Scenario 1: Logging

- Log Data Format: JSON

JSON is an appropriate format for storing logs due to that it can easily include a few common fields and any number of customizable fields.

- Store in: MongoDB

Once the log data format is determined, we will find that many databases are suitable for storing JSON, including MongoDB or any other non-relational and document-based database.

- Submit log by: system

As shown in the scenario, users should not actively upload any log data. All log entries should be automatically generated by the system. And these entry templates are pre-defined by the developer. When certain conditions are met such as when an error occurs, they are automatically generated and stored in the database.

- Query log and see:

System administrators can access the logs by querying the database directly, or through a log interface which is designed by developer.

- Web server: express.js

I consider most event-based web servers to be qualified for this position. Among them, express.js is the easiest one I've ever used.

Scenario 2: Expense Reports

- Store in: MySQL

Since all the data has the same structure and it needs to look up the user's email address based on the user provided in the expense form, it is better to use a relational database. MySQL is a common open-source relational database that is supported by most of languages.

- Web server: express.js

I consider most event-based web servers to be qualified for this position. Among them, express.js is the easiest one I've ever used.

- Emails handle: Nodemailer

Nodemailer is a module for Node.js applications to allow easy as cake email sending.

- PDF generation: LaTeX

LaTeX is a document preparation system for creating printable documents. And it also provides node.js driver named node-latex.

- handle templating for web: Handlebars.js

Handlebars is a simple templating language. It uses a template and an input object to generate HTML that keep the view and the code separated. Which also fits well with node.js.

Scenario 3: A Twitter Streaming Safety Service

- Twitter API: Twitter API v2 Academic Research

Twitter API v2 Academic Research can achieve the maximum number of filtering rules (1000 rules) and lengths (1024 characters) that are most compatible with the requirements of this system.

- How to expandable:

First, make the trigger position settings and the keyword list changeable. Second, set up groupings of officials of

different levels and different regions. Third, there should be a user interface that has made it easy for users to CRUD different filtering rules to triggers.

- Stability:

Using a non-blocking programming ensures that the system will not crash even if it encounters an error. Perform stress testing and edge case testing before the system goes live to simulate the stability of the system under extreme case testing.

- Web server:

I would choose to use a distributed web server, divided into two parts. One to monitor tweets, scan and store tweets, and send alert emails. The other is a web server for presenting the data pages to the front-end users. Apache Hadoop is a good technology used for distributed storage and computation

- Databases:

For trigger rules, I would use a relational database that can store fixed types over time, such as MySQL, and for tweets and histories, I would use Elasticsearch for long-term storage to facilitate full-text keyword searches.

- Real time, streaming incident report:

For real-time reporting, I will use the analytics data generated by Elasticsearch in real-time and present it through some web visualization framework such as eChart.js.

- Media Storage: ImageMagick

ImageMagick is a very powerful image manipulation utility that essentially allows me to perform any sort of conversion imaginable on an image.

Scenario 4: A Mildly Interesting Mobile Application

- Data handle: JSON

I will store this geolocation information into different JSON entries and show a certain logical inclusion or parallelism.

- Images storages:

I will use ImageMagick to compress and re-encode the images, store the source data for long-term storage in a cloud storage CDN such as Amazon S3, and store the links to the images to be retrieved quickly for a short time in a Redis database li

- API: express.js

I consider most event-based web servers to be qualified for this position. Among them, express.js is the easiest one I've ever used.

- Databases:

I will use MongoDB to store user data and various data uploaded by users, and Redis to store data that needs to be retrieved quickly for a short period of time.