Scenario 1: Logging

- Log Data Type: JSON

Because logs are asked to have some common fields and any number of customizable fields, using JSON to store logs is suitable for the flexible requirement.

- Store:

Once the log data type is determined, we will see that many databases are suitable for storing JSON data, e.g., MongoDB or any other non-relationship database.

- Submit:

From my perspective, users should not be the person who submits the log, and it should be generated by the backend system developers' design.

Developers can use NoSQL to submit logs or log entries into databases.

- Query:

As we store logs in non-relationship databases, the way of querying will be NoSQL.

- Visualization:

Due to log data being in the database, we need to retrieve the data and visualize them. Users and developers may view logs on different devices, so viewing logs in browsers is proper.

For browsers, we can build a web server using node-express, Apache tomcat, or other advanced webserver determined by languages.

Many frameworks show colorful and explicit charts datagram for visualization, such as echart.js.

Scenario 2: Expense Reports

- Data Store: Relationship Database

Due to all data having the same structure, the relationship database is more suitable.

- PDF generation:

Due to different languages, there are many packages or libraries. For Java, there is Apache PDFBox; for JS, there are PDFKit, Pdfmake.

- Email:

Like PDF generation, there are many packages or libraries. There is a default mail tool for Java; for JS, there is a nodemailer.

- Templating:

It's simple to use the concept of object-oriented to handle all the templating. We can build a class for each template.

Scenario 3: A Twitter Streaming Safety Service

- Twitter API:

Enterprise APIs, cause the number of area Twitter will be beyond 100k.

- How to expandable:

set location of triggers is changeable.

create a dashboard that administrator can change the email receivers;

The rules of alert words should be editable, and the developer also can set a lightweight language for policies to add new rules.

- How to make sure stable:

Test the service regularly, e.g., create some test cases to check the accuracy.

- Web server:

Cause the webserver should access daily twits and police can monitor the output in real-time, I will divide the webserver into two parts: one accepts twitts, stores data in the database, and triggers emails, one is for police to view the database. So, the webserver will be distributed web server.

- Database

Cause the twitter data is JSON data, we need to use a non-relationship database. The database will have two collections, one for storing the history twitter identity (not complete twit), another is to store the alert and the result of the alert.

Scenario 4: A Mildly Interesting Mobile Application

- Database: non-relationship database

Because the picture has common fields(user) and any number of customizable fields(interesting events), the database should be non-relationship.

Take MongoDB as an example, the database should have two collections: users and pictures.

The pictures collection will store pictures' paths in the file system, their geographical location (Longitude, latitude, direction, distance), and their users. Then for searching, the pictures will store some descriptions, e.g., topics, events, time.

- File System:

The application can use Redis for fast retrieval or some CDN instances for the short term. In the long term, the images can be stored in other instances with big storage and little calculation.

From my perspective, the most challenging part of the mobile application is to create the searching algorithm and how to get the precise tags of the pictures.

A well-trained Deep learning model is a good way to get tags automatically, but the previous cost is high. And if we give the task to users, it will make the application annoying.