**Scenario 1: Logging**

I would use MongoDB to store log entries, as they are capable of handling large data volumes effectively. To enable log submissions, a RESTful API accepting JSON-formatted entries can be implemented, with authentication and authorization measures put in place to ensure that only authorized users can submit logs. To facilitate log queries, a search API can be created, utilizing the MongoDB database's indexing capabilities to efficiently search for entries based on specified criteria. Users can view their own logs through a web application user interface, built using front-end frameworks by React, that consumes the aforementioned RESTful API. For the web server, I would implement Express.js to handle incoming requests and route them to the appropriate API or user interface.

**Scenario 2:  Expense Reports**

Firstly, to develop an reporting web application, I would design an database schema by using MySQL, which I can store expense data efficiently because their in-built table operation.The id column would be set as the primary key and auto-incrementng, while other columns would store specific information related to each expense. For the web server, a suitable framework like Django can be chosen to facilitate development, which offer the in-built support for PDF generation and sending email task by using built-in EmailMessage class can. To enable programmatic PDF generation, a library like ReportLab can be utilized. This library allows for easy creation of PDF documents, which will be useful for generating expense reports.To render dynamic content in HTML, Django templates can be used. This framework have a simple syntax that allows for easy rendering of dynamic content and displaying expense data to users.

**Scenario 3: A Twitter Streaming Safety Service**

To build this service, I would use the Twitter streaming API, which allows for real-time access to Twitter data. This API would allow me to monitor tweets in a specified geographic area and scan them for specific keywords. To make if more expandable, I would deploy this application by using AWS and using Docker to containerize.

To ensure that the system is constantly stable, I would use a combination of automated testing, continuous integration, and monitoring. This would involve setting up automated tests to ensure that new code changes do not introduce bugs, using a continuous integration system such as Jenkins to automatically build and deploy the application. For the web server technology, I would use Django. This would provide the necessary tools to develop the web-based interfaces for managing triggers, viewing historical data, and accessing the real-time incident report. For the trigger database, I would use a database management system such as MySQL. This would allow me to store trigger combinations in a structured and organized manner, making it easy to query and retrieve information.

For the historical log of tweets, I would use MongoDB, which are designed for storing large amounts of unstructured data, making them ideal for storing the vast amount of tweets that would be generated by this system. To handle the real-time, streaming incident report, I would use a combination of websockets and server-sent events. This would allow me to push new tweets to the web-based interface as soon as they are detected, providing real-time updates to users. To handle storing all the media associated with the tweets, I would use Amazon S3 or Google Cloud Storage. These two are now popular cloud base services to store your data.

Top of Form

Bottom of Form

**Scenario 4: A Mildly Interesting Mobile Application**

I would choose MongoDB too in this application to help me handle geospatial nature of data.This would allow me to efficiently store and query data based on its location. For storing images, I would use cloud storage services such as Amazon S3 for long-term, cheap storage, and a content delivery network (CDN) like Cloudflare or Akamai for short-term, fast retrieval. I would write the API using a Node,js with Express framework, because they are very suitable for building web application’s APIs.They are light-weight and have rich community to help you. Additionally, their framework offers router to direct user to specific APIs. I would use MySQL to store the user and administrative data and a MongoDB for storing the 'interesting events' data. This is because MsSQL are suitable for relational structured and defined data information, but NoSQL database are more suitable for those unstructured and large amount of data. Using the combination of two kind of database would allow this application more flexibility and more efficient querying function.