UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Designing a surveillance system using Web camera

Student: Darian-Adelin Bojan

Structure of Computer Systems Project

Project supervisor: Madalin Ioan Neagu

Contents

Introduction

1.1 Context

This project has a particular goal to introduce the surveillance system using web camera as a basic element for surveillance. This system should capture real-time video feeds and allow you to monitor and sense activity around a certain space. The system consists of both hardware and software.

1.2 Objectives

The system needs to be built using programming languages such as Java, C, C++, C#. The specific objectives of the project are as follows:

- Motion detection
- Real-time Video Streaming
- User-interface design
- Optimization and performance
- Data Storage and retrieval

Bibliographic research

2.1 What is a surveillance system?

A surveillance system is a combination of hardware and software that observes, records, and analyzes audio or visual data to protect a selected area. Such systems usually include cameras, sensors and software applications that can analyze the imagery in real-time and generate alerts when required. Contemporary surveillance systems use a variety of innovations from web camera to computer Vision and real-time streaming methods.

2.2 Motion detection in Surveillance Systems

Motion detection algorithms examine frames taken by the cameras to find differences between consecutive images. Where simple implementations take the easy route and compare pixels to find a less-than-perfect approximation of motion, more sophisticated approaches utilize optical flow in combination with machine learning techniques to ascertain whether or not movement is taking place.

2.3 Real-time streaming protocols

Streaming protocols send live video feeds to the users through surveillance systems. Video data is usually transmitted using protocols with lower latency such as Real-Time Streaming Protocol (RTSP) and Web Real-Time Communication (WebRTC).

Analysis

3.1 Project proposal

Features of the final Surveillance System:

1. Real-Time Video Feed:
    - Streaming live video from a webcam to monitor movement in an area.
    - Integration with JavaFX to display the real-time video feed to user directly.
2. Motion Detection:
    - Detection of motion in the camera's field of view using pixel difference techniques.
    - Adjustable sensitivity to minimize false positives in various lighting and environmental conditions.
3. User Interface with Configurable Options:
    - A JavaFX interface to start/stop the video feed, modify how sensitive Motion should be and other settings
    - Real-time display of video feed with overlayed alerts for easy monitoring.
4. Optimization for Low Latency:
    - Efficient video processing pipeline to ensure minimal lag in the video feed.
    - Frame rate adjustment options to balance video quality and processing speed.
5. Test Scenarios to Validate Motion Detection:
    - A series of test cases designed to evaluate the system's reliability in detecting motion under various conditions, such as changing light, different movement speeds, and low/no motion.

- Test programs to challenge and refine motion detection sensitivity and alert accuracy.

3.2 Project analysis

3.2.1 Functional Requirements

The primary functional requirements of the surveillance system include **Real-Time Video Capture, Motion Detection, User Interface**, **Data Storage**

3.2.2 Technical Requirements:

1. **Programming Languages and Tools**:
    - **Java**: Primary programming language for the application.
    - **JavaFX**: For building the graphical user interface.
    - **OpenCV**: For real-time video processing and motion detection.
2. **Hardware Requirements**:
    - **Webcam**: A basic USB or built-in webcam to capture video.
    - **Computer**: System requirements depend on the efficiency of frame processing and memory; however, a standard laptop or desktop with a dual-core CPU should suffice.
3. **Software Requirements**:
    - **Java Development Kit (JDK)**: Compatible with JavaFX and OpenCV.
    - **JavaFX SDK**: For the user interface components.
    - **OpenCV Library**: Provides tools for video capture and image processing.

3.2.3 System Architecture

The system architecture is based on a modular structure, with the following core modules:

1. **Video Capture Module**: Uses OpenCV's VideoCapture class to initialize the webcam and retrieve real-time frames.

2. **Motion Detection Module**: Analyzes each frame using pixel comparison techniques to detect motion. Advanced algorithms could be integrated for higher accuracy, such as background subtraction or machine learning-based motion detection.
3. **User Interface Module**: Built with JavaFX, this module provides an ImageView for live video display, along with buttons for configuring the application.
4. **Notification Module**: Responsible for alerting the user when motion is detected, using visual cues within the application.

Design

The design of the **Surveillance System** focuses on creating software components that interact to capture, process, and display live video from a webcam while enabling motion detection and alert features. This design involves modular class structures for separation of concerns, making it easier to maintain and extend functionality.

4.1.1 Class Design

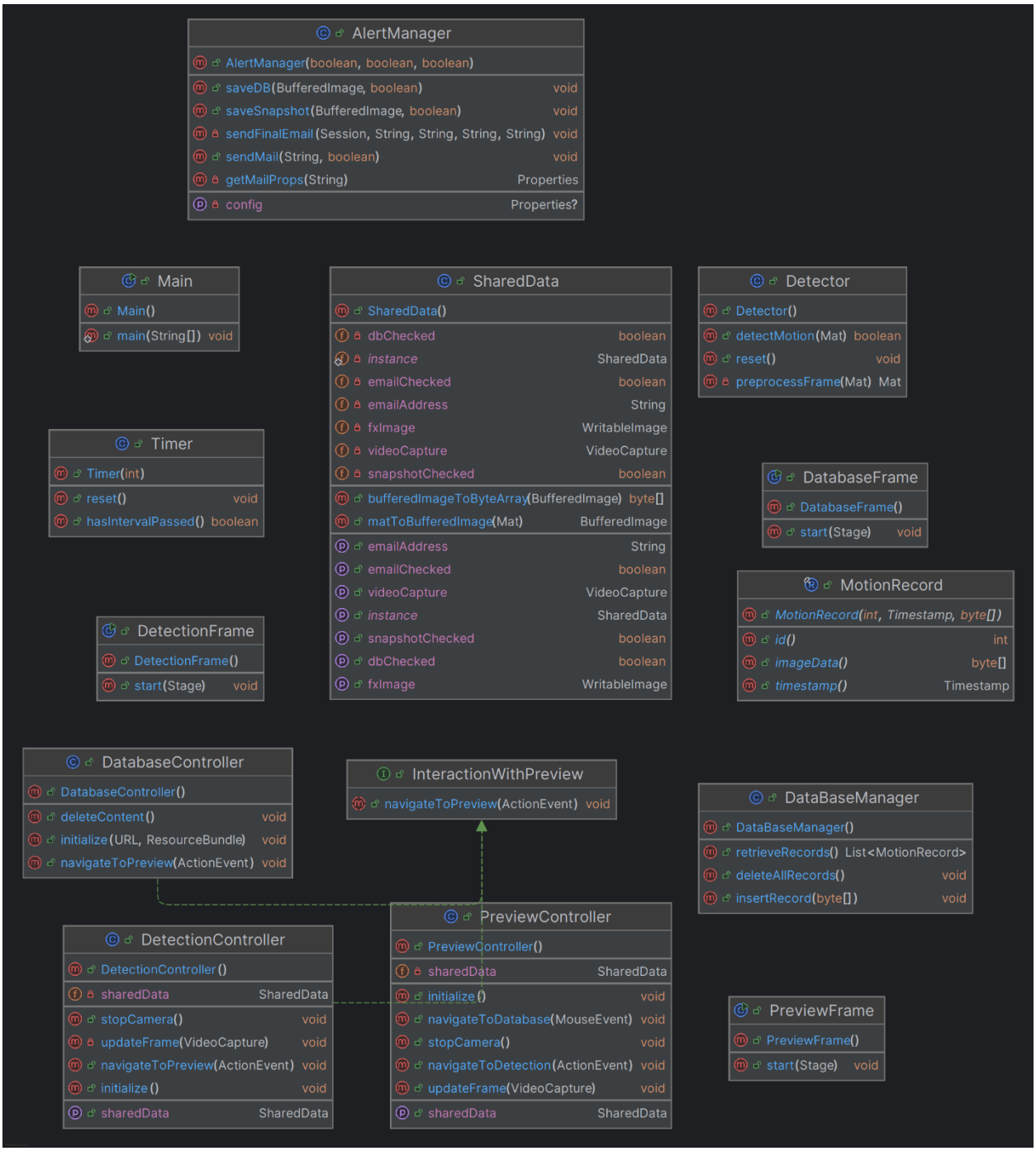
This system has several classes which help in creating the user-interface and the logic for motion detection.

Class Diagram

The class diagram below shows the relationships among the core classes in the surveillance system:

- **Main**: Main class to initialize and start the surveillance system.
- **PreviewController**: Contains main functions for controlling the Preview Class meant for UI.
- **DetectionController**: Has methods regarding the camera update, stopping camera or navigation within the DetectionFrame.

- **Detector**: Detects motion within video frames by comparing differences between consecutive frames. Contains a certain threshold.
- **PreviewFrame**: Builds the graphical user interface using JavaFX and displays the real-time video feed to the user, also enabling the user to choose from a list of settings before navigating to DetectionFrame.
- **DetectionFrame**: Creates the graphical user interface and checks if there is any motion during this time. If yes, based on the checked boxes from previewFrame, the user will be able to see the results (email or database).
- **AlertManager**: Handles notifications and alerts, such as returning the result in a database or receiving a text on email.
- **DatabaseManager**: Creates the connection to a MySQL database.
- **Timer**: creates a duration between 2 actions recorded by the detection mechanism.


AlertManager
AlertManager(boolean, boolean, boolean)
saveDB(BufferedImage, boolean) void
saveSnapshot(BufferedImage, boolean) void
sendFinalEmail(Session, String, String, String, String) void
sendMail(String, boolean) void
getMailProps(String) Properties
config Properties?
Main
Main()
main(String[]) void
SharedData
SharedData()
dbChecked boolean
instance SharedData
emailChecked boolean
emailAddress String
fxImage WritableImage
videoCapture VideoCapture
snapshotChecked boolean
bufferedImageToByteArray(BufferedImage) byte[]
matToBufferedImage(Mat) BufferedImage
emailAddress String
emailChecked boolean
videoCapture VideoCapture
instance SharedData
snapshotChecked boolean
dbChecked boolean
fxImage WritableImage
Detector
Detector()
detectMotion(Mat) boolean
reset() void
preprocessFrame(Mat) Mat
Timer
Timer(int)
reset() void
hasIntervalPassed() boolean
DatabaseFrame
DatabaseFrame()
start(Stage) void
MotionRecord
MotionRecord(int, Timestamp, byte[])
id() int
imageData() byte[]
timestamp() Timestamp
DetectionFrame
DetectionFrame()
start(Stage) void
DatabaseController
DatabaseController()
deleteContent() void
initialize(URL, ResourceBundle) void
navigateToPreview(ActionEvent) void
InteractionWithPreview
navigateToPreview(ActionEvent) void
DataBaseManager
DataBaseManager()
retrieveRecords() List<MotionRecord>
deleteAllRecords() void
insertRecord(byte[]) void
DetectionController
DetectionController()
sharedData SharedData
stopCamera() void
updateFrame(VideoCapture) void
navigateToPreview(ActionEvent) void
initialize() void
sharedData SharedData
PreviewController
PreviewController()
sharedData SharedData
initialize() void
navigateToDatabase(MouseEvent) void
stopCamera() void
navigateToDetection(ActionEvent) void
updateFrame(VideoCapture) void
sharedData SharedData
PreviewFrame
PreviewFrame()
start(Stage) void

4.1.2 Main classes Descriptions

1. **Main**:
    - The main class starts by running the PreviewFrame

    The main class starts by running the PreviewFrame.

    - **Method**:
        - public static void main(String[] args): Initializes first frame and starts the application.
2. **PreviewFrame**:
    - PreviewFrame is the design class which displays a webcam, such that the user can test the webcam before starting the system
    - It includes options like saving snapshot locally, saving data to a database or sending an email with a warning.
    - **Method**:
        - start(): Sets up the user interface for previewing.
3. **DetectionFrame**:
    - The detection frame contains the webcam view and a button for coming back to preview mode and possibly to make other changes.
    - **Method**:
        - start(): Sets up the user interface for detection.
4. **MotionDetector**:
    - Responsible for detecting motion by analyzing consecutive frames.
    - **Attributes**:
        - previousFrame: It is used to compare the current frame to the previous one, therefore we can detect motion.

- **Methods**:
    - detectMotion(): Compares consecutive frames to determine if motion has occurred.

5. **Controller classes**:
    - Provides control for the design classes. It sends information to other classes, initializes instances of sharedData. Vital for processing information from the user.
    - **Attributes**:
        - webcamView: an imageView, the place where the webcam will be located.
        - motionDetector: Instance of the Detector class.
        - alertManager: Instance of the AlertManager class.
        - Timer: Instance of the Timer class
    - **Methods**:
        - updateFrame(): Continuously reads frames from the camera
        - navigateToPreview(VideoCapture): Navigates from detection to preview
        - navigateToDetection(VideoCapture): Navigates from preview to detection
        - stopCamera(): Stops detection mechanism.
        - setSharedData(): setter for sharedData
        - deleteContent(): deletes all the database elements

4.1.3 Notification System

- **AlertManager**: Sends email or in-app notifications when motion is detected.
    - **Attributes**:
        - emailChecked: boolean to see whether an email should be sent.
        - snapshotChecked: boolean flag to save a snapshot locally or not.

        - dbChecked: boolean flag to save to a database the timestamp of the motion
    - **Methods**:
        - sendEmailAlert(String message): Sends an email when an alert is triggered.
        - saveSnapshot(String message): Saves a screenshot of a movement by storing it locally.
        - sendToDatabase(): Sends the current timestamp when the motion was detected and a snapshot to the database.
- **Timer**: Sets a duration between two motion detection alerts.
    - **Attributes**:
        - lastActionTime: stores the timestamp of the last action
        - interval: it defines the minimum time that must pass between actions for the hasIntervalPassed() method to return true
    - **Methods**:
        - hasIntervalPassed(): checks whether the specified time interval has passed since the last action
        - reset(): resets the timer by updating lastActionTime to the current system time.

- **Database manager**: Creates the database connection and updates the database with records. Also can delete the records from the database.
    - **Attributes**:
        - DB_URL: JDBC URL to connect to a MySQL database
        - DB_USER: username for authentication
        - DB_PASSWORD: password for authentication

Implementation

The implementation chapter covers the code structure and approaches used to develop key components of the surveillance system.

5.1 Video Capture and Display

The video capture is achieved using OpenCV's VideoCapture class, which retrieves frames from the webcam in real time. The SystemController class initializes this capture, converts frames to JavaFX-compatible images, and updates the display in a separate thread.

Code Snippet for Video Capture Initialization

```
capture = new VideoCapture(0, Videoio.CAP_DSHOW);

if (!capture.isOpened()) {

    System.out.println("Error: Could not open the webcam.");

    return;

}

capture.set(Videoio.CAP_PROP_FRAME_WIDTH, 720);

capture.set(Videoio.CAP_PROP_FRAME_HEIGHT, 540);
```

5.2 Image Conversion for Display

The matToBufferedImage method converts Mat frames to BufferedImage, compatible with JavaFX's WritableImage. This code from matToBufferedImage handles the conversion of an OpenCV Mat object to BufferedImage:

Code Snippet for Image Conversion

```
private BufferedImage matToBufferedImage(Mat mat) {

    if (mat == null || mat.empty()) {

        return null;

    }

    int type = BufferedImage.TYPE_BYTE_GRAY;

    if (mat.channels() > 1) {

        type = BufferedImage.TYPE_3BYTE_BGR;
```

```
    }

    int bufferSize = mat.channels() * mat.cols() * mat.rows();

    byte[] buffer = new byte[bufferSize];

    mat.get(0, 0, buffer);

    BufferedImage image = new BufferedImage(mat.cols(), mat.rows(), type);

    final byte[] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();

    System.arraycopy(buffer, 0, targetPixels, 0, buffer.length);

    return image;

}
```

5.3 Email Notification Setup

Email notifications are handled by the AlertManager class, which sends an alert email to a configured address when motion is detected.

```
public void sendMail(String address, boolean movementDetected) {
    if(emailChecked && movementDetected) {
        Properties config = getConfig();

        assert config != null;
        String host = config.getProperty("host");
        String from = config.getProperty("from");
        String password = config.getProperty("password");

        String subject = "Surveillance Alert: Motion Detected";
        String warningMessage = "Warning! The surveillance system has detected motion within the monitored
area.";

        Properties mailProps = getMailProps(host);

        Session session = Session.getInstance(mailProps, new jakarta.mail.Authenticator() {
            @Override
            protected jakarta.mail.PasswordAuthentication getPasswordAuthentication() {
                return new jakarta.mail.PasswordAuthentication(from, password);
            }
        });

        sendFinalEmail(session, from, address, subject, warningMessage);
    }
```

5.4 Database configuration

Public method for creating the connection to the database. It takes a parameter imageData, which is a byte array and will be stored in the MySQL database as a BLOB.

```
public void insertRecord(byte[] imageData) {
    Connection conn = null;
    PreparedStatement pstmt = null;

    try {
        conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);

        String sql = "INSERT INTO motion (created_at, image_data) VALUES (?, ?)";

        pstmt = conn.prepareStatement(sql);

        Timestamp currentTimestamp = new Timestamp(System.currentTimeMillis());
        pstmt.setTimestamp(1, currentTimestamp);

        if (imageData != null) {
            pstmt.setBytes(2, imageData);
        } else {
            pstmt.setNull(2, Types.BLOB);
        }

        int rowsAffected = pstmt.executeUpdate();
        if (rowsAffected > 0) {
            JOptionPane.showMessageDialog(null, "Record added to the database successfully.");
        } else {
            JOptionPane.showMessageDialog(null, "Failed to add the record to the database.");
        }

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (pstmt != null) {
                pstmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
```

```
    }
}
```

5.5 Timer

The project also includes a timer which will permit reevaluating the possibility of a movement in each 10 seconds (so that a single message can be sent via email, or store a single record in a decent interval of time).

```
public boolean hasIntervalPassed() {
    long currentTime = System.currentTimeMillis();
    return (currentTime - lastActionTime) >= interval;
}

public void reset() {
    lastActionTime = System.currentTimeMillis();
}
```

5.6 Saving snapshot locally

This method is used to save a screenshot of the current frame from the webcam in case of a movement.

```
public void saveSnapshot(BufferedImage image, boolean movementDetected) {
    if (snapshotChecked && movementDetected && image != null) {
        try {
            String filePath = "../SurveillanceSystem/snapshots/snapshot_" + System.currentTimeMillis() +
".png";
            File outputFile = new File(filePath);
            outputFile.getParentFile().mkdirs();
            ImageIO.write(image, "png", outputFile);
            System.out.println("Snapshot saved to: " + filePath);
        } catch (IOException e) {
            System.err.println("Error saving snapshot: " + e.getMessage());
        }
    }
}
```

5.7 Seeing the database records

Each record is going to be seen on a dashboard on a separate frame (“Database Frame”).

```
public List<MotionRecord> retrieveRecords() {
    List<MotionRecord> records = new ArrayList<>();
    try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
         PreparedStatement pstmt = conn.prepareStatement("SELECT id, created_at, image_data FROM motion");
         ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            int id = rs.getInt("id");
            Timestamp timestamp = rs.getTimestamp("created_at");
            byte[] imageData = rs.getBytes("image_data");
            records.add(new MotionRecord(id, timestamp, imageData));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return records;

@Override
public void initialize(URL location, ResourceBundle resources) {
    DataBaseManager manager = new DataBaseManager();
    List<MotionRecord> records = manager.retrieveRecords();

    for (MotionRecord record : records) {
        GridPane grid = new GridPane();
        grid.setAlignment(Pos.CENTER);
        grid.setHgap(20);
        grid.setVgap(10);

        ColumnConstraints col1 = new ColumnConstraints();
        col1.setHalignment(HPos.CENTER);
        ColumnConstraints col2 = new ColumnConstraints();
        col2.setHalignment(HPos.CENTER);
        ColumnConstraints col3 = new ColumnConstraints();
        col3.setHalignment(HPos.CENTER);
        grid.getColumnConstraints().addAll(col1, col2, col3);

        Text idText = new Text("ID: " + record.id());
        Text timestampText = new Text("Timestamp: " + record.timestamp());
        ImageView imageView = new ImageView();

        if (record.imageData() != null) {
            Image img = new Image(new ByteArrayInputStream(record.imageData()));
            imageView.setImage(img);
            imageView.setFitWidth(150);
            imageView.setFitHeight(100);
            imageView.setPreserveRatio(true);
        }
```

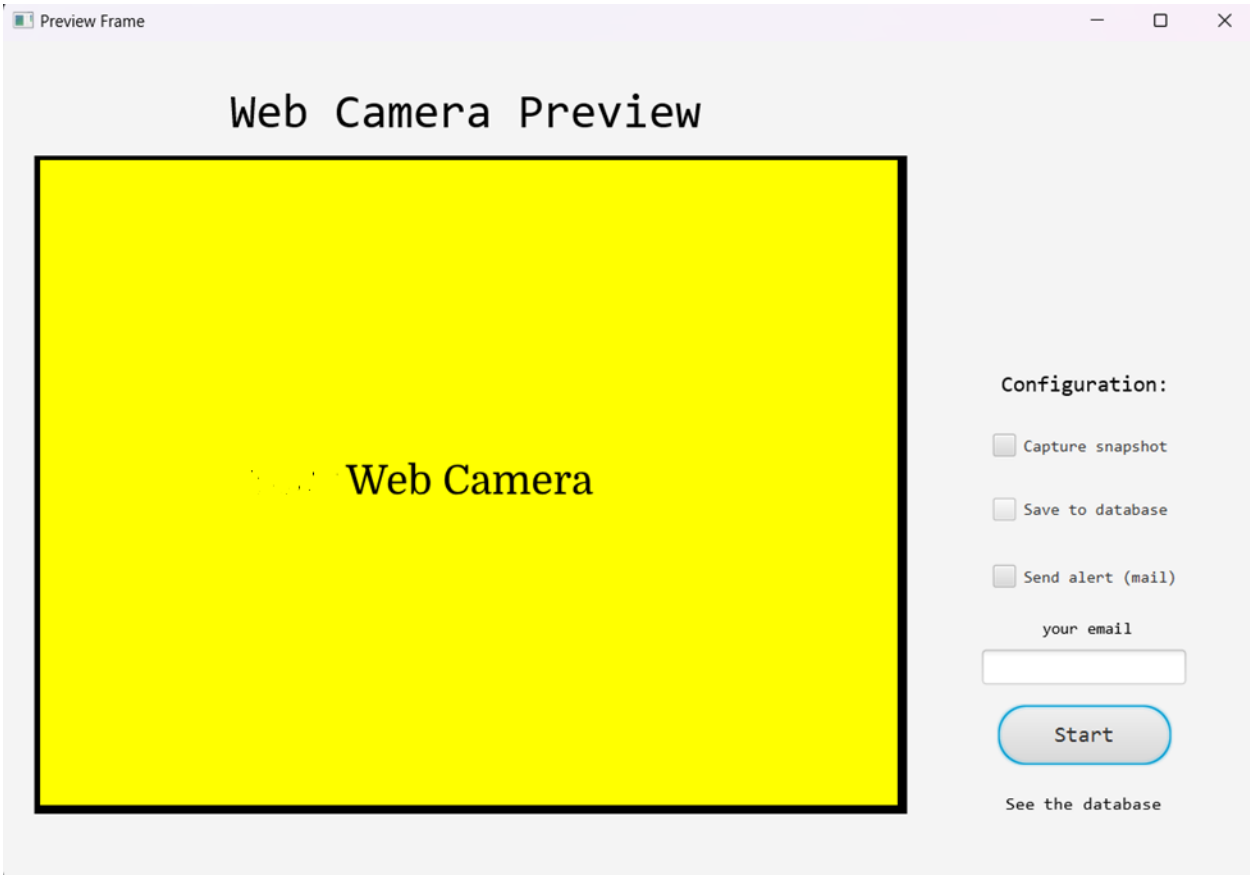
```
        grid.add(idText, 0, 0);
        grid.add(timestampText, 1, 0);
        grid.add(imageView, 2, 0);

        recordsContainer.getChildren().add(grid);
    }
}
```

5.7 Deleting content

Each record is going to be deleted, then the id will begin to autoincrement again from 1.

```
public void deleteContent() {
    DataBaseManager manager = new DataBaseManager();
    try {
        int result = JOptionPane.showConfirmDialog(
                null,
                "Are you sure you want to delete all records?",
                "Confirm",
                JOptionPane.YES_NO_OPTION
        );
        if (result == JOptionPane.YES_OPTION) {
            manager.deleteAllRecords();
            JOptionPane.showMessageDialog(null, "Records deleted successfully. You can go back to Preview
Frame");
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e.getMessage() + ", could not delete content");
    }
}
```

Testing the implementation



For each motion detected, a print statement is displayed on the console. To not include too many records in the database or to save too many pictures locally, a 10-second timer is implemented. This timer ensures that only one motion record is saved within a 10-second interval, even if multiple motions are detected during that time frame.

The message shown in that case will be: “Motion detected, but waiting for the timer”.

There are 3 cases in which we need to test the application: capturing a snapshot, saving information to database, sending an alert via email.

- **Capturing a snapshot**

We will consider only the first case now. We will consider a moving ball to be the experiment for our cases.

```
Motion detected, but waiting for timer.
Snapshot saved to: ../SurveillanceSystem/snapshots/snapshot_1734280031282.png
Motion detected, but waiting for timer.
Motion detected, but waiting for timer.
```

Result:

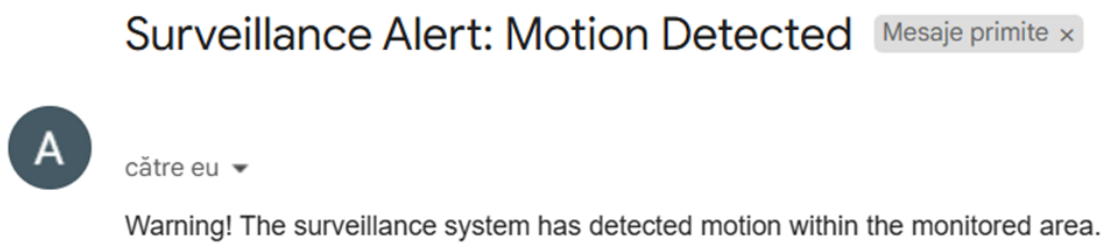
In the folder called snapshots of the Java application, each motion detected will be converted into a .png and saved locally.

- **Saving data to database**

Result:

Each activity detected by the system is saved in the database as can be seen in the picture above. The time interval between two motion detections is at least 10 seconds.

- **Sending mail in case of an alert.**




```
Email sent successfully to name@gmail.com
Motion detected, but waiting for timer.
Motion detected, but waiting for timer.
```

Result:

A confirmation text is displayed on the console and the alert can be seen (it has been sent successfully via mail).

All in all, the three tests were vital in concluding that the application works well, and it accomplishes the goals set from the start. These results provide a strong foundation for scaling the application and exploring additional features to enhance its functionality further.

Bibliography

https://www.efour.co/custom-security-surveillance-system/

https://www.researchgate.net/

https://dvss.weebly.com/

https://vmsoftwarehouse.com/