Scenario 1: Logging

In this scenario, you are tasked with creating a logging server for any number of other arbitrary pieces of technologies. Your logs should have some common fields, but support any number of customizable fields for an individual log entry. You should be able to effectively query them based on any of these fields.

How would you store your log entries? How would you allow users to submit log entries? How would you allow them to query log entries? How would you allow them to see their log entries? What would be your web server?

Since the Requirements suggests that that logs along with having the common fields, should also have customizable fields, I would think of going with a No SQL database like Mongo DB or Dynamo DB which allows schema-less structure. This will allow us to add a log entry with varied components. Secondly, to effectively query them based on any of these fields we need to have a unique id or Object ID to individually identify the requested resource, we can use Object ID in Mongo DB as our first filter and similarly we can also have a timestamp in case we have sub-objects added in our main log object to further identify individual components.

Each individual log entry will be stored as an object with the customizable areas being stored as a sub-object with a timestamp that individually identifies them inside the main object. To submit the log entries, we can create a user interface with Add-Delete components and allow user to select the data-type of component to create a single log entry. These will give them flexibility to select create a customizable log entry. In order to query log entry, we can use findOne() of MongoDB to identify the fields in our database and user can see their query results and identify them on the basis of unique timestamp or combination of that with other fields on browser. I would prefer Node JS and Express web server for the same as Node JS and Express fits perfectly and have a huge community for reference.

Scenario 2: Expense Reports

In this scenario, you are tasked with making an expense reporting web application.

Users should be able to submit expenses, which are always of the same data structure: id, user, isReimbursed, reimbursedBy, submittedOn, paidOn, and amount.

When an expense is reimbursed you will generate a PDF and email it to the user who submitted the expense.

How would you store your expenses? What web server would you choose, and why? How would you handle the emails? How would you handle the PDF generation? How are you going to handle all the templating for the web application?

For this particular scenario, I would separate my concern into front end and backend and I would like to use some of the AWS web Services for the same. Since the structure of the expense is always going to be the same, we can use SQL or No SQL database. My front end may be built on Node JS or React that will take care of submitting the expense to an endpoint where all other functionality will be performed. I can use Handlebars to template my web application as it is easy. Since I am going to use some of the AWS services I would use Dynamo DB, NoSQL database to store my expense where Id will be my partition Key and user will be my sort key, I can also create an Index and create a composite key will give me an amazing querying capacity.

Now for the next part, my backend system will perform all the necessary tasks. I will use Lambda functions and Serverless system to generate a pdf and email the expense report. Serverless computing gives us the advantage of scaling in and out on demand with high availability. For this my lambda function will be triggered upon the reimbursement request made to DynamoDB, my function will then execute and uses a sample expense document to populate the submitted fields. For example, I can send my id, user, amount etc. to the document template as variables and using JODReport script I can make a dynamic document which will then be converted to PDF using some third-party API like Convert API. After the execution of this I can use Amazon's Simple Notification Service to send an email to the user who submitted the expense, this is actually the most common use case of this service.

Scenario 3: A Twitter Streaming Safety Service

In this scenario, you are tasked with creating a service for your local Police Department that keeps track of Tweets within your area and scans for keywords to trigger an investigation.

This application comes with several parts:

- *An online website to CRUD combinations of keywords to add to your trigger. For example, it would alert when a tweet contains the words (fight **or** drugs) AND (Small-town USA HS or SMUHS).*
- *An email alerting system to alert different officers depending on the contents of the Tweet, who tweeted it, etc.*
- *A text alert system to inform officers for critical triggers (triggers that meet a combination that is marked as extremely important to note).*
- *A historical database to view possible incidents (tweets that triggered an alert) and to mark its investigation status.*
- *A historical log of all tweets to retroactively search through.*
- *A streaming, online incident report. This would allow you to see tweets as they are parsed and see their threat level. This updates in real time.*
- *A long-term storage of all the media used by any tweets in your area (pictures, snapshots of the URL, etc.).*

Which Twitter API do you use? How would you build this so its expandable to beyond your local precinct? What would you do to make sure that this system is constantly stable? What would be your web server technology? What databases would you use for triggers? For the historical log of tweets? How would you handle the real time, streaming incident report? How would you handle storing all the media that you have to store as well? What web server technology would you use?

Twitter search API allows us to have access to either the last 30 days of Tweets or access to Tweets from as early as 2006. Geocode parameter in API returns the Tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by " latitude, longitude, radius ", where radius units must be specified as either " mi " (miles) or " km " (kilometers). we can use this geocode parameter to search near geocodes directly. This application will be completely expandable as because the regions can be changed by changing the location coordinates of the area specific to the department of police and radius parameter which allow us to cover a larger area in miles as discussed. We can have our trigger enable on every insert operation in Dynamo DB which will check for the "hot keywords" like *(fight **or** drugs) AND (SmallTown USA HS or SMUHS)* in this case. As soon as we have found these words on insert, we can then use Simple notification

service to push an email to alert different officers depending on the contents of the Tweet, who tweeted it, etc. Further we can also use the same service to generate a text alert to inform officers for critical triggers.

For long term storage of all the media used by any tweets we might want to store them in a non-frequent storage device which is also cost effective, In fact we just want to store them and need them only when we require so in that case, we will use S3 and have a lifecycle management enabled on it which will put the old tweets from frequently accessed storage like Dynamo Db or S3 in our case to IA storage like Glacier or Dataware house. All these can be done on a serverless system which allows massive scaling within seconds without caring about the underlying infrastructures.

Scenario 4: A Mildly Interesting Mobile Application

In this scenario, you are tasked with creating the web server side for a mobile application where people take pictures of mildly interesting things and upload them. The mobile application allows users to see mildly interesting pictures in their geographical location.

Users must have an account to use this service. Your backend will effectively amount to an API and a storage solution for CRUD users, CRUD 'interesting events', as well as an administrative dashboard for managing content.

How would you handle the geospatial nature of your data? How would you store images, both for long term, cheap storage and for short term, fast retrieval? What would you write your API in? What would be your database?

We can get the location coordinates where the picture was clicked from the EXIF data -- Exchangeable Image File, a format that is a standard for storing interchange information in digital photography image files using JPEG compression. Using the location coordinates, we can have map those coordinates to a geo location and show all entries satisfying the boundary conditions. AWS and other cloud vendors provide caching server like Cloudfront etc. that will process the request faster in all geographic locations. I would store the images for the long term on Amazon S3 or Google Drive and enable the lifecycle management rules on them since they are much reliable, secure and have negligible downtime. For the short-term solution, I would use Redis to cache the images. I would write my API in NodeJS and use DynamoDB as my database for the technology stack.