken by all the devices, searches by the text of the photo as well as filtering by date of when the photo was taken.
- A devices page with an example in Figure 6b. A normal user will not have access to view, monitor and change settings of connected devices. An admin example view of the same page will be provided below.



(a) Photos page

(b) Devices page

Figure 6: Photos and devices pages

In case the registered user is an admin, there will be no difference on the photos page. However, on the devices page, a list of all registered devices at one moment in time can be seen as it is highlighted in Figure 7. There the admin user can change between the normal and live mode of each and every device.

Figure 7: The devices page for an Admin User

2.1.3 Server

The server is responsible for all communication across other components, state management and processing. It is written in Go, making it suitable for concurrent tasks.

- **Connectivity:**
  - HTTP Server: the server listens on port 8080 for HTTP requests from the client website.
  - MongoDB: a NoSQL database, used for keeping information about users, devices and photo metadata.
  - MQTT-Client: connected to our broker for fast communication between the server and connected devices. It receives photos from devices, as well as register/disconenct events from them, and publishes mode configuration changes to them.
  - S3: object storage from AWS, used for photos.
- **Features:**
  - Authentication: users can register accounts and login to be able to use the platform
  - Role Based Access Control: there are 2 distinct roles: **admin** and **user**. The admin has a few extra permissions apart from a normal user. It can visualize connected devices and switch them from normal to live modes.
  - OCR Processing: for every image received by the server from the MQTT broker, OCR processing is performed using the **tesseract** tool to extract text from that image before uploading it to the S3 object storage and saving its metadata in the MongoDB
  - Device Management and Registration: every device must send a registration signal to be able to send photos. The server will keep device state so the client website will be able to send configuration settings to active devices.

2.2 CI/CD evidence or execution logs

Currently, several CI/CD actions run on the repository:

2.2.1 CodeQL

Static analysis is being performed on every pull request and commit to the main branch. Found vulnerabilities are reported and explained in the following section.

2.2.2 Syft

Syft is used to generate an SBOM regarding our server and it is triggered on every change of the server files.

2.2.3 Scorecard

OSSF runs on every pull request to analyze the repository. A score, reported problems and potential fixes are generated by the action regarding the whole repository and not just for the server.

2.2.4 Testing and Deployment

Unit tests run on changes on the server files. If the coverage of the tests is above a threshold, then the new server is deployed. The current tests cover over 60% of the codebase.

3 Security & Compliance

3.1 Threat Modeling & Mitigations

For threats we are using a static code analysis tool (CodeQL) that generates and reports issues as it can be seen in Figure 8. Some of these issues have been solved when reported, some still to be solved. New issues will be reported at each new PR where CodeQL runs for each line of added code.

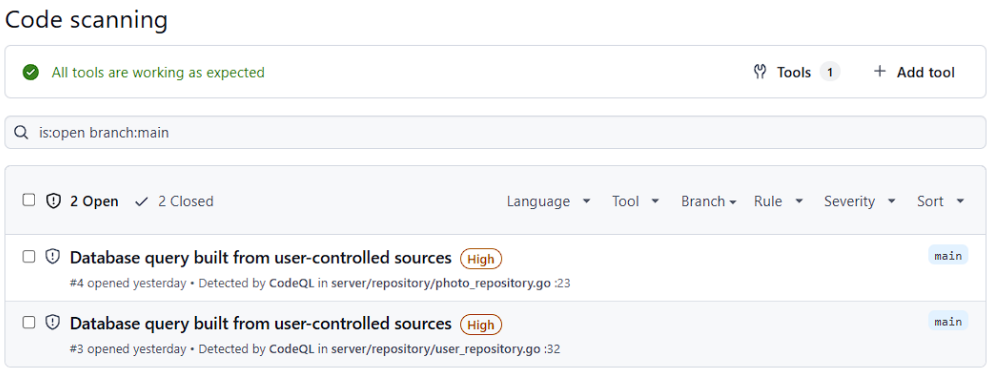


Figure 8: CodeQL reported issues

3.2 Testing & Coverage

For CI/CD coverage testing, we used *go tool cover*. The action runs at every change to the server files, and if the coverage is less than 30%, it fails.

```
48  mqtt-streaming-server/routes/device.go:20:      InitDeviceRoutes        0.0%
49  mqtt-streaming-server/routes/device.go:30:      SwitchDeviceMode        68.8%
50  mqtt-streaming-server/routes/device.go:62:      GetDevices              100.0%
51  mqtt-streaming-server/routes/init.go:14:        InitRoutes              0.0%
52  mqtt-streaming-server/routes/init.go:26:        withCORS                0.0%
53  mqtt-streaming-server/routes/init.go:43:        withAuth                0.0%
54  mqtt-streaming-server/routes/photo.go:21:       InitPhotoRoutes         0.0%
55  mqtt-streaming-server/routes/photo.go:29:       GetPhotos               76.3%
56  mqtt-streaming-server/routes/user.go:22:        InitUserRoutes          0.0%
57  mqtt-streaming-server/routes/user.go:33:        Register                83.3%
58  mqtt-streaming-server/routes/user.go:75:        Login                   90.9%
59  mqtt-streaming-server/routes/user.go:117:       GetProfile              100.0%
60  total:                                          (statements)            60.1%
61  Total coverage: 60.1%
```

Figure 9: Code Coverage.

3.3 SBOM & Dependencies

All the dependencies are pinned using hashes (commit hashes for GitHub actions and image digests for Docker images).

OpenSSF detects no vulnerabilities in the code, as seen in Figure 11.

A GitHub action runs on every change in the server files and generates the SBOM. The action uses Syft for generating the SBOM.

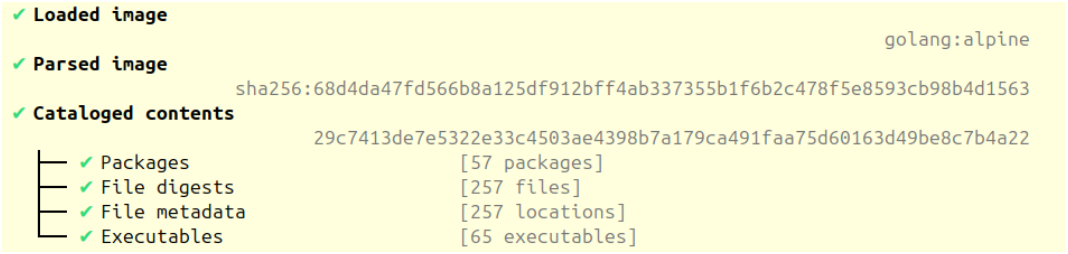


Figure 10: SBOM report for the server.

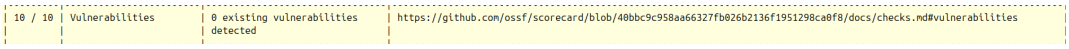


Figure 11: OpenSSF vulnerability report.

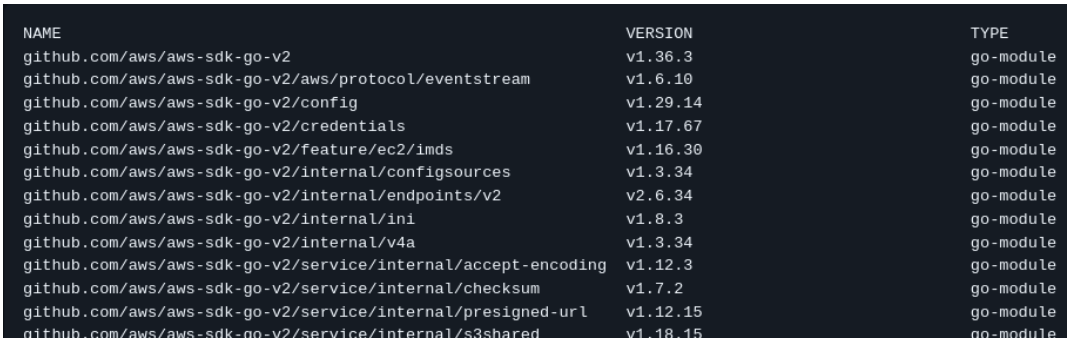


Figure 12: OpenSSF vulnerability report.

4 Team Contributions

In the below table the team members and their contributions to Github can be seen. The contributions were and are not limited to Github code.

Team Member	Lines Added	Lines Removed	Nr. of commits
Mihnea-Andrei Blotiu	4235	6303	53
Stefan-Darius Iordache	5710	342	31
Stefan-Dorin Jumarea	1145	171	10
Roxana Popa	223	0	3

Table 1: Git statistics per team member

5 OSSF Criticality Score

We used OSSF to assess the security of the web server. The OSSF score is 10.0/10.0 as stated and highlighted in Figure 13. This score is obtained using the following tests:

- Binary-Artifacts: No binaries found in the repository.
- Dangerous-Workflow: No dangerous workflow patterns detected.
- Dependency-Update-Tool: Update tool detected (renovate).
- Fuzzing: Project is fuzzed (using gofuzz).
- License: The project uses MIT License.
- Pinned-Dependencies: All dependencies are pinned by hash (docker image digest has, git commit hash, publickey hash).
- SAST: CodeQL is used.
- Security-Policy: SECURITY.md file is present.
- Token-Permissions: All actions have readonly token permissions.
- Vulnerabilities: No vulnerabilities found.

```
Aggregate score: 10.0 / 10

Check scores:
Finished [Fuzzing]
Finished [Vulnerabilities]
Finished [Pinned-Dependencies]
Finished [License]
Finished [Binary-Artifacts]
Finished [SAST]
Finished [Token-Permissions]
Finished [Dangerous-Workflow]
Finished [Dependency-Update-Tool]
Finished [Security-Policy]
Finished [Packaging]

RESULTS
-------
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
|  SCORE  |         NAME         |             REASON              |                                        DOCUMENTATION/REMEDIATION                                    |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Binary-Artifacts     | no binaries found in the repo   | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#binary-artifacts     |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Dangerous-Workflow   | no dangerous workflow patterns  | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#dangerous-workflow   |
|         |                      | detected                        |                                                                                                     |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Dependency-Update-Tool | update tool detected          | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#dependency-update-tool |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Fuzzing              | project is fuzzed               | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#fuzzing              |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | License              | license file detected           | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#license              |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Packaging            | packaging workflow detected     | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#packaging            |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Pinned-Dependencies  | all dependencies are pinned     | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#pinned-dependencies  |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | SAST                 | SAST tool detected: CodeQL      | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#sast                |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Security-Policy      | security policy file detected   | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#security-policy      |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Token-Permissions    | GitHub workflow tokens follow   | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#token-permissions    |
|         |                      | principle of least privilege    |                                                                                                     |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
| 10 / 10 | Vulnerabilities      | 0 existing vulnerabilities      | https://github.com/ossf/scorecard/blob/40bbc9c958aa66327fb026b2136f1951298ca0f8/docs/checks.md#vulnerabilities      |
|         |                      | detected                        |                                                                                                     |
|---------|----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------|
```

Figure 13: OSSF Report