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- In this particular scenario I would use a NoSQL database such as MongoDB because this will create a flexible environment necessary for adding extra levels including several customizable fields for an individual log entry. Individual log entries can be stored as an object that consists of customizable fields inside as individual fields and as sub-object inside the log entry object, for example log entry object can have username, name etc as individual object and have the time as an individual field. The user will be submitting the log entry through the rendered HTML page, from which it is received through the route functions and the information is verified with necessary conditions and will be sent to the data functions, where the interaction between MongoDB and Nodejs is executed and the log entry will be stored in the database using insertOne() function. There will be an input field for the date on the HTML page where the user can enter the date and log entries can be queried by using findOne() function from MongoDB according to the given date and then we can render the response page with results extracted from the database. The user log entries can be fetched from the database using find() function and can be displayed by rendering a HTML page through a button, where the user can click it if he wants to view the log entries after login functionality has been executed. The web server that I would use will be server side Node along with Express as it is compatible with Node.js and MongoDB and because of the node package manager I can use the Express- session as well, where the session can be created when the user is logged in and can be destroyed when the user has logged out, it will prevent the direct access of the user account after the user has logged out.

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- For this particular data structure storage, I would choose MongoDB to store the necessary data because it will be easier to expand and add further fields in the future whenever there is a new requirement because for now this might be structured data but when new fields are to be added to the reports then the MongoDB feature to handle the unstructured data will be very useful for the application. As a web server for this application I would use server side Node.js because it consists of node package manager which has several packages such as express-session that would be useful for creating and destroying a session whenever the user log in and log out to generate the reports and this server would be compatible with the other packages necessary for the application such as Nodemailer, which I would use to handle the emails since SMTP is the main transport in the Nodemailer and SMTP based sending is supported by several mail delivery provider and if the provider is changed, we can just make necessary changes in the configuration options to replace one provider with another one. The node package manager is also useful for installing a package pdfkit, which is a pdf generation library for node and the browser. It is simple to use as it requires few function calls and it has several features such as Encryption, Higher level APIs for creating tables and laying out content etc. I would use handlebars and bootstrap along with css to handle the templating of the web application, because it would be compatible with the other components that I'm using in the application such as Node.js and MongoDB,and bootstrap also consists of easy to use functions and have several ready-to-use templates for this specific scenario. The login page would be available for the user and after logging in, user should be able to submit the expenses and this data would be stored in the MongoDB as the fields given in the scenario and after the expenses are reimbursed the fields would get updated and they would be receiving an email regarding the reimbursement in a pdf format.

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- The twitter search API can be used in this scenario to fetch the tweets by using the keywords as search parameter in the query and the location is of two types, one is tweets with a specific latitude and longitude "Point" coordinate, come from GPS enabled devices, and represent the exact GPS location of the tweet and another one is tweets with a Twitter “Place” which consists a polygon that is of 4 longitude-latitude coordinates defining the general area from which the user is posting the tweet. Additionally, the Place will have a display name, type (e.g. city, neighborhood), and country code corresponding to the country where the Place is located, among other fields. The geocode parameter can be used for searching the tweets in the location. The geocode parameter should have latitude, longitude, and a radius (mi or km), using this you can create "circles" that cover all your areas of interest. Twitter returns tweets within the given circle by using their own internal methods to geolocate tweets. Twitter uses a combination of device/gps coordinates, user provided profile location, and network/ip address location to determine tweets that fit within the geocode search parameter. Using this ability to configure the location latitude, longitude, and a radius, the circle can be expanded beyond the local precinct. The system should be constantly stable, so I would be updating the application using a new patch if there is any new version from the technology that is being used in the application with standard versions, and also continuous optimization, integration and testing should be performed. The server technology I would use would be Node and Express since they have really good compatibility with the overall technologies used in the application and since it has a node package manager, I can easily use numerous packages necessary for the application such as Redis and MongoDB. I would use redis Pub/Sub feature for triggers where the subscriber to the keywords can be authorities and when the publishers publish about something using those words, the authorities receive messages through Redis, and also it is fast accessible when compared to fetching from the MongoDB using queries. For the main database of the application I would use MongoDB for the historical log of tweets as it is compatible with the whole application components such as Node, express server and redis. MongoDB will also be useful for the horizontal expansion of the data and different fields for the tweet log can be stored and fetched by using queries. The real time streaming incident report can be handled by using WebSocket client subscribed to the required events which would be matched with the parameters set by the authorities corresponding to latest tweets. For this particular application I would use Amazon S3 for storing all the media because the cost of hardware would be lesser, capacity planning is not necessary because storage is technically infinite and we have to pay only for what you actually store and a quick configuration can put CloudFront on top of the S3 bucket and get better last-mile performance utilizing Amazon's CDN. The web server technology I would use would be Node and Express since they have really good compatibility with the overall technologies used in the application and since it has a node package manager, I can easily use numerous packages necessary for the application such as Redis and MongoDB.

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- The geospatial nature of the data can be handled by using Geolocation API as it is useful to calculate the location of the user using the IP address and network available, which gives a response as location of the user consisting of two fields that is latitude and longitude in degrees, it also gives another field in response that is accuracy, this represents the radius of a circle around the given location in meters. For short term, fast retrieval I would use a cache server such as redis because if we use Redis, the hash data type can be used. It gives access to each field in the hash individually thus CRUD (create, read, update, delete) operation can be executed on each one of them. Redis supports the data storing in two different ways, one is RDB snapshot and another one is AOF log (Append Only File log), If the dataset stored in Redis is large then the RDB file will take some time to be created, which has an impact on the response time, but it will be faster to load on boot up compared to the AOF log. I would prefer using Amazon S3 for long term and cheap storage because the cost of hardware would be lesser, capacity planning is not necessary because storage is technically infinite and we have to pay only for what you actually store and a quick configuration can put CloudFront on top of the S3 bucket and get better last-mile performance utilizing Amazon's CDN.

I would write the API in Node.js which would support all the necessary packages such as Express and Express-session to maintain a session for a particular user during the login and usage, it also helps in destroying the session. It is also compatible with Redis and other technologies in the application. The database that I would be using would be MongoDB as it would be flexible and compatible with Node.js and the data storage can be increased horizontally if necessary and it is easier to store and retrieve entries from MongoDB.