Scenario 1:

I would store my log entries in a MongoDB database where all of the log messages will be stored as JSON objects.

Creating the MongoDB database allows me to easily manage the different objects and query them based on different fields as well. The querying will all be done with data functions that will be directly called from the routes of the web server.

The web server will be an express server with different routes for different services. One route will be for form submissions that allow users to submit new logs, one route will allow the user to query the log (handlebars will take care of showing the log), and miscellaneous routes will be created for logging in and logging out. This will then allow the data functions to filter by user by using the user's information.

Scenario 2:

I would choose to use an express server in order to use the html-pdf npm package. This will allow an easy way to generate pdfs for users by using an html template and converting the html into a pdf. Using handlebars, the html templates will be able to take in all the different fields and generate unique pdfs for each reimbursement.

As for sending the emails, I would use EmailJS to set up a template and then use the proper fields to generate and send the proper emails. We can use the *user* field to get a user's email and have the email send to them that way.

In the end, I would use html-pdf to generate a pdf and then EmailJS to send the pdf as an attachment in an email back to the user.

Scenario 3: A Twitter Streaming Safety Service

I would use the Twitter standard search API detailed here: https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets. It allows for key word search as well as geolocation tracking by taking in a latitude, longitude, and radius. It is also possible to get tweets made within a specific time period which helps detectives and police to look at tweets taken place near the time of a crime.

Beyond the local precinct, other stations will be able to put in their latitude and longitude and adjust the settings to match their desired search.

In order to ensure that the system is stable, there will be a regulated control system of tweeting out some tweet in the area with a special id that must be picked up by the system or else it will send notice that it was not detected, and something will be fixed.

Due to the nature of this system and how we would like to expand it in the future, I would go with a React server because it allows us to update everything in real time by using component states and to show threat levels.

I would like to use Firebase since we are going with a React server and I would store all of the tweets in Firestore under different collections based on what day the tweet may have triggered the alarm. This would allow easy querying to be made for a history of tweets and would allow for easy querying when sending officers the information of the tweet.

The email sending will leverage EmailJS and the text message service will most likely use a service similar to Twilio or any other mass communication service.

Scenario 4:

I would write an express server that would have many different GET, POST, PUT, PATCH, and DELETE routes for the users and images. The mobile app will use these different routes for accessing and changing the data.

I will have objects that store the image itself, the latitude and longitude of where the user posted, and the user themselves. Using the latitude and longitude, I can display all photos taken from all users within a user's radius.

For long-term storage, everything will be stored in a MongoDB database and for short-term storage, everything will be cached using Redis.