**Title: Report Document**

**Author: Nam Cap**

**Student ID: a1907345**

1. **Functional Reactive Programming (FRP) Usage:**

* I use of the Sodium library for FRP seems appropriate. I am creating streams and cells, and using them to react to events. This is a core part of FRP, and your implementation aligns well with this paradigm.
* Ensure that all data processing and transformations required for the GPS data (like filtering and distance calculations) are done reactively using the Sodium library.

1. **Implement the refactored GUI setup:** We'll ensure that the refactoring of createAndShowGUI and related methods is integrated into your GpsGUI class.
2. **Implement filter logic:** We'll write the logic for filtering the stream of GPS events based on user input.

private static void setupFilterButtonListener(STextField latitudeField, STextField longitudeField,

            SButton applyButton, JLabel statusLabel) {

        // Apply filter logic on button click

        applyButton.sClicked.listen(ignored -> {

            String latStr = latitudeField.text.sample().trim();

            String lonStr = longitudeField.text.sample().trim();

            try {

                double lat = Double.parseDouble(latStr);

                double lon = Double.parseDouble(lonStr);

                // Validate latitude and longitude values

                if (lat < -90 || lat > 90 || lon < -180 || lon > 180) {

                    JOptionPane.showMessageDialog(frame,

                            "Latitude must be between -90 and 90 and longitude between -180 and 180.");

                    statusLabel.setText("Current filter: Invalid");

                    return; // Exit the method if the input is invalid

                }

                // If the input is valid, update the filter status label and apply the filter

                filterStatusLabel.setText("Current filter: Lat " + lat + ", Lon " + lon);

                // Filter the stream for events within the specified latitude and longitude

                Stream<GpsEvent> allEventsStream = combineAllTrackerStreams();

                Stream<GpsEvent> filteredStream = allEventsStream

                        .filter(event -> Math.abs(event.getLatitude() - lat) < LATITUDE\_THRESHOLD &&

                                Math.abs(event.getLongitude() - lon) < LONGITUDE\_THRESHOLD);

                // Update the combined data display with the filtered data

                filteredStream

                        .map(event -> "Tracker " + event.getTrackerId() + ": Lat " + event.getLatitude() + ", Lon "

                                + event.getLongitude())

                        .hold("No data")

                        .listen(filteredData -> combinedDataDisplay.setText(filteredData));

            } catch (NumberFormatException e) {

                // If parsing the double fails, show an error message

                JOptionPane.showMessageDialog(frame, "Invalid input for latitude or longitude.");

                statusLabel.setText("Current filter: Invalid");

            }

        });

    }

1. **Implement Distance Update Logic**

The distance update logic is likely to be complex as it involves geospatial calculations. You'll need to implement a method to calculate the distance between two GPS coordinates, possibly including altitude if required. The calculateDistance method should then be used to update the distance displayed for each tracker.

public static double calculateDistance(GpsEvent startEvent, GpsEvent endEvent) {

        final int R = 6371; // Radius of the Earth in kilometers

        double latDistance = Math.toRadians(endEvent.getLatitude() - startEvent.getLatitude());

        double lonDistance = Math.toRadians(endEvent.getLongitude() - startEvent.getLongitude());

        double a = Math.sin(latDistance / 2) \* Math.sin(latDistance / 2) +

                Math.cos(Math.toRadians(startEvent.getLatitude())) \* Math.cos(Math.toRadians(endEvent.getLatitude()))

                        \*

                        Math.sin(lonDistance / 2) \* Math.sin(lonDistance / 2);

        double c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

        double distance = R \* c; // convert to meters

        distance \*= 1000; // convert to meters

        return Math.round(distance);

    }

1. **Write test cases:** We'll write additional test cases in Gui\_Test.java to ensure the GUI behaves as expected.
2. **The setupPeriodicTasks method**

The setupPeriodicTasks method schedules a periodic task that will update each tracker's distance display every five minutes. This is an important piece of functionality that ensures the GUI reflects the most recent state of the system.

private static void setupPeriodicTasks() {

        ScheduledExecutorService executorService = Executors.newScheduledThreadPool(1);

        executorService.scheduleAtFixedRate(() -> {

            // For each tracker, update its distance display

            trackerDistances.forEach((trackerId, distance) -> {

                updateTrackerDistanceDisplay(trackerId, distance);

            });

        }, 0, 5, TimeUnit.MINUTES); // Schedules the task to run every 5 minutes

    }