Project 6 Update:

Project Title: Herd Safety

Team Members: Nova White, Kaleb Moore, Joseph Rizzo

Work Done:

Nova:

- Drafted the first code base structure in Android Studio, constructing the base starting point for our application in the file MapsActivity.java.
- Designed the initial UI for the home page in this file. This included implementing a functional, interactive map using the Google Maps SDK for Android Studio. A yellow ‘Report’ button was also included on the page, which links to the reporting tab.
- Also built the reporting tab in AlertFormPage.java, which pops up on the screen when the user clicks the ‘Report’ button. This tab contains an interactive text box, with the default text ‘Enter a description here’, as well as a ‘Cancel’ and ‘Submit’ button.
- When the ‘Submit’ button is clicked, implemented functionality to show the report confirmation page, contained in ReportConfirmationPage.java.
- Added alert types and alert factory per alert creation.
    - Also added alert type selection validation
- Added alert type processing for the database

Kaleb:

- Created Class Similarity checker than will eventually compare reports to determine if a new report is similar to others
    - Created Inherited classes CautionSimilairty, WarningSimilarity, CrimeSimilarity (not yet implemented as working on base code for superclass)
    - Defined methods for SimilarityChecker superclass

        - normalizeDescription(): takes a string and normalizes it using regex to be all lowercase and returns an array of strings
        - buildVectors(): takes in an array of words and populated the algorithms Hashtables with word counts to represent vector of original string
        - buildMasterTable(): Creates master table of 3 word vectors from classes vectors for hamming distance and cosine similarity for CrimeSimilarity and WarningSimilarity

<u>Joseph:</u>

- Created class DBHandler in DBHandler.java. This class inherits and extends from the built-in SQLiteOpenHelper class. It declares the name and version of the database, as well as all table and column names upon instantiation.
- Defined methods for DBHandler class.
    - onCreate(): Executes SQL queries to create tables.
    - addNewUser(): Needs refactoring in current state, will take UserModel object and insert user data into DB.
    - addNewAlert(): Only handles alert ID and description currently, but is functional. Takes AlertModel and inserts alert data into DB.
    - addNewAlertInFull(): Needs refactoring in current state, will take AlertModel and add all possible user data (more of first draft, while previous function is more practical currently).
    - deleteAllAlerts(): Executes SQL query to clear Alerts table. Used for testing.
    - onUpgrade(): Drops tables and resets DB. Used for testing.
- Created AlertModel class in AlertModel.java. This class was built to handle alerts and the data that comes with that. It holds the ID and description currently, but will be expanded upon in the future. Contains constructors, getters, setters, and an overridden toString() method for quick formatting.
- Slightly modified AlertFormPage.java and MapsActivity.java, including logic to check validity of user input, and to pull and store in DB if valid.

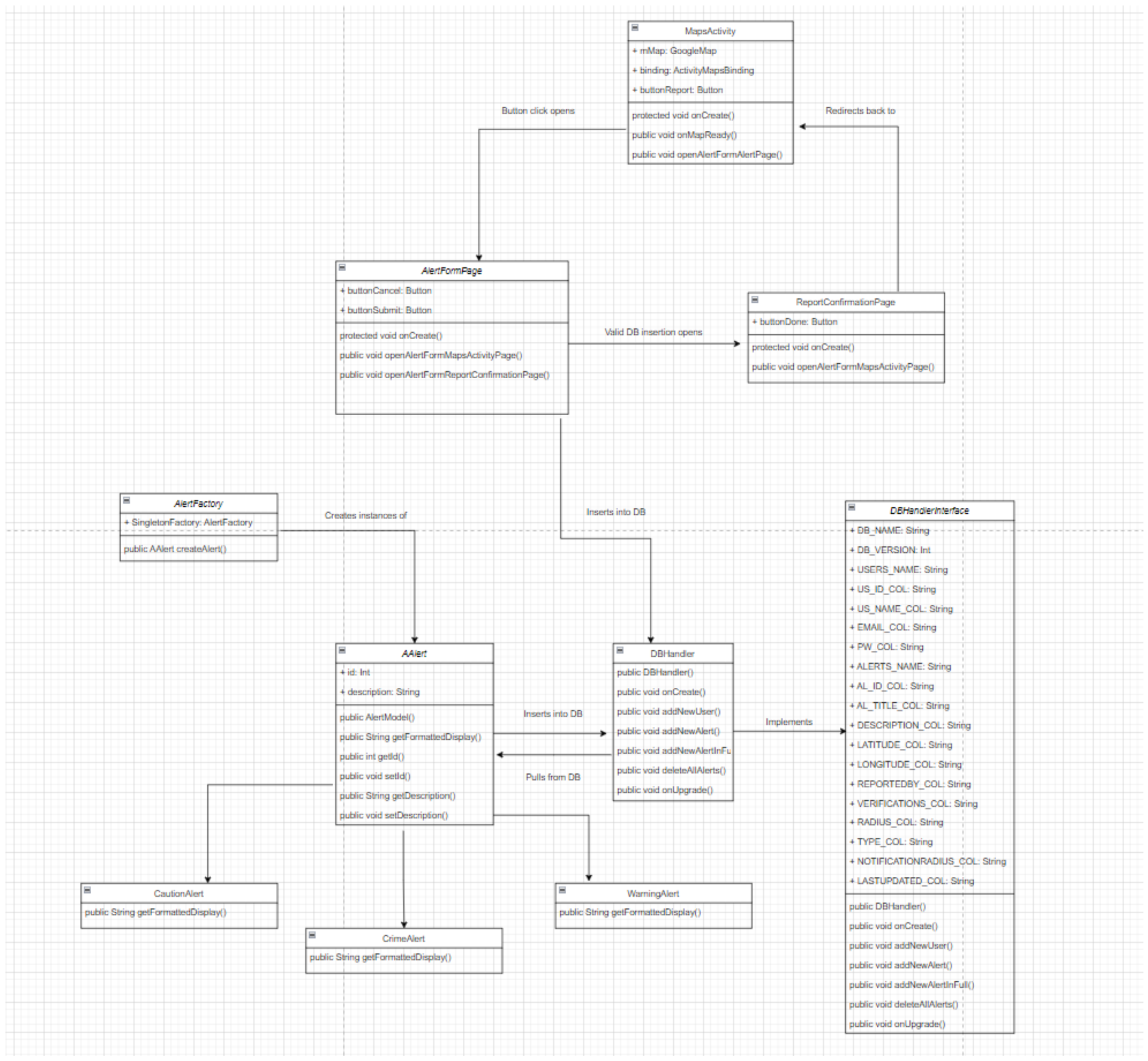
Changes/Issues Encountered:

- In our initial project proposal, we were discussing whether to use MySQL or PostgreSQL to implement our database, and we ended up choosing PostgreSQL to begin. However, upon further digging, I realized that I had overlooked how Android Studio comes with built-in SQLite functionality. For the scope of this project, we decided that it would be more straightforward to switch to SQLite to implement our database.

Patterns used:

- Data Access Object pattern: This pattern is used to separate the low-level actions of the data-accessing structure from high-level operations. At the highest level, we created a DBHandlerInterface class as an interface to declare abstract methods for the database handler. Then, we constructed the format of the entities we wished to store in the database as an AlertModel class in AlertModel.java. We then created a DBHandler class to specify the implementation for each method in the interface so it can handle AlertModel objects, manipulating the database as necessary.
- Factory Pattern and Singleton Pattern: Factory pattern is used to create the desired type of alert based on the given input for the createAlert function. Also, we use the singleton pattern for factories, since we only need one instantiation for it.
- Command Pattern: The Button class has a setOnClickListener function that is given an OnClickListener class with an onClick function inside of it. In this case, the setOnClickListener is the setCommand function, OnClickListener is the Command, and onClick is the execute function within the command. In this case that is what makes a Command Pattern.

Class diagram (so far, will be updated for final code submission)*:*



Future Work:

- For project 7:
    - Will further detail the reporting tab to include alert titles.
    - We will identify places to implement the strategy pattern for processing different alert types.
    - We will make a UI for the verify page.
        - The user will be directed to the “verify page” instead of the “confirmation page” if their description matches a previously filed description.
        - The design of the “verify page” will look a lot like that of the “confirmation page” but instead of having a message with a done button it will contain a short informative message in regards to why they got directed there. At the bottom of the page they will also have two button options which will give them the choice of either reporting a new alert or verifying an old one.
    - Aggregate list of alerts during app startups for alert display list with a tab for viewing all alerts at once (Event feed).
    - Handle location storing while reporting an alert
    - Will add pins to map display, representing where reported alerts have occurred.
    - Implement strategy pattern to handle alert similarity
        - Complete cosine similarity and sentiment analysis in caution and warning classes
        - Add method to grab all strings and alert ids from location data for comparison
        - Include similarity algorithm in AlertModal
        - When a new post is created make sure similarity checker runs similarity score based on strategy algorithm on all alerts of the same type within the location radius
        - If a similarity is found trigger verification page with confirm similar post button as well as usually verification UI content
    - Add JUnit tests to verify that our database is being updated correctly and our verification system is functioning as intended