Scenario 1: Logging

- Let's start with the Database, as the requirements state the logs can contain any number of customizable fields as well, I believe going with a NoSQL DB is preferable as it gives us the flexibility to add more parameters to a document and not follow a strict table attribute SQL DB format. As I personally have more experience with MongoDB and it does provide all the features required for storing the logs in an optimal way, I will choose that as the Database.
- Coming to writing the server, I will prefer writing an Express Server in Node.js as it can easily serve the purpose and has robust drivers available to connect with our Mongo Database. We can either create a middleware which automatically logs the data to the server as the server runs or we can render/send web pages for user to manually enter the data. Also, Express App can be scaled in the future as well if we plan to add more functionality.
- Coming to the part of how the user will submit logs and query them, I would prefer implementing both the options of a webpage and sending RAW JSON data. The frontend has to be fairly basic to just get the user input when logging and displaying formatted logs when querying the DB. This can be achieved with plain HTML as well, and I would prefer it to keep it simple. I would also make routes for APIs to send RAW JSON data, the user can work with this data as they want.
- As the requirement states the user should be able to query based on specific fields as well, I will add those filters to front end and for the API route.

Scenario 2: Expense Reports

Implementation:

- As the structure of the Database is defined with a fixed set of attributes, and all the attributes can be uniquely identified by the ‘id’, I think a SQL Database will be best suited in this scenario. We can have one single table with all the listed attributes.
- This case requires features such as webpage templating (to show different data in the same format), a module to convert data from the DB in a formatted PDF and then another module to email that PDF to the user – I believe all this can be implemented in an Express App with modules from NPM.
- I can use the MS SQL as DB and for drivers I can use ‘node-mysql’ to connect our app to the DB. I will use handlebars as view engine to template the webpages.
- To convert data in a formatted PDF, I can use the NPM module ‘pdf’. On the Express App we can fetch data from the DB and format and create the PDF using the module.
- Finally, to send the email I will use the NPM module ‘nodemailer’, which can use email services like Gmail to send emails to the users.

Scenario 3: A Twitter Streaming Safety Service

Implementation:

- I will have my client running in React to stream the online incident report. I will connect this to Firebase DB (Real-Time DB) which updates the website as and when new data is written.
- I will write the server in Node.js
- This server will server multiple clients, first I will configure it with Firebase DB to write to it as and when Tweets are parsed.
- I will configure the Twitter Enterprise Power Tracker API, which is a real-time API that gives us option to parse tweets based on keywords as well.
- I will get the tweets from this API, which will also provide with the geographical data and details of user who posted.
- I will also log this data to my Mongo Database to store the tweets triggered by the keyword. I will assign additional attributes to the document which will also have the investigation details.
- If the Tweet has any media linked to it, I will create a copy of that media and upload it to a cloud storage hosting site, and then save the URL with the Mongo Document.
- Saving the Tweet and a copy of media ensures we have the data even if the Tweet is deleted from Twitter.
- I will also use NPM module like ‘nodemailer’ to alert different officers depending on the keywords identified in the Tweet.
- I will setup SMS APIs like ‘Twilio’ to alert the officers on SMS as well.
- I will host my entire app on platform like AWS or Azure with distributed servers and multiple nodes which provides stability and load distribution for out app. We won’t have to worry about the hosting server going down as it will be backed up by running on multiple servers.
- After iterations of builds when my app is much stable and streamlined, I can expand this to other precinct by adding their details from the admin panel as well. As the app uses MongoDB, it will be easy to scale the application.

Scenario 4: A Mildly Interesting Mobile Application

Implementation:

- I will create the app with React-Native, this way it will be easier to design the front-end and components can be reused.
- For the backend, I'll use Express & MongoDB. Express App can be used to create API Endpoints for our React App. We can create different endpoints for different operation in CRUD.
- To authenticate the user, we can use firebase authentication which will enable social login for our app.
- We can create a different React App for the Admin Dashboard, and configure API Endpoints for it in our React App. Admin can be signed up with local email and password which will be hashed before string in out DB.
- I will use Google Maps API to add geographical information to the photo data, we can use also use this data in other ways to enhance out app in the future
- I will store photos in a cloud storage and map the links in my photo data document in MongoDB. I can create thumbnails of the photos using different module and store them separately for faster access.
- My MongoDB Document will contain all the information about the user, and the photos. There will be different collection for these two information.