Main.java

```
package j1;

import org.opencv.core.Mat;
import org.opencv.videoio.VideoCapture;
import utils.CameraUtils;

public class Main {
    public static void main(String[] args) {
        // Initialize camera and video processor
        VideoCapture camera = CameraUtils.initCamera(0);
        VideoProcessor processor = new VideoProcessor();

        if (camera != null) {
            while (true) {
                Mat frame = CameraUtils.captureFrame(camera);
                if (frame != null) {
                    // Face detection
                    Mat processedFrame = processor.detectFaces(frame);

                    // Object detection
                    processedFrame = processor.detectObjects(processedFrame);

                    // Motion tracking
                    processedFrame = processor.trackMotion(processedFrame);

                    // Display the processed frame (implementation of display method
needed)
                } else {
                    System.out.println("No frame captured.");
                    break;
                }
            }
            camera.release();
        }
    }
}
```

VideoProcessor.java

package j1;

import org.opencv.core.Mat;

import org.opencv.core.MatOfPoint;

```
import org.opencv.core.MatOfRect;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;
import org.opencv.video.BackgroundSubtractorMOG2;
import org.opencv.video.Video;

import java.util.ArrayList;
import java.util.List;

public class VideoProcessor {
    private CascadeClassifier faceDetector;
    private BackgroundSubtractorMOG2 backgroundSubtractor;
    private List<Point> previousCenters;

    public VideoProcessor() {
        // Load the classifier from the resources folder
        String xmlFile =
getClass().getResource("src/main/resources/haarcascade_frontalface_alt.xml").getPath();
        faceDetector = new CascadeClassifier(xmlFile);

        // Initialize background subtractor for object detection
        backgroundSubtractor = Video.createBackgroundSubtractorMOG2();
        previousCenters = new ArrayList<>();
    }
```

```
    public Mat detectFaces(Mat frame) {
        MatOfRect faceDetections = new MatOfRect();
        faceDetector.detectMultiScale(frame, faceDetections);

        // Draw rectangles around detected faces
        for (Rect rect : faceDetections.toArray()) {
            Imgproc.rectangle(
                frame,
                new Point(rect.x, rect.y),
                new Point(rect.x + rect.width, rect.y + rect.height),
                new Scalar(0, 255, 0),
                3
            );
        }

        return frame;
    }

    public Mat detectObjects(Mat frame) {
        Mat foregroundMask = new Mat();
        backgroundSubtractor.apply(frame, foregroundMask);

        // Find contours of moving objects
        List<MatOfPoint> contours = new ArrayList<>();
        Imgproc.findContours(foregroundMask, contours, new Mat(), Imgproc.RETR_EXTERNAL,
Imgproc.CHAIN_APPROX_SIMPLE);
```

```
        for (int i = 0; i < contours.size(); i++) {
            Rect boundingRect = Imgproc.boundingRect(contours.get(i));
            if (boundingRect.area() > 500) { // Filter small objects
                Imgproc.rectangle(
                    frame,
                    new Point(boundingRect.x, boundingRect.y),
                    new Point(boundingRect.x + boundingRect.width, boundingRect.y +
boundingRect.height),
                    new Scalar(255, 0, 0),
                    2
                );
            }
        }

        return frame;
    }

    public Mat trackMotion(Mat frame) {
        MatOfRect faceDetections = new MatOfRect();
        faceDetector.detectMultiScale(frame, faceDetections);

        List<Point> currentCenters = new ArrayList<>();
        for (Rect rect : faceDetections.toArray()) {
            Point center = new Point(rect.x + rect.width / 2, rect.y + rect.height / 2);
            currentCenters.add(center);
        }
```

```
        if (!previousCenters.isEmpty()) {
            for (int i = 0; i < Math.min(previousCenters.size(), currentCenters.size()); i++) {
                Point prev = previousCenters.get(i);
                Point curr = currentCenters.get(i);

                Imgproc.line(frame, prev, curr, new Scalar(0, 0, 255), 2);
            }
        }

        previousCenters = currentCenters;

        return frame;
    }
}
```

CameraUtils.java

```java
package utils;

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.videoio.VideoCapture;

public class CameraUtils {
    static {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    }

    public static VideoCapture initCamera(int cameraIndex) {
        VideoCapture camera = new VideoCapture(cameraIndex);
        if (!camera.isOpened()) {
            System.out.println("Error: Could not open camera.");
            return null;
        }
        return camera;
    }
```

```
    public static Mat captureFrame(VideoCapture camera) {
        Mat frame = new Mat();
        if (camera.read(frame)) {
            return frame;
        }
        return null;
    }
}
```

VideoStream.js

```
import React, { useEffect, useState } from 'react';

const VideoStream = () => {
    const [imageSrc, setImageSrc] = useState('');

    useEffect(() => {
        const fetchVideoStream = () => {
            fetch('http://localhost:8080/video-stream')
                .then(response => response.blob())
                .then(blob => {
                    const imageObjectURL = URL.createObjectURL(blob);
                    setImageSrc(imageObjectURL);
                })
                .catch(error => console.error('Error fetching video stream:',
error));
        };

        // Fetch a new frame every 100ms (10 frames per second)
        const intervalId = setInterval(fetchVideoStream, 100);

        return () => clearInterval(intervalId); // Cleanup interval on component
unmount
    }, []);

    return (
        <div>
            <h1>Real-Time Video Stream</h1>
            <img src={imageSrc} alt="Video Stream" style={{ width: '100%',
maxHeight: '500px' }} />
        </div>
    );
};
```

```
export default VideoStream;
```

App.js

```
import React from 'react';
import './App.css';
import VideoStream from './VideoStream';

function App() {
    return (
        <div className="App">
            <header className="App-header">
                <VideoStream />
            </header>
        </div>
    );
}

export default App;
```