אפקה
המכללה האקדמית להנדסה בתל אביב

המחלקה להנדסת תוכנה

GuardianView

Student Name: Omer Iny Dori Rozen
Supervisor's Name: Yoram Rozenwaks

Project Book

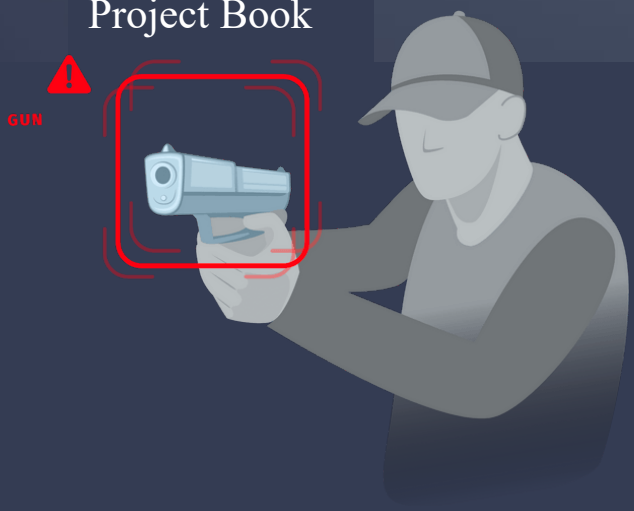


Table of Contents

Extensive summary:

Guardian View is designed to address the critical need for enhanced security in public spaces and to reduce response time to potential security incidents, especially in malls, airports, and secure areas where high foot traffic and activity complexity increase vulnerability to threats. It is also intended for main streets monitored by security cameras, particularly in Israel where knife and gun threats are common and security awareness is high.

The project introduces an AI-powered surveillance system aimed at detecting potential security threats such as weapons in real-time using advanced object detection technology. At the core of the system is the integration of the YOLOv8 model, known for its accuracy and speed in detecting objects in video streams. This technology allows security personnel to receive immediate alerts and visual verification of threats, enabling a rapid and informed response.

The system's architecture leverages cloud technologies for scalability and real-time data processing, using Firebase for reliable data management, storage, and user authentication. This cloud-based infrastructure ensures that the system is flexible and can handle large amounts of data seamlessly, which is crucial for live video processing. Additionally, we have integrated Docker for future needs of running the system on the cloud for scalability purposes.

Way of Compliance:

Ever since the engineering report was submitted ,the engineering method adopted for Guardian View focuses on integrating cutting-edge AI technology with modern cloud infrastructure to create a reliable and efficient security monitoring solution. The project was initiated with a thorough analysis of the needs and constraints associated with security in closed environments as well as the constraints we have as students with low resources and access to real time cameras streaming. Following this analysis, the decision was made to utilize the YOLOv8 model due to its effectiveness in real-time object detection and its ability to be integrated into a Python-based backend system.

The engineering approach involved several key steps:

- **Model Training and Integration**: Customizing and training the YOLOv8 model to specifically detect weapons and integrating it with the backend infrastructure.
- **Cloud Integration:** Setting up Firebase as the backbone for real-time alerts, video storage, and user management, ensuring that data flows seamlessly between the frontend and backend.
- **Frontend Development:** Designing a user-friendly interface with Flutter that allows security personnel to monitor live feeds, receive alerts and review historical data effectively.
- **Testing and Optimization**: Rigorous testing was conducted to ensure the system's reliability under different scenarios. This included stress tests, security audits, and user acceptance testing to refine the interface and functionality.

Further details on the engineering methods will be elaborated in the subsequent chapters, specifically focusing on the technical development, challenges overcome, and the iterative processes applied to refine the system's capabilities. This comprehensive approach not only meets the immediate security needs but also provides a scalable solution that can adapt to future enhancements and integrations.

Compliance

Significant attention was dedicated to the changes that have occurred since the submission of the engineering report document. These changes, whether in the project's goals, methodology, or technical architecture, are not deviations but rather informed decisions made to optimize project outcomes considering new information, technical constraints, and strategic reassessments. By detailing these adjustments in a tabular format, the report provides transparency, allowing stakeholders to understand the rationale behind each decision, who initiated these changes, and their implications on the project's trajectory.

Table of changes in Guardian View

Change	Manifestation of the Change	Time	Who initiated the change	Significance	Approved by
Addition of Settings Navigation and Live Streaming features	We added a navigation to destination of alert in the front end and a settings tab that contains activation of live detection and changing of confidence threshold	Mar 24	Students	Enabled new functionalities for better system management and user experience.	Yoram
Database Redo	Database was restructured due to the lack of cameras and different usage needs.	Mar 24	Supervisor	Adapted to new requirements and improved data management.	Yoram
Removal of Flask	Flask was removed because all communication is now managed through Firebase.	Apr 24	Students	Simplified architecture by consolidating communication through Firebase.	Yoram
Removal of On Duty Feature	The "On duty" feature was removed.	May 24	Students	Reduced complexity and focused on core functionalities.	Yoram
Demo Instead of Real Security Integration	Instead of real-time security integration, a demo of the system with pre-	Mar 24	Students (Dori)	Adjusted due to infrastructure and computing limitations, enabling practical	Yoram

	recorded videos was created and a live feature from a webcam was created as well.			demonstration of capabilities.	
Upload videos and Guardian View app were united	Architecture of the system changed after this change	Mar 24	Students (Dori)	For better flow and usage of the system apps were merged	Yoram
Added location to user	For better threat response. While currently the locations are autogenerated because this system is a demo, we don't have live connected cameras and actual users. this is a setup for the future. Thus, we can simulate an event that a security personal can navigate from his location to the threat location.	Mar 24	Students	When a user is close to a possible threat it will alert him as well	Yoram
Removal of TDD Testing	The initial plan included TDD for the project, but it was removed due to time constraints and prioritizing feature development, In addition, because most of the code is boiler code, there is no need to check the same thing multiple times.	Apr 24	Students	Allowed more focus on delivering core functionalities within the tight schedule	Yoram
Change in System Architecture	Shifted from micro services architecture to Event-Driven Architecture due to our system needs and ways of communication	Mar 24	Students	Enabled real-time data processing and quick response to events, improving system efficiency and alert management	Yoram

Our achievements while developing Guardian View

Phase	Achievements	Impact
Core Architecture Setup	Set up Firestore DB to manage core data collections: Users, Videos, Settings, and Alerts.	Enabled real-time data synchronization and storage, ensuring up-to-date and accessible data,
Customized YOLO model training and development	Found a database of treat pictures trained our model on a paid google collab GPU and developed a model that satisfied our requirements	We got to a model that has low loss and high accuracy according to our available resources and the best available DB online
Backend System Development	Integrated YOLO model for real time video data processing and used Firebase Storage for video data.	Enabled efficient video analysis and threat detection, forming the core functionality.
Iterative Development	Continuously integrated and tested new features.	Minimized errors and allowed quick resolution of issues, maintaining high code quality.
Event-Driven Architecture	Implemented an architecture for real-time data processing and alert generation. While both frontend and backend updates and listens to firebase.	Ensured quick response to new data, providing timely alerts and maintaining system efficiency.
User Interface with Flutter	Developed a cross-platform Flutter application for managing user settings and receiving notifications with a video slice of the event captured with an option to navigate to the event.	Provided a responsive and intuitive interface for both normal and admin users.
User Authentication	Used Firebase Auth for user authentication and permissions management.	Ensured secure access with appropriate permissions, enhancing overall system security.
Handling Challenges	Integrated third-party services, managed real-time data synchronization, and implemented error handling.	Resulted in a resilient system capable of handling unexpected issues gracefully.
Continuous Testing	Regularly tested and debugged the system.	Ensured all components functioned correctly and efficiently, delivering a high-

		quality product.
Live Video Analysis	Implemented live video analysis using YOLO model and OpenCV.	Enabled real-time monitoring and alert generation from live video streams.
Settings Listener	Developed a listener for Firestore settings to dynamically update system configurations.	Allowed real-time updates to confidence thresholds and live detection status.

Gantt chart and Responsibilities

Our project schedule, as shown in the accompanying Gantt chart, outlines a comprehensive timeline for the "Guardian View" project, capturing the key milestones and phases from the initial research and design through to the final stages of testing and documentation. Each phase is marked with specific start and end dates, providing a detailed roadmap for project progression and expected deliverables. We have successfully completed the tasks in the Gantt in their timeframes.

Engineering Knowledge that was learned

Between the period of the first part of the project after the initiation phase, Dori was learning flutter in order to create our front end as professionally as possible, while Omer was learning the YOLO Model and ultralytics library how it works and how to connect to it and refine it in order for it to be our backbone for this project, training the model and integrating it with a backend in python. We both needed to learn how to use and work with the firebase framework in there we created our DB and our cloud storage and, in the future, we can run the backend on the cloud using this framework.
In addition, the both of us took courses to learn other architectures microservices, cloud computing and design of systems in Afeka to help us with the design of the DB and the architecture in general for this elaborate project. due, in the end we decided to use an event driven architecture because it better suited our needs.

Problems that interrupt the project's progress so far and how we dealt with them:

1. Lack of real security cameras that made significant changes in the design of the entire project and its purposes. We adapted by creating a demo system using pre-recorded video footage, enabling us to simulate real-time threat detection. we added a live feature as well from the webcam of the local computer.

2. Lack of computing power to train an ai model that suits our exact needs and a good dataset in order to train it. in order to deal with that problem first we took the best DB of photos of threats we could find which contained about 18,000 annotated photos of guns knifes and people then we paid google Collab and ultralytics and trained 5 models with the computing power of google and the ultralytics framework, at the end we chose the best model and integrated it with the python backend.

3. Delays in the planning phase that caused a chain reaction of delay in the entire project.
   We implemented a more agile project management approach, using tools like Gantt charts and Monday.com to better track progress, manage deadlines, and allocate resources efficiently.

4. Learning to develop the front side of out project using Clean Architecture within Flutter which is the best organized way to develop an app and understanding state management using BLoC and providers, was challenging.

Methodology

Chosen Tools and Engineering methods and Technologies:

Firebase Firestore: For real-time database management, enabling swift data synchronization and seamless state management across client applications.

Firebase Auth: To provide a robust authentication mechanism, ensuring secure user access and data protection.

Firebase Storage: Utilized to store and serve user-uploaded content such as videos and images. It provides secure file uploads and downloads. Firebase Storage ensures scalability and reliability, handling large amounts of data with ease.

Flutter: Chosen for front-end development to create a cross-platform, user-friendly interface with a single codebase.

Python: Utilized for the backend to create lightweight and scalable, facilitating smooth integration with our AI models and Firebase services.

Ultralytics YOLO (You Only Look Once): Integrated for its real-time object detection capabilities, forming the core of our threat detection mechanism the model was trained by us on the framework of ultralytics based on theirs cutting edge yolo algorithm.

Git: For version control, allowing the team to collaborate effectively and manage code changes.

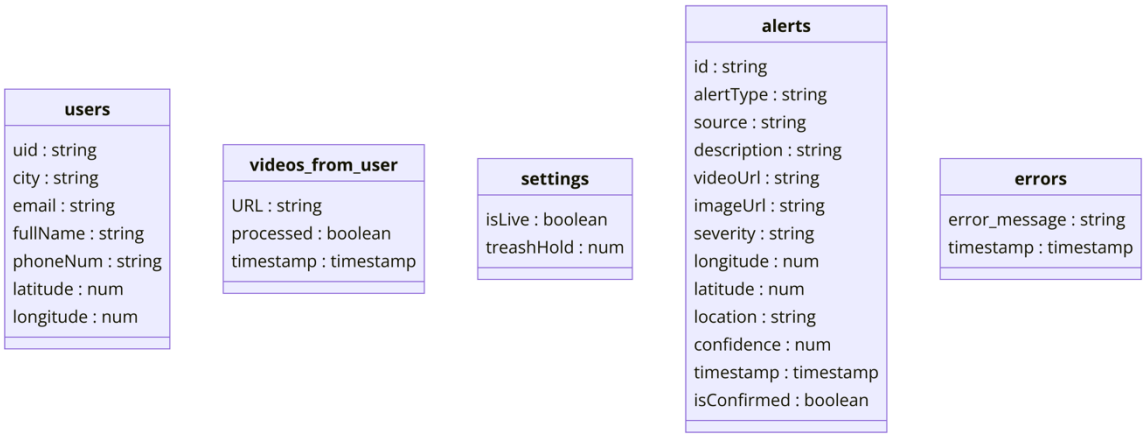
PlantUML and draw.io: For diagramming and visual documentation, crucial for conceptualizing system architecture and workflows.

Gantt Charts Trough Monday.com: For project management, enabling the team to track progress, deadlines, and resource allocation.

Docker: Utilized to containerize the application, ensuring consistent environments across different stages of development and deployment. Docker enables easy scalability, deployment, and management of microservices, facilitating smooth integration and continuous delivery. Right now, docker is set up but the system doesn't run on docker because the system uses local computer resources to run like computer local memory, display, webcam and more.

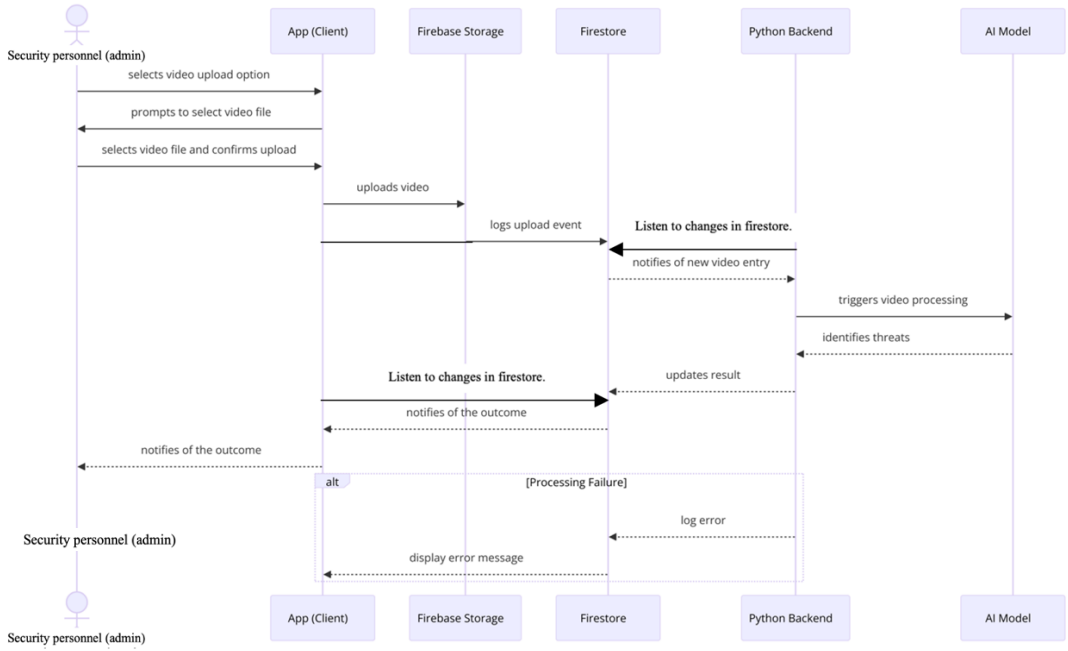
Each tool and method were carefully evaluated and selected based on their ability to contribute significantly to the system's design and development. And was chosen because it was the best possible alternative after checking all our options in the engineering report

The ERD Bellow illustrates the structure of our Firestore database with five main collections: users, videos_from_user, settings, alerts, and errors. The users collection stores user information, while videos_from_user holds metadata for user-uploaded videos. The settings collection contains system settings for live video analysis and confidence thresholds. The alerts collection records detailed alert information, and the errors collection logs system errors with corresponding timestamps. Each collection has its own primary key which is an String ID in some of the tables managed by us and in others managed by firebase and thus not showing in the ERD



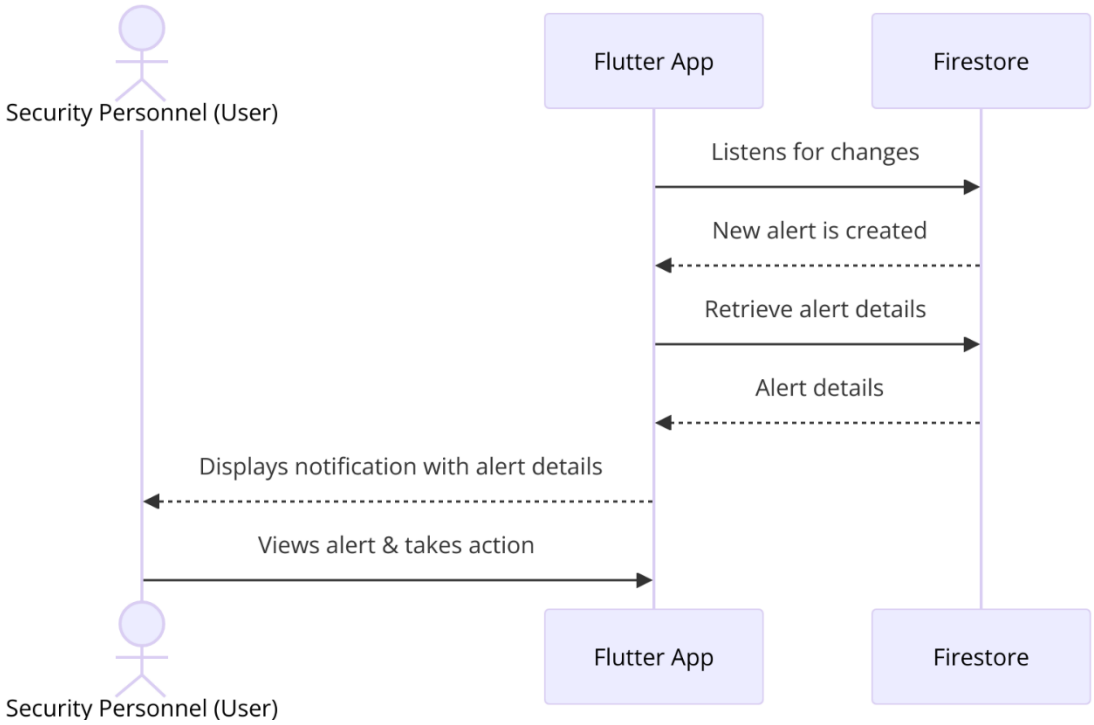
Upload and Process Video for Threat

Security personnel upload videos for threat detection the system processes the uploads and generates alerts for identified threats according to the model and updates the firestore database. The client app listens to the database, notifying the user if a threat was detected and updates the alerts list.



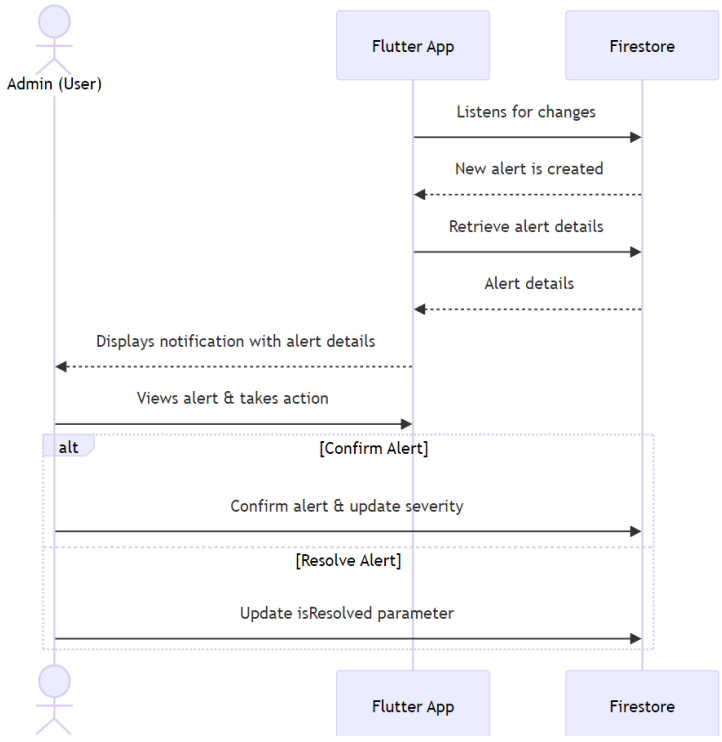
Receive Alert Notifications

Users receive real-time notifications for alerts generated by the system, ensuring they are promptly informed of new or updated threat situations.



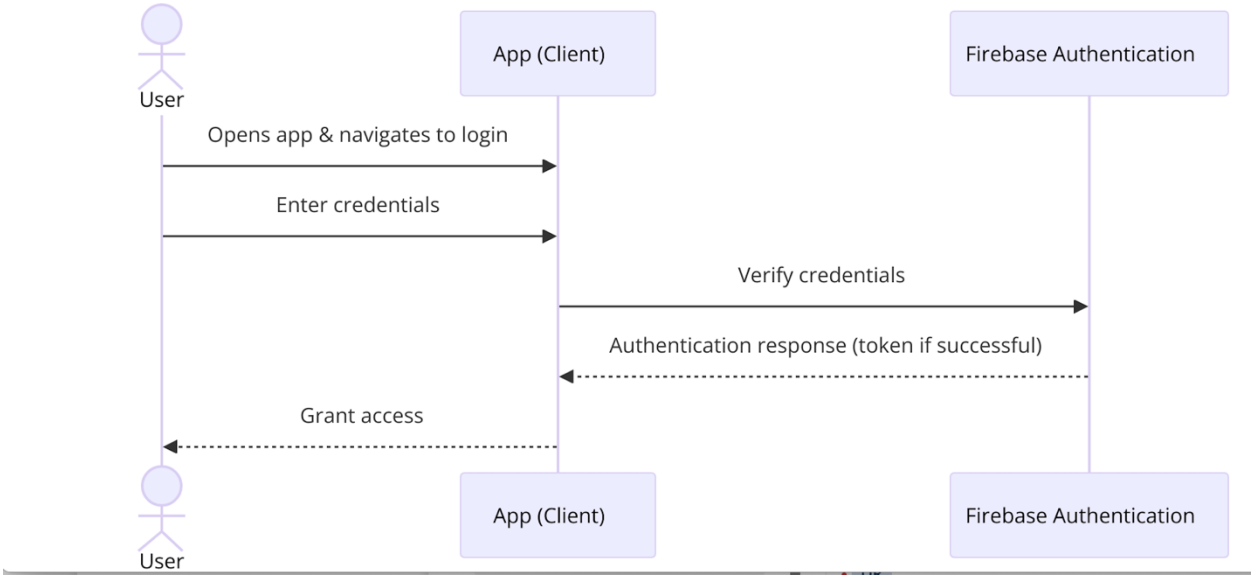
Admins receive Alert Notifications

Admins receive real-time notifications for alerts generated by the system with all severity levels, having the power to confirm or resolve the event appeared, ensuring they are promptly informed of new or updated threat situations.



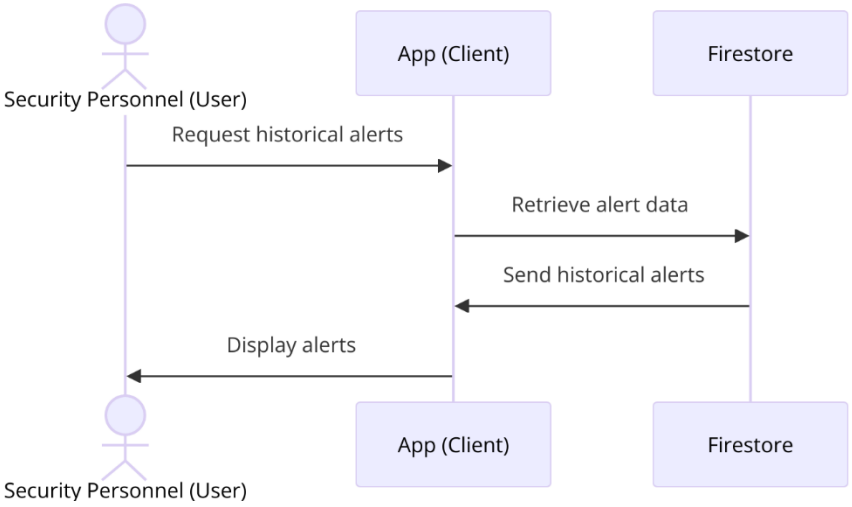
Authenticate User

Users authenticate with the system using their credentials to gain access to its features, with the system validating the login and granting appropriate access.



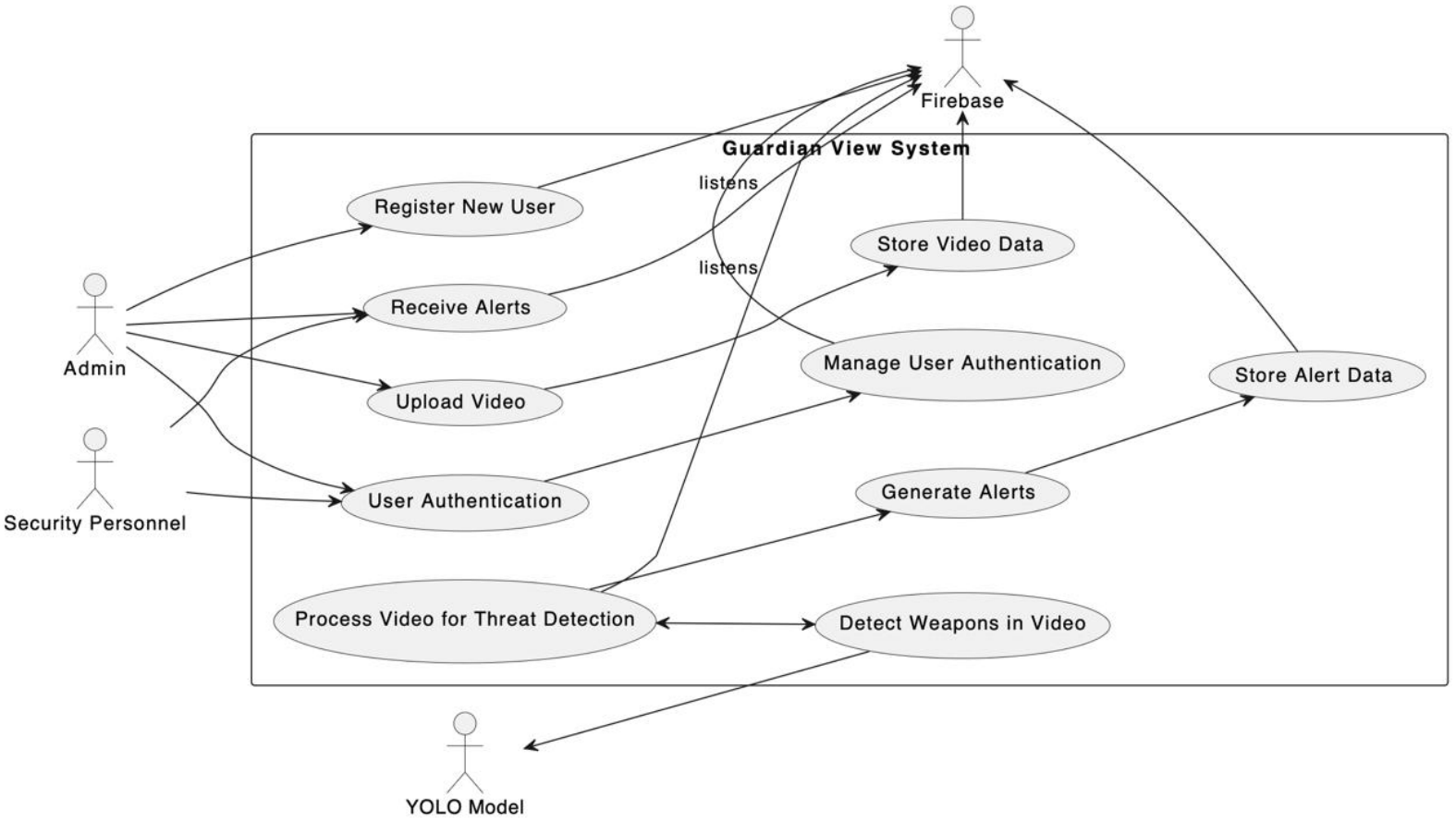
Review Historical Alerts

Security personnel review historical alerts to stay informed of past incidents, with the system providing a comprehensive archive of alert data.



Use case diagram:

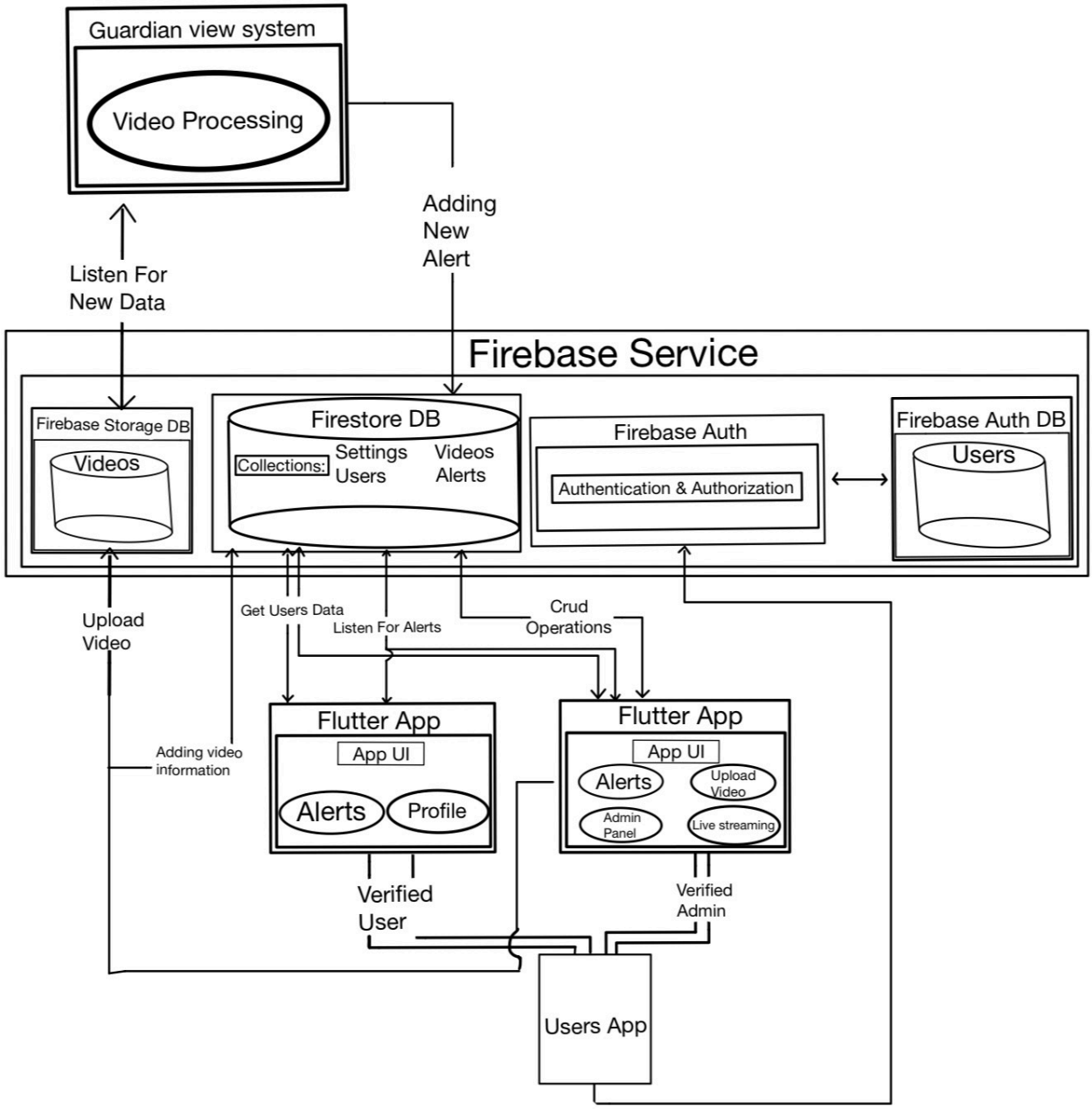
In the following diagram we illustrate the interactions between users and the system in terms of how users interact with the system to achieve specific goals.



Results

Architecture Diagram

The diagram below encapsulates the system's architecture, illustrating the interplay between our application's components and the data flow. They are essential in providing a visual representation of the system's operational dynamics and the relationships between its various services.



Components of the diagram:

The system architecture utilizes Firestore DB for data storage and synchronization.

It consists of four main Firestore DB collections: Users videos settings and Alerts.

The system includes a Guardian side system with Process Data (YOLO) for monitoring and generating alerts from the Video collection and from live streaming inside the backend system.

Flutter App handles the user interface and interactions, with components for Alerts, and demonstration videos of camera for generating alerts.

The admin side will manage the users who can access to the system and get alerts.

and will be a second check for each alert received, for making sure it's a real alert by confirming it and then it will display also for the users.

Firebase Storage DB is used for storing video data.

Description of the chosen architecture:

The architecture chosen for the project is Event-Driven, as it deals with real-time data processing and alert generation. This architecture allows for efficient handling of incoming data and quick responses to events. Initially we wanted to implement a microservices architecture but for efficiency purposes we chose to shift to event driven architecture, we assumed that because our system is event driven low latency and imitated action is high priority to our system.

The communication, the processing and consistency of the events in our system is the core structure of the solution to the problem we face. Also notifications that arrive in real time. This architecture allows coupling Minimal, which makes it a good option for us.

Architecture components

Flutter Application: Responsible for displaying alerts with the additional information on it, location data video interfaces. Also having user authentication, and data management. It also Utilizes Firestore DB for user alert settings and video data, and Firebase Storage for video data.

Backend system: Handles alert processing using YOLO model and communicate with firebase for receiving and updating the alerts data if needed.

Functional/logical components

User Management:

- User registration and login
- User profile management
- User permissions (authenticated and unauthenticated user)

Manage alerts:

- Create new alerts (for a demo purpose).
- Reading and updating existing alerts.
- Deleting notifications (can be deleted directly from firebase only).
- Listen to changes in alerts in real time.

Data and video processing:

- Data processing using the backend guardian view system using listeners to the firebase DB checking if a new video was uploaded or the live function should be activated.
  then immediately it starts processing the video and creates new alerts if needed.
- Adding new alerts based on data and video processing

Video management:

- Upload videos to Firebase Storage.
- Videos data is saved as well in firestore DB.
- Videos are stored in Firebase Storage.

Data synchronization:

Firebase handles data synchronization on its own, providing real-time responses to listeners when data changes

User interface:

- Show notifications
- User authentication using Firebase Auth
- Permission management for regular users and administrators

Operative software modules:

User authentication module:

- Managing login and registration processes, user authentication against Firebase Auth DB.
- Distinguish between normal users and authenticated administrators.
- Every user that passed the auth phase successfully will be passed to the admin/user screen according to his user definition which is set in flutter.

Notification management module:

- Create, read, update and notifications in Firestore DB.
- Processing and displaying notifications in the user interface.
- Synchronization of notifications in real time between the server and the application.

Video processing module:

- Running the YOLO model for data processing.
- Analyzing data with added logic so we can reduce false positive alerts and creating new alerts accordingly.

Video management module:

- Upload videos from the app to Firebase Storage
- Upload videos from live streaming to Firebase Storage
- Manage and organize stored videos information in firestore

Data synchronization module:

- Synchronization of user data and notifications between Firestore DB and the application
- Manage listening for changes in real time

User Interface Module:

- Display notifications, location and video in the app interface
- Manage admin interactions with displayed data like confirmation of an alert.

User actions:

Normal user:

Receiving notifications on the device that include:

- Video in some cases, a picture with the identification of the weapon, location, general details and access to a navigation pass to the scene of the incident.
- In case the event is close to the user, he will receive this information even if he knows that he is among the first to respond to the event.

Changing basic user settings such as: phone number, password and more.

Admin user:

- Adding new users and edit existing ones.
- Confirmation of an alert that is categorized as a low/medium and even high reliability level depending on the degree of certainty.
- Initial setting of the system to the percentage of the safety level with which he would like to operate the system and define the threshold of the model. Also added an activation and deactivation of the live video analysis.

Algorithms of video processing service in the backend:

Video Analysis Algorithm: The video analysis algorithm processes user uploaded videos by first downloading them to the local machine. Each frame of the video is then analyzed using our YOLO model, which is trained to detect threats such as guns and knives. The algorithm scans each frame, logging detections with the highest confidence scores. If a threat is detected consistently over a specified number of consecutive frames, an alert is generated. The algorithm also applies bounding boxes to detected threats to indicate their location within the frame. Upon completion of the analysis, the status of the video in the Firestore video collection is updated to "processed," ensuring the system doesn’t process the same video twice.

Live Video Analysis Algorithm: The live video analysis algorithm captures video frames in real-time from a live camera feed. Each frame is analyzed using the YOLO model to detect threats with confidence levels above a set threshold. If a threat is consistently detected across multiple frames, an alert is generated. The algorithm ensures continuous monitoring and dynamically adjusts alert states based on real-time detections.

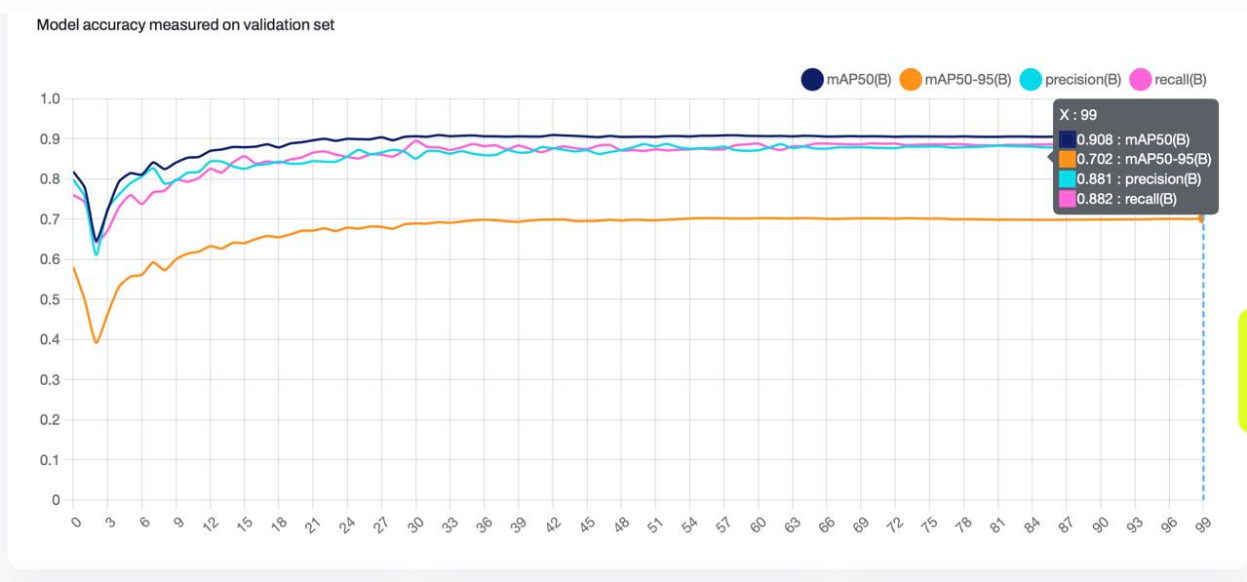
Training the YOLOv8 Model using Ultralytics Library

Training Process:

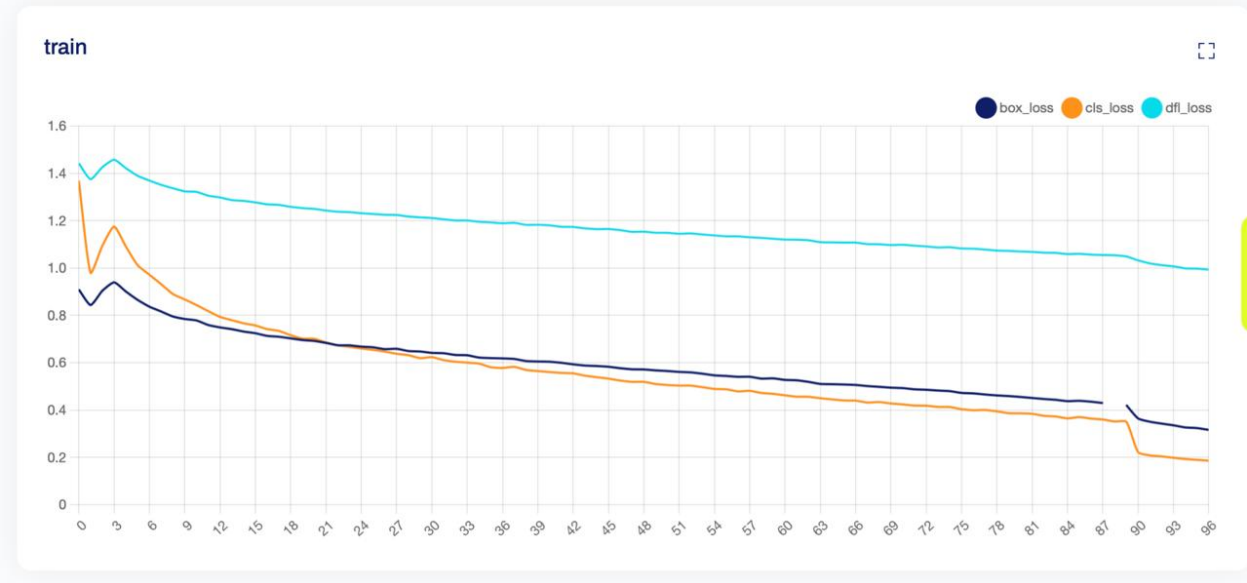
1. **Dataset Acquisition**: The dataset was sourced from Roboflow, which offers datasets in formats compatible with YOLO models (e.g., COCO or VOC annotations). The dataset was downloaded and preprocessed to match the YOLOv8 requirements.

2. **Environment Setup**: The environment was set up using the Ultralytics YOLO library. This involves installing necessary libraries, including PyTorch and Ultralytics YOLO.

3. **Model Training**: The training script utilized the Ultralytics library to load the dataset, define the model architecture (YOLOv8m), and start the training process. Key parameters such as batch size, learning rate, and number of epochs were configured. And the training was done using google collab as they have powerfull GPUs we had to use.



Model Accuracy on Validation Set: The chart shows the performance of the model across training epochs, with metrics like mAP50, mAP50-95, precision, and recall. The high values indicate good overall detection performance and effective learning.



Box Loss, Class Loss, and Object Loss: These charts depict the reduction in different types of losses during training. Box Loss measures the accuracy of bounding box predictions, Class Loss indicates the correctness of class predictions, and Object Loss reflects the model's confidence in detecting objects. The decreasing trends show that the model is learning effectively.

Alpha Version

Use Cases:

Actors:

The user (the hotline or the person in the field).

Prerequisite:

The system must be initialized to a certain threshold for us to receive notifications with a certain level of security.

Be connected to the application and available for the notification that will appear on the screen.

Use case – editing the alert settings.

The admin defines the security level he would like to receive notifications about. Meaning he can adjust the confidence level in which the model will detect threats.

Use case - receiving alerts in the system and confirming them

All alerts with a security level that is low / medium / high / emergency will appear only to the admin who will have to confirm the event. In the event of an urgent alert, it will reach the person in the field directly, without the need of confirmation, but of curse still will be seen for the admin.

Use case - user management.

The admin can add new users, and edit information of existing users, when both an admin and a regular user can edit their own information.

Use case - user receives a notification.

The user is connected to the application and receives a notification with a certain security level, It contains information about the event that includes preliminary information on whether it is very close to the event or not. The user observes the details of the event and can click on a button called "navigate to the location" of the event.

Walkthrough of the system:

Auth:

The definition of our user object includes the following attributes:

uid, fullName ,email, phoneNum, city, Location (consisting of longitude and latitude)

in the flutter code the admins are defined and when a user logs in it presents the screen according to the users permissions.

User Location Significance

The user's location provides additional context about their proximity to events received in notifications.

Admin Account Management

- Admin accounts are manually added to the database and saved locally into the flutter app.
- Admin credentials are securely provided through a third-party method

Login Process

The login process follows these steps:

1. User enters their email and password.

2. The entered credentials are sent to Firebase Authentication for verification.

3. Firebase Authentication checks if the credentials are valid and registered.

4. If authentication is successful, the system receives authentication approval and user credentials.

5. The system then checks if the authenticated email matches the designated admin email.

6. Based on the email check:

- If it's a match: The user is navigated to the admin panel.

- If it's not a match: The user is directed to the regular user screen.

7. If authentication fails at step 3, the user is shown an error message and prompted to try again.

User Management by Admin

Administrators can add new users to the system by:

- Adding the new user's information to the database through the system.

Firebase Authentication

Firebase Auth handles User Credential information automatically, including:

- Additional user info
- Token ID
- Credentials
- Other relevant data

User Registration Process

1. An existing admin initiates the new user registration process.

2. The admin enters the new user's information, including email and temporary password.

3. The system uses Firebase Authentication to create a new user account.

4. Firebase returns a UserCredential object containing the unique user ID (UID).

5. The system then creates a new document in the Firestore database's 'users' collection:

- The document ID is set to the user's UID for easy reference.

- This document stores additional user information that can't be saved in Firebase Authentication.

- Examples of additional data might include user preferences, role-specific information, or application-specific settings.

6. The admin provides the new user with their login credentials (email and temporary password).

7. Upon first login, the new user should be prompted to change their password for security reasons.

Note: Firebase Authentication stores basic user information (like email and password hash), while the Firestore database is used to store extended user data that Firebase Authentication doesn't support natively.

Getting The Alerts :

The definition of our alerts object includes the following attributes:

alertType, source, description, videoUrl, ImageUrl, id, severity, location, latitude, longitude, confidence, timestamp, isConfirmed.

Location Information

The latitude and longitude parameters indicate the incident's location. Users can utilize this data to navigate to the site using Google Maps integration within the app.

Media Content

The imageUrl and videoUrl are crucial components. The UI fetches this content from Firestore and Firebase Storage URLs.

Confirmation Flag

The isConfirmed flag helps the system differentiate between alerts that should be sent to admins versus regular users. When an alert has been confirmed as an actual threat, then an alert pops to all users, if the severity level is emergency the alerts pops up to all users without the need to confirm the alert.

Resolved Flag

The isResolved flag helps the system to close the event and making the alert invisible to the user in the field, but saves the alert for the administrator to view an historical events.

Real-time Updates

The Flutter app continuously listens to the database, requesting data updates. Any changes in the DB are reflected in real-time UI updates.

Alert Distribution

Admin View

- Admins receive alerts of all severity levels.

User View

Regular users receive alerts that are either:

- Confirmed (isConfirmed = true)
- Marked as emergency severity

Admin Alert Management

- Admins review all captured alerts.
- For important alerts, the admin presses a confirm button ,this action updates the database.
- Confirmed alerts are then pushed to regular users.

This system ensures that admins have full visibility of all alerts, while users receive only the most relevant and verified information. The real-time nature of the app ensures quick dissemination of critical alerts to all necessary parties.

Uploading Video:

Our Video object consists of the following attributes: Uid, processed, timestamp.

processed: Boolean flag indicating processing status

Processing Flag

The "processed" boolean flag serves as an indicator for the backend:

- When false: Signals that the video is new and requires processing

- When true: Indicates the video has been processed

Upload Process

The video upload occurs simultaneously to two Firebase services:

- Firebase Storage: Stores the actual video file
- Firestore: Stores the video metadata (uid, URL ,processed status, and timestamp). The processed flag in uploaded as false by default and is changed to true after the video has been analyzed.

Purpose of the App

This application serves as a testing tool for our project:

- It simulates real-time camera feeds that we don't have physical access to
- Allows us to create and test fake events
- Helps in validating the entire system's workflow without real-world incidents

Workflow

1.User uploads a video through the app

2.Video file is stored in Firebase Storage

3.Video metadata is stored in Firestore

4.Backend system listens for new entries in the video collection

5.When a new, unprocessed video is detected, the backend initiates processing

6.After processing, the "processed" flag is updated to true

7.Relevant alerts are generated based on the video content

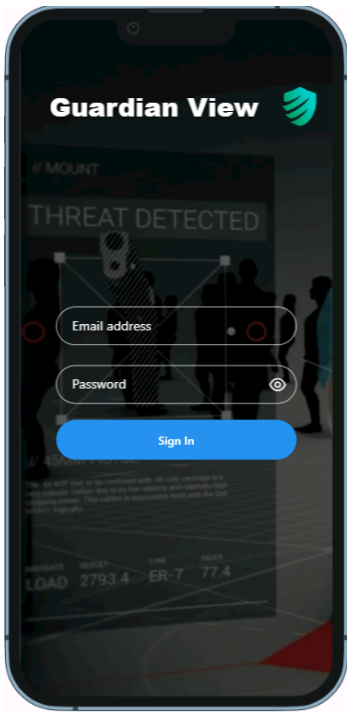
Processing the Video in Backend using YOLO Model:

Classifier Algorithm Overview:

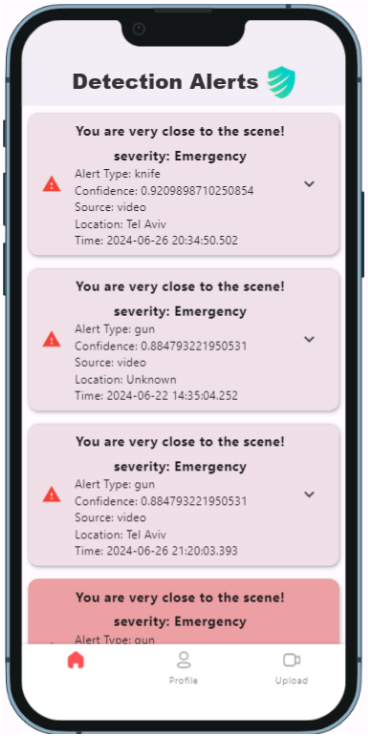
1. **Integration with Firebase Storage:**
   - Firebase Storage serves as a secure and scalable cloud storage solution, housing the uploaded videos for processing by the analysis algorithm.
   - Upon video upload, the system downloads the video to the local machine for processing.
2. **Application of YOLO Model:**
   - The YOLO model is applied to each frame of the video after preprocessing.
   - It analyzes each frame, identifying potential weapons and people, including guns and knives, by predicting bounding boxes and associated class probabilities.
3. **Weapon Detection Process:**
   - The YOLO model detects regions within each frame that contain objects resembling weapons.
   - Detection is based on learned patterns and features from the training data, allowing accurate distinction between weapons and other objects.
4. **Bounding Box Generation:**
   - Upon detecting a potential weapon, the YOLO model generates a bounding box around the object to indicate its location and size.
   - These bounding boxes are visual indicators of weapon presence, facilitating subsequent analysis and action.
5. **Logic to Filter False Positives:**
   - **Consistency Check:** The system requires multiple consecutive frames with a detected weapon before generating an alert. This reduces the likelihood of false positives from transient objects.
   - **Confidence Threshold:** Only detections with confidence levels above a predefined threshold are considered. This threshold is dynamically adjustable based on real-time feedback from the system.
   - **Bounding Box Validation:** Detected bounding boxes are validated based on size and aspect ratio to ensure they match typical weapon characteristics.
   - **Contextual Analysis:** The system analyzes the context of the detection within the frame, considering the surrounding objects and environment to further validate the presence of a weapon.
6. **Results:**
   - As the YOLO model processes each frame, it aggregates the detected weapons and bounding boxes into a comprehensive summary.
   - This summary consolidates all detected weapons across the video, providing a holistic view of potential security threats.

7. **Communication with Firebase Firestore and alert generation:**
    - After completing the weapon detection process, the system updates Firebase Firestore and generates alert if needed with the video data and detection results.
    - The alerts database is updated with the detected weapons and their significance level, ensuring prompt and informed responses to potential threats.

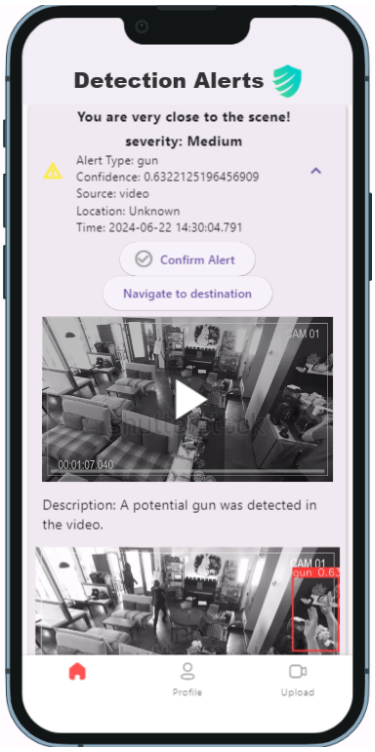
Alpha GUI:



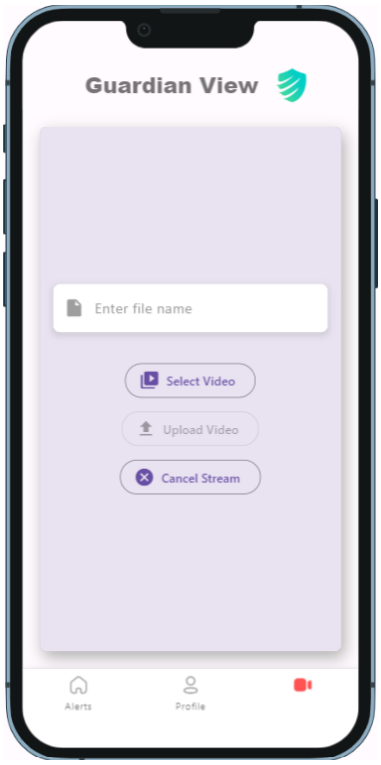
Login Page



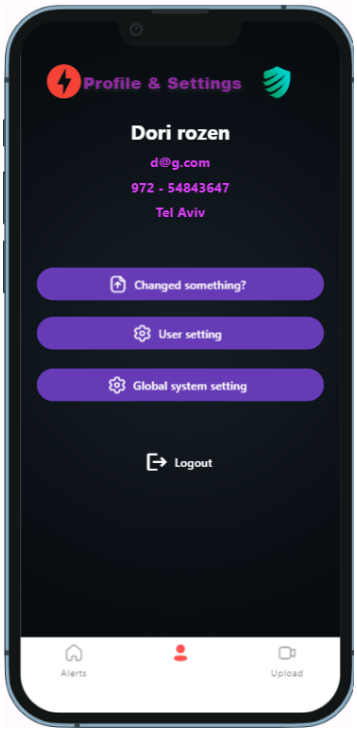
Alert page



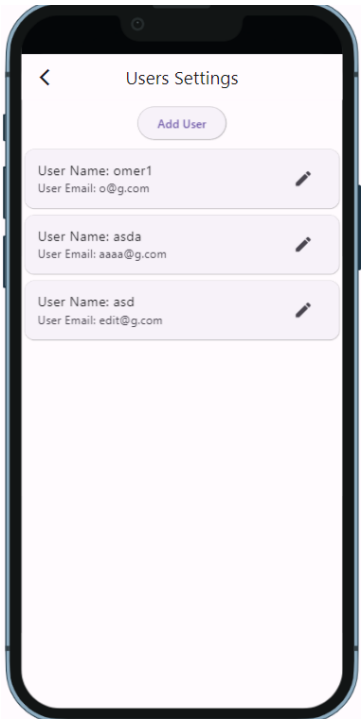
Alert Widget Expanded



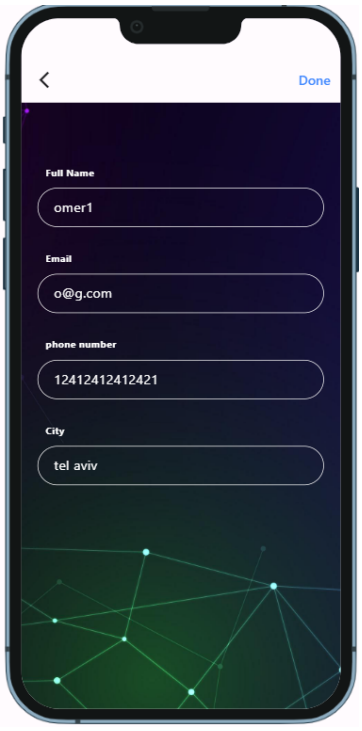
Upload Video Page



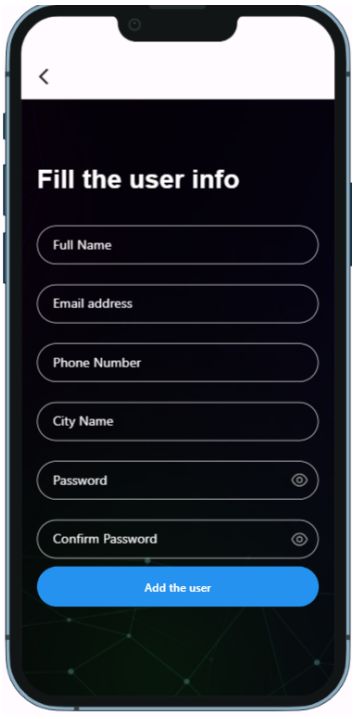
Profile Page



Profile Sub Page

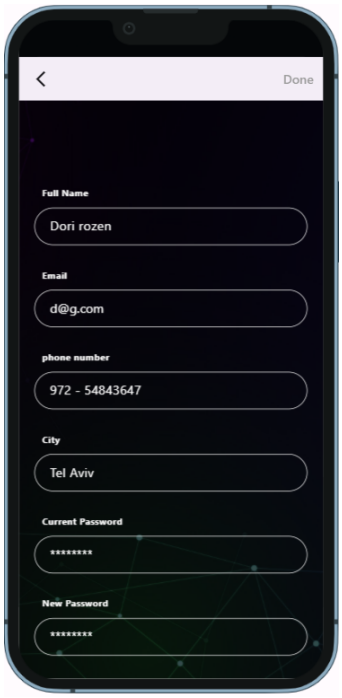


Profile Sub Page

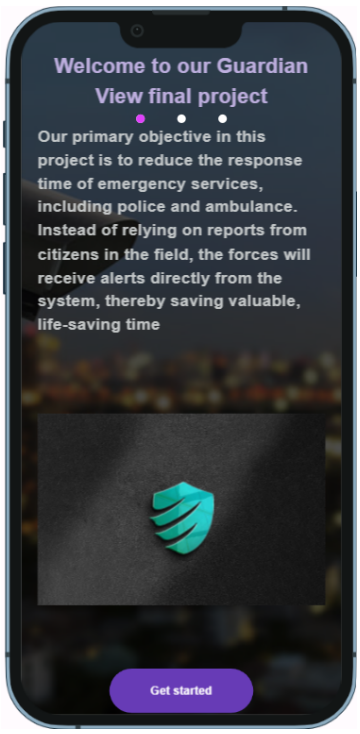


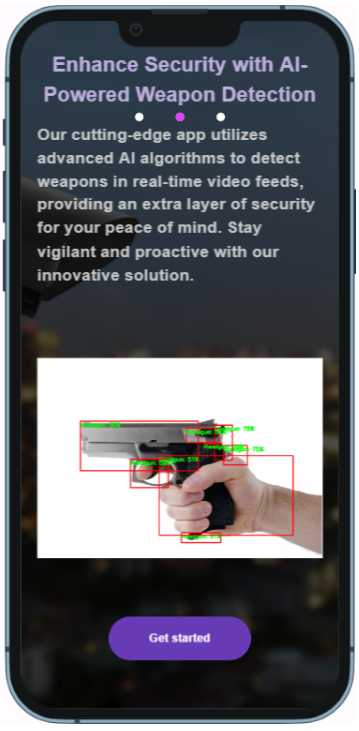
Profile Sub Page

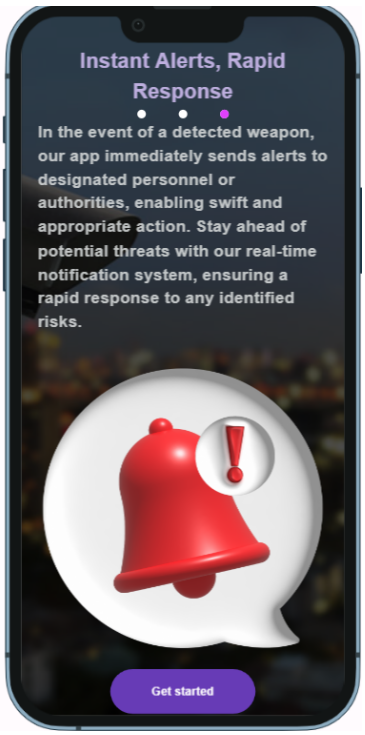
Profile Sub Page



Profile Sub Page







Welcome Pages

Testing

Our project's testing strategy ensured comprehensive validation across all system modules and functionalities. We conducted targeted tests on critical components including user authentication, backend connectivity, threat detection, data integrity, and real-time alerts. Each test was crafted to verify that system operations met our requirements for efficiency and reliability. This approach not only affirms the system's operational integrity but also reinforces our commitment to delivering a robust solution. Our testing also covered UI components and user experience, focusing on eliminating bugs, exceptions, and unexpected behaviors to ensure a smooth, intuitive application performance for end-users. The outcomes of these tests guide our ongoing development efforts and were conducted using logs and manual testing.

Tested Module	Specs of Test	Desired Outcome	Actual Outcome
Auth User	Test if an admin can successfully add a user	Only admins can register a new user, ensuring security and control.	Passed
User Authentication Flow	Test login and authentication flows for various user roles	Users can log in based on their credentials; unauthorized access is prevented.	Passed
Python Connectivity to Firebase	Validate Python server's ability to connect and communicate with Firebase	Seamless data exchange between Python backend and Firebase services.	Python connects to firebase and prints a log that its successfully connected
YOLO Model Up	Check if the YOLO model is operational and can process	YOLO model processes video data accurately for threat detection.	Model successfully preforms analysis on its input

	videos by the Guardian View system		
Firestore - Data Integrity	Ensure complete and accurate retrieval of data.	All requested data is retrieved correctly.	Data retrieval is successfully retrieved. if some fields are invalid error pops up
Firestore - Alerts Generation	Simulate conditions for alert generation and verify database updates	Alerts are generated and logged correctly based on detected threats.	Checked that when given a video with a threat it detects that threat and pushes an alert with no empty fields
Video Upload and Storage	Test uploading and retrieval of video data to/from Firebase Storage	Videos are uploaded, stored, and retrievable without data loss.	Passed
Real-time Alerts List Retrieval	Validate the real-time retrieval of alerts from Firestore	Security personnel receive real-time updates on alerts. And all the users get the data that they are supposed to get	Passed
Processing Video in Video Processing Service	Verify video processing done correctly with the YOLO model manually.	Videos are processed efficiently, and potential threats are accurately detected.	while the threats are visibly very clear in the video and take a big chank of the image on the video it detects the

			weapon. Otherwise, the model is struggling to detect and sometimes doesn't detect threats or miss possible threats. This is because of a lack of computing power. The model is not perfect, but it works.
Flutter	TDD was initially used for complex and sensitive parts of the code. As development progressed, testing shifted to manual methods. Later testing relied on developers' expertise with the complex core architecture.	The whole inner code will be correct as the programmer wanted in the first place.	Passed and works smoothly.

Risks

Risk	Probability	Severity	Damage Potential	Realized	Mitigation Actions
Team Availability	**5**	**5**	**25**	No	Implemented agile practices and flexible work policies
Coping With The Impact Of The War	**9**	**5**	**45**	No	Luckly we both didn't have too much schedule changes due to the war
Difficult implementation of Complex Features	**5**	**9**	**45**	Yes	Broke down features into smaller tasks, used simpler alternatives where necessary
Lack Of Data	**2**	**7**	**14**	Yes	After 2 weeks of research, we found a balanced dataset that has high quality and fits our needs.
Lack Of Computing Power	**5**	**8**	**40**	Yes	Opted for more efficient models, leveraged cloud computing resources
Firebase Cancels Free Access	**1**	**10**	**10**	No	Wasn't realized
Poor Communication	**4**	**8**	**32**	No	Held weekly meetings, used project management tools for clear communication
Difficulties in integrating third-party services (e.g., Firebase, YOLO model)	**4**	**5**	**20**	Yes	Utilized support and documentation, assessed fallback options
Incompatibility of chosen technologies	**1**	**10**	**10**	No	Had alternative technologies in place

Underestimation of development time for features	**5**	**8**	**40**	Yes	Included buffer time, adopted agile methodology
Changes in project requirements or scope creep	**3**	**8**	**24**	Yes	Continuously prioritized features, focused on core functionalities

Discussion

Transitioning from the Engineering report to more advanced stages, the "Guardian View" project has seen significant achievements. Initially designed to integrate with live security camera feeds, Guardian View had to adapt due to the unavailability of live camera feeds, leading to a pivotal redesign. This shift necessitated the use of prerecorded videos for analysis for us to create a working demo of the system, influencing the architecture and the approach to data handling. Despite these challenges, the project successfully integrated Firebase for real-time data management and YOLO for video analysis, creating a responsive and dynamic user interface using Flutter. The development process thought us about the importance of flexibility, effective problem-solving, and continuous learning. Key lessons include the need to adapt to changing requirements, the value of robust technical skills in both front-end and back-end development, and the critical role of effective team collaboration and communication. As we progressed, we transitioned from a traditional architecture to an event-driven architecture, enhancing our ability to handle real-time data and respond to security threats with less latency and more efficiently.

Summary and conclusions

The "Guardian View" project, dedicated to enhancing security through advanced AI and system integration. focused on foundational development and strategic planning. Key achievements include the development of a dynamic front-end interface using Flutter, integration with Firebase for real-time data management, and the implementation of the YOLO model for video analysis. The project adapted to unforeseen challenges, notably the reevaluation of its approach due to technological constraints, leading to a transition to an event-driven architecture. These experiences have enriched the team's expertise, underscoring the importance of flexibility, continuous learning, and innovation in tackling complex engineering problems. The journey thus far sets a solid groundwork for the next phases of development, with a clear vision for achieving a state-of-the-art security monitoring solution. The project's success in overcoming integration challenges, adapting to changing requirements, and leveraging new technologies positions it well for future enhancements and broader impact on security and safety in various environments.

References

Flutter :

Main site docs:
https://docs.flutter.dev/
Ultimate List Of Flutter Resources:
https://wilsonwilson.dev/articles/flutter-resources

Firebase:

Main site docs:
https://firebase.google.com/docs

Flask:

Ultimately wasn't used
https://flask.palletsprojects.com/en/3.0.x/

Yolo:

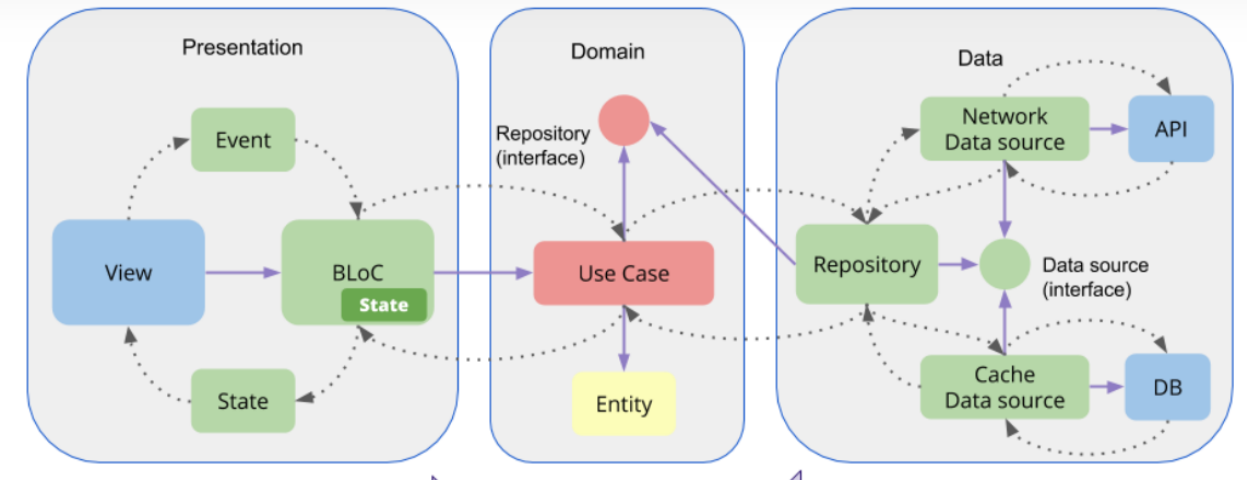
Library and how to work with it using the docs
https://github.com/ultralytics/ultralytics

Roboflow:

A site for datasets for training of the model
https://universe.roboflow.com

Appendices

Flutter Clean Architecture :



Presentation Layer:

- **View:** Represents the user interface (UI) components responsible for displaying information to the user and capturing user input**.**
- **Event:** Represents events or actions triggered by user interactions within the UI**.**
- **State:** Represents the current state or data that needs to be displayed in the UI.
- **Bloc (Business Logic Component):** Handles the business logic and state management of the application, separating it from the UI layer.

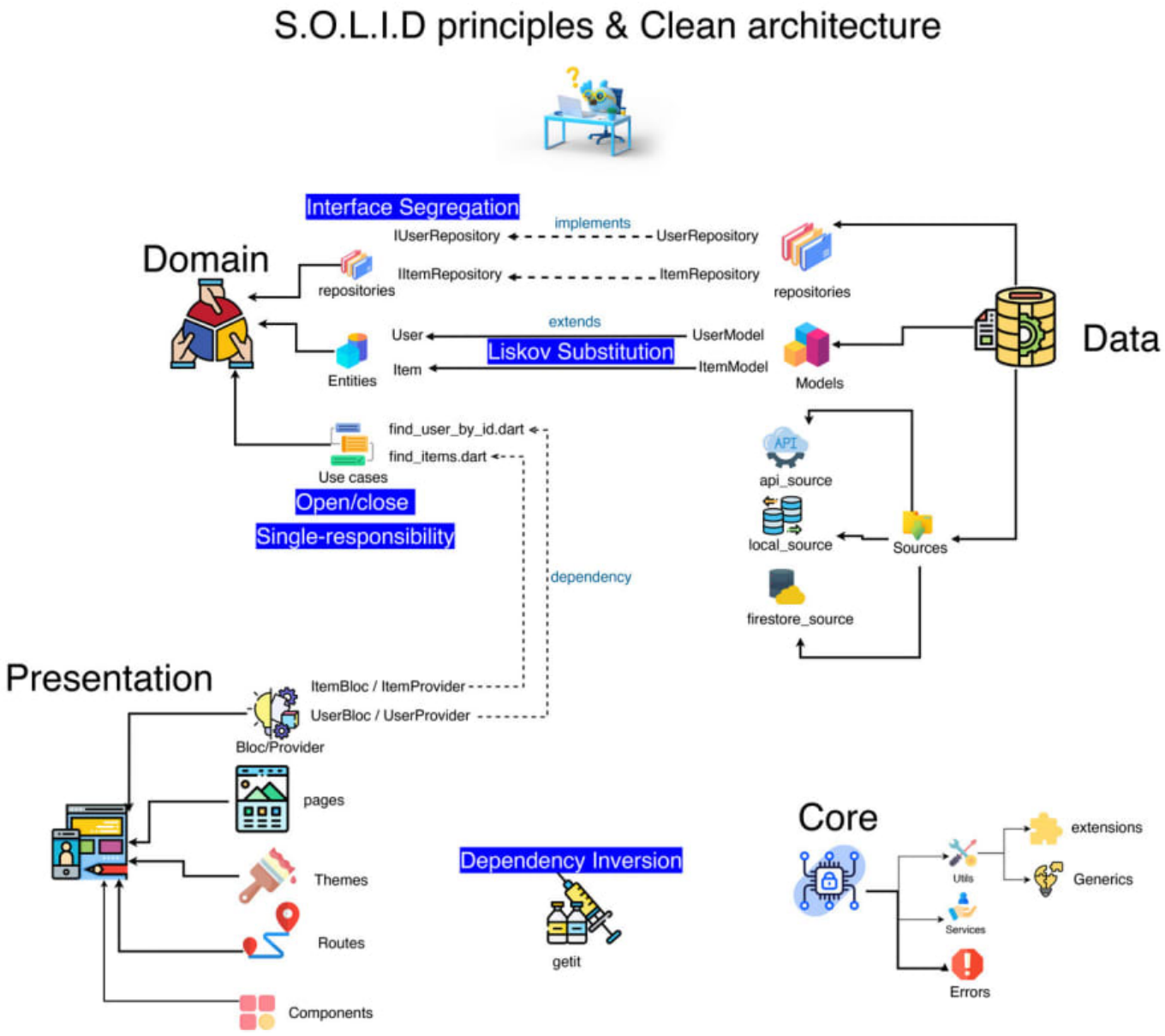
Domain Layer:

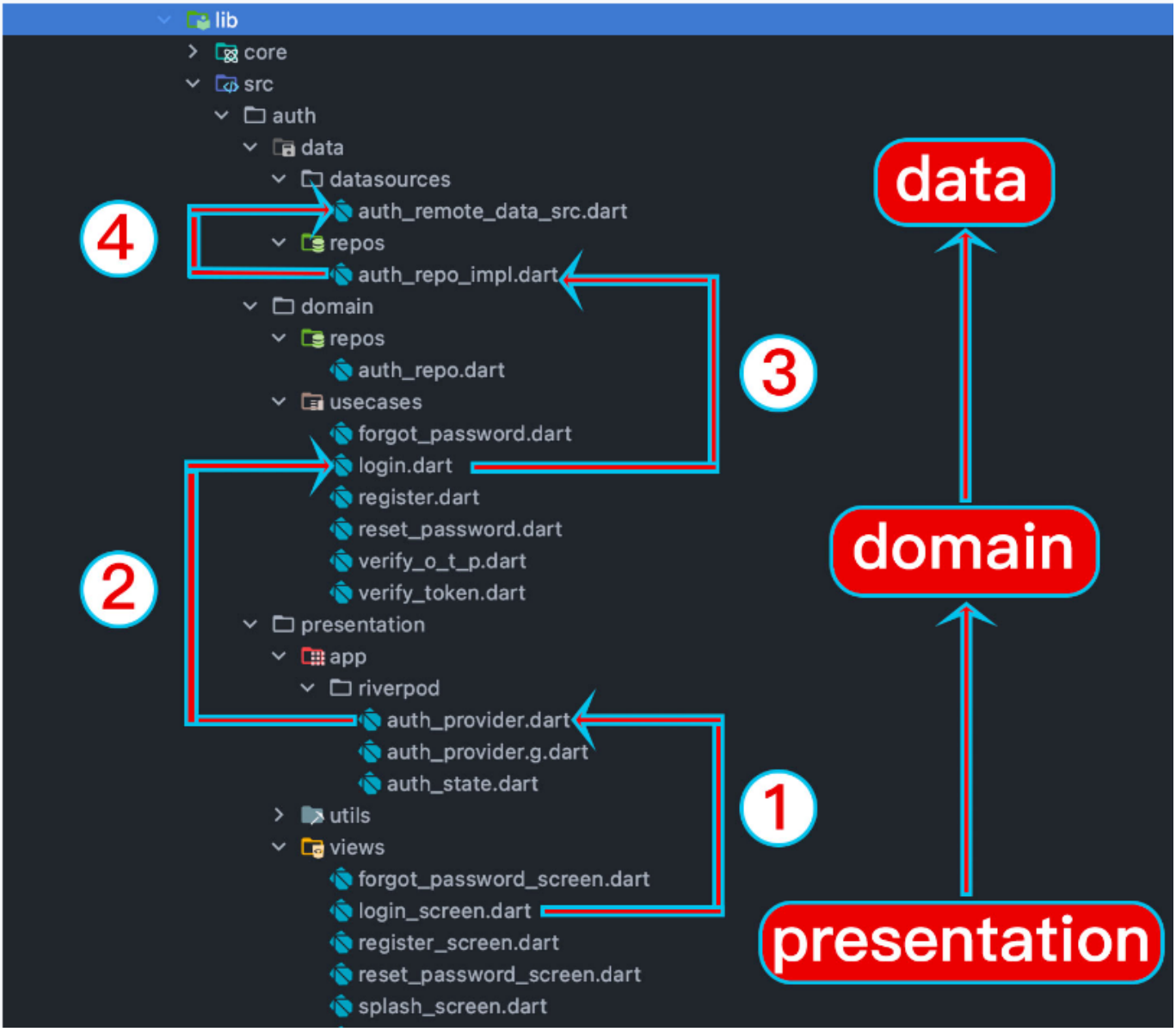
- **Use Case:** Contains the application-specific business rules and logic. Use cases represent high-level actions or operations that can be performed in the application.
- **Entity:** Represents the business objects or concepts in the application. Entities encapsulate enterprise-wide business rules and behaviors.
- **Repository (Interface**): Defines an interface or contract for interacting with data sources. Repositories abstract the data access logic from the domain layer, allowing it to remain independent of specific data storage implementations.

Data Layer:

- **Repository (Implementation):** Implements the repository interfaces defined in the domain layer. Repositories in the data layer are responsible for interacting with data sources such as databases and APIs.
- **Model (extends Entity**): Represents the data structures used to transfer data between the domain layer and data sources. Models typically extend the domain entities to ensure consistency across layers.
- **Data Sources (DB and APIs):** Represents the external data sources such as databases (DB) and application programming interfaces (APIs) used to fetch or store data.

Overlook on Clean Architecture and SOLID Principles Within Flutter Project:




lib
core
src
auth
data
datasources
auth_remote_data_src.dart
repos
auth_repo_impl.dart
domain
repos
auth_repo.dart
usecases
forgot_password.dart
login.dart
register.dart
reset_password.dart
verify_o_t_p.dart
verify_token.dart
presentation
app
riverpod
auth_provider.dart
auth_provider.g.dart
auth_state.dart
utils
views
forgot_password_screen.dart
login_screen.dart
register_screen.dart
reset_password_screen.dart
splash_screen.dart
4
3
2
1
data
domain
presentation

Poster:

AFEKA
The Academic College of Engineering in Tel Aviv

Software Engineering

Guardian View : AI-Powered Real -Time Threat Detection System

Dor rozen & omer liny
Advisor: Mr.Rozewaks Yoram

1. Problem description

Public spaces such as shopping malls, airports, and city streets face continuous security challenges from potential threats. Currently, security incident reports rely on human observers or real-time monitoring of security camera feeds. This approach has critical limitations:

- Incidents may be missed due to lack of witnesses or momentary inattention to specific camera feeds.
- Human reporting introduces delays in communicating crucial details like location and nature of the threat and even problems speaking in the same language or there is noise in the place.
- Time lost in explaining the situation to authorities can be critical in life-threatening scenarios.

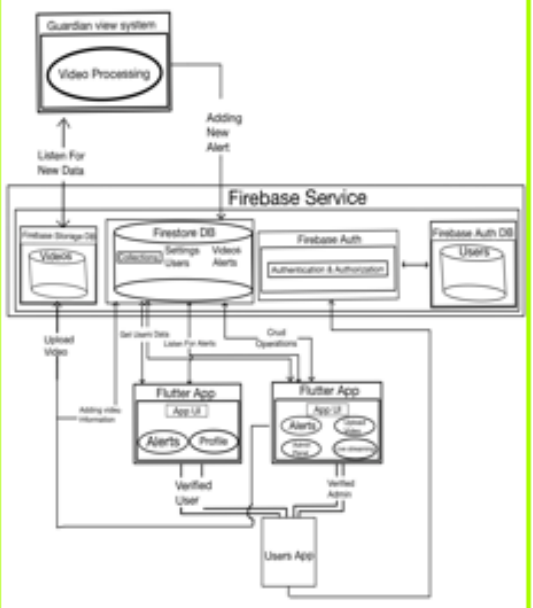
2. Research goals

- Implementing and developing an AI-based system for continuous, automated threat detection across multiple video feeds.
- Providing instant, detailed alerts to relevant authorities, eliminating delays in threat reporting, delivering precise information about the incident's location and reduce reaction time.

The project's goal is to significantly reduce response times to potential security threats recognizing that every second saved can be crucial .

3. Architecture

- The architecture chosen for the project is Event-Driven, as it deals with real-time data processing and alert generation. This architecture allows for efficient handling of incoming data and quick responses to events.



4. Technologies

- Team management tools - Monday, PlantUML, WhatsApp, Microsoft Teams Git.
- Backend development technologies: Python Firebase, VSCode, Ultralytics.
- Client development technologies: Android Studio and Flutter.

5. Development process

- Project managed with Monday.com for task tracking and Git for version control.
- Developed an AI-powered surveillance system for real-time threat detection in public spaces.
- Implemented YOLOv8 model for accurate and fast object detection in video feeds.
- Designed and developed a Flutter-based frontend for a user-friendly interface, enabling security personnel to receive instant alerts and visual verification of threats.
- Integrated Firebase services (Firestore, Auth, Storage) for robust data management, user authentication, and video storage.
- Developed a Python backend to process video data, interact with the YOLO model, and communicate with Firebase services.
- Implemented a real-time alert system to notify security personnel of potential threats instantly.
- Created a scalable architecture using Docker for consistent development and deployment environments.
- Designed and implemented a Firestore database structure to efficiently manage users, videos, settings, and alerts.
- Developed a video processing service to analyze uploaded and live-streamed video data for potential security threats.
- Implemented location-based alerting to notify near to detected threats.

6. Conclusions

- With the integration of AI-powered object detection and real-time video processing, we can efficiently identify potential security threats in public spaces.
- Our main challenge was to train the classification model to distinguish between regular videos and real threats in it.
- Our system features an intuitive interface with streamlined pages, allowing security personnel to quickly access alerts, view threat details, and respond to incidents efficiently. The system's ability to provide instant, location-based alerts to nearby responders significantly enhances the speed and effectiveness of security responses in public spaces.