Scenario 1: Logging

Prompt: 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?

I would store the log entries in a MongoDB database. This database would be better than SQL because of the customizable fields which require flexible attributes. Since logging records something, the logs can be submitted through create operations such as insertOne and read operations such as findOne. I would save these logs in a .log format. These .log files alongside the log entries. The log entries would have a link to this file. The web server would be express and node since these technologies work well together alongside mongoDB.

Scenario 2: Expense Reports

Prompt: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?

Since these expenses are of the same data structure, I would store them in a SQL database. The SQL database operations must be atomic to ensure that each expense is accounted for once. I would choose a LAMP stack because MySQL is a SQL database and the rest of the technologies work well with it. To send the email once isReimbursed is true, I would create a php script using the mail function that contains the attached pdf. The pdf would be generated from the row in the database. The library FPDF is a PHP class that generates PDF files with PHP. The templating for the web app can be done in PHP and I would use Apache for the web application.

Scenario 3: A Twitter Streaming Safety Service

Prompt: 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?

Using the MERN stack taught in class, I would make axios calls to the get search tweet api: https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets. Two important parameters are the query and the geocode. The trigger keywords would be saved in a redis database for quick access and the geocode would allow me to focus on a specific region. To make sure the system is constantly stable, I would decide a reasonable count (another parameter of the tweet search api) for each request. Since I am allowed 450 requests per 15 minutes, I would adjust it according to the web server capabilities. I would use a mongoDB database for the historical log of the tweets. To make a real time incident report, I would use node.JS to make new requests to the api. For the real time, streaming incident report, I can use PowerTrack: https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data for real time. Streaming incidents since this receives tweets when new data arrives. Using mongoDB will allow me to store all the media.

Scenario 4: A Mildly Interesting Mobile Application

Prompt: 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?

I would use Firebase/Redis for the backend for this mobile application with express/ node/react stack. For user CRUD operations, Firebase Auth would be used. For CRUD interesting events, both Firebase and Redis would be used. With Firebase, I would have a dashboard for managing the content of user accounts. MongoDB is better for large databases but I choose not to use it since the Cloud Firebase can be used for long term and large storage.

To use this application, users must have the geospatial tracking on. The "interesting events" data structure would be the id, geospatial location, picture file, captionMessage, userid. Every interesting event would have a location attached. I would use ImageMagick to compress the images and make a save for the interesting event in a redis database. I would store both the interesting events in firebase and redis.