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?

Solution: For the purpose of logging, I would prefer to use a NoSQL database like MongoDB. Being a NoSQL database, it allows expanding the table horizontally with ease. This helps in adding more level of customizable fields for an individual log entry. Each individual log entry will be stored as an object with the customizable areas being stored as a sub-object inside the main object. The user will be submitting the log entries from an HTML page from which they will be passed in by the routing class into the Nodejs main class which will then use the insertOne() function in order to insert the data into MongoDB. Queries can be implemented using findOne() function in MongoDB and then submit the response to the page with the result using Express. I would use Node along with Express as my webserver since it is robust and I am pretty comfortable with it. Users will be displayed with a radio button in a radio group where they will be selecting the type of log they are going to create(description only or description & time).

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?

Solution: I would prefer to use MongoDB in order to store the expense table, in case there is a new requirement to add a new parameter to the table it will be easier to configure the same. This will future proof my database. I could have used MySQL or Postgres but then it won't be easier to expand the table horizontally in case there is a need in the future. I would use Node along with Express as my webserver as it is robust and there is good no of NPM packages to add functionalities with ease. Emails can be handled using the NPM package called nodemailer. For PDF generation I would have to use latex as it provides professional looking PDF and con be done absolutely by programming. I would use HandleBars to template my web application. I have used it many times and it had never disappointed me in any aspect. HandleBars is pretty easy to understand and code as well.

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 (SmallTown 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?

Solution: I would use the Twitter search API with the geocode parameter in order to filter out the tweets of a particular region. 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). This application will be completely expandable in case the police f other regions want to use the same application then we just need to change the location coordinates of the area specific to the department of police and it can be used. I would be making the system stable by constantly updating the application with new patches whenever there is a new release form any technology provider so that the code does not go obsolete and contain previously known vulnerabilities. My server would be based on Node and Express since it is robust and there is perfect compatibility with Redis and MongoDB. I would use Redis since it is fast and reduces access time compared to querying from MongoDB. Of course, MongoDB will be there behind Redis storing al the data. Redis will be used for triggers. For storing the historical log of tweets, MongoDB will be used in the application designed by me. In order to handle the realtime streaming incident report, I will be using WebSocket client subscribed to the required events qualifying the required parameters set by the authorities corresponding to new tweets. I would prefer to store all the media in Amazon S3 since it is secure, reliable and scalable.

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

Solution: We can get the location coordinates where the picture was clicked from the EXIF data of the picture. Using the location coordinates, we can create a filter of say 30 miles and show all entries satisfying the boundary conditions. I would deploy plenty of caching servers everywhere so that at any particular location there will be a nearby caching server 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 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 MongoDB as my database since it works flawlessly with NodeJs.