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?

For logging common and customizable fields individual log entry I would use the console.log or logger function of **Node.js**. As an asynchronous event-driven JavaScript runtime Node.js can log - error, warning, information, verbose and can be used for debugging. The reason I would use Node.js for logging because when we use Node server to run a website, it becomes easier to use inbuilt Node functions for logging.

I would use **MongoDB** to store log entires and its function for data manipulation. MongoDB has some really efficient and uncomplicated data manipulation functions which can be used to Add, Remove, Update and Query various logs which would be stored in the Database.

I would allow the user to see their log entires in the terminal or server side log screen which would be easier to locate.

Since the logging is using Node.js and **MongoDb** is highly compatible with Javascript and the Node package, I would use server side Node as a web server along with Express although Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

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?

I would use **MongoDB** to store each entry of expenses based on the structure. Since MongoDB is a cross-platform document-oriented database program and it is classified as a NoSQL database program it is easier to store and retrieve entries from MongoDB.

I would choose **Server side Node.js** as my web server because of compatibility with different javascript packages and its readily available node package manager. The user would have to create an account and session would be created using **express-session** every time he logs in.

Javascript has a very frequently used Package while is compatible with Node.js and can be installed using npm called - **Smtp.js** which uses SMTP server to send an email to the user. It is free to use and provides encryption and supports sending file attachments like pdf.

PDFKit is JavaScript PDF generation library for Node and the browser. It supports Vector Graphics, Text, Font & Image embedding, Annotations, Outlines and Encryption. It is an easy to use JS library which can be used to make custom documents. For this scenario the PDF would include - the details about the user, the transaction ID and other important transaction details.

The templating of the Web Application can be done using **Bootstrap** or **Semantic UI** which again provides easy to use functions and have a lot of ready-to-use templates for the specific scenario.

(Scenario 3 on next page)

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?

The **Standard search API from twitter** can be used to fetch tweets based on keyword. This API is JSON compatible and includes parameters such as query parameters, geo-location radius & count etc. Using these parameters we can expand it beyond the local precinct with a wider radius and languages other than English to reach out to different communities. These tweets can be displayed in real-time every time they are parsed through the API as JSON objects.

To make sure the system is constantly stable, we should parse a certain number of tweets at a time and have a small timeout period after which the system resets the list and starts parsing agin. Adding certain milliseconds of delay can also help in keeping the system stable.

MongoDB can be used for the triggers because it is a easy to use, highly responsive NoSQL database which can handle high volume transactions. MongoDB can further be used to store all media files and to maintain both a historical database with all values and a flag for the threat level, and investigation status along with a separate database to log previous tweets and actions taken to search through.

An email to different officers depending on the content can sent using the **Smtp.js** which uses SMTP server to send an email to the user. It is free to use and provides encryption and supports sending file attachments like pdf. Text Messages to inform specific officers based on tweets parsed can be send using the **Twilio** package which can be used in Node.

Web Server which will be used would be **Node** because it is the most reliable and compatible.

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?

To handle geospatial nature of the data, the latitude and longitude of the user while they were performing the action will be stored and can be plotted back using **Google Maps Web AP**I where the details can be displayed at the pin point. Google Maps API offers a free service where you can plot locations on top the actual map layout.

To store images for both long term, cheap storage and for short term, fast retrieval, I would use **MongoDB** because it follows NoSQL structure which makes data storing and retrieving faster for both short-term and long-term use. MongoDB also supports storing values or multiple categories like text, document, image file etc.

The API would be written in client-side **Node** which would support all the necessary packages. **Express** and **Express-session** to maintain a session for a particular user when they login to the web application.