<u>Scenario 1: Logging</u>

The task assigned in this scenario is focused on creating a logging server where these logs are supposed to have some common fields of the user who is trying to access the server. While the server should be created in such a way that it supports data field customization each and every individual's log entry.

To complete the task log entries should be stored first in a database. Considering the scenario our database should support customization of fields efficiently. Supporting our situation, we need a non-relational database like MongoDB. For log entries a NoSQL database will work very efficiently because of its scalability. Main reason to choose MongoDB is because of its document database stored in the form of JSON style in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another. Advantages of MongoDB are replication, high availability, rich queries and fast in-place updates.

Obtaining log entries from the user will be the next task needed to store the data. For this, it is required to have a webpage where user can provide the log entries. Which will require a user-friendly UI which can be a HTML form. To get data from the user, I will require the user to submit entries via a post request. This post will request to an API endpoint something like user/logs. This will help us getting log entries via HTML form using API request.

To allow user to query log entries will be using get request again using HTML forms where /user/search can be used where every log entry has specific id. This can be completed by using MongoDB queries. We can install MongoDB as a dependency in my NodeJS. Where MongoDB can be used to extract data from the collection using a specific query or function when an API request is made.

For user to view log entries a straight-forward front-end application can be done where user can easily view the data. For this again an API request can be made such as get request to get/user/logs which returns a JSON from MongoDB collection and can be processed into basic front-end UI for better visualization. ReactJS or Handlebars can be an option to show the log entries.

Looking at the scenario requested in the question I would prefer to user NodeJS web server which will work very efficiently and also will support all the requirements we need while creating log server.

<u>Scenario 2: Expense Reports</u>

In this scenario I assigned the task to make an expense reporting web application where user can submit expenses using a provided data structure as mentioned in the question. This web application should be built in such a way that it does not collapses when user fails to provide correct inputs.

Important thing will be to store data where customization is possible and it can efficiently work through the data structure provided to me. Here, a relational database like SQL can work very efficiently and would make it easier for data viewing and storing. While MongoDB can also be used considering its Document oriented database where such data structure can be stored in JSON format and can be accessed using MongoDB queries.

Considering the situation, I would use Django web server considering its ability to create the tables themselves and define the queries or procedures which sometimes translates to the hefty amount of SQL that is prone to be complex and hard to track. Another main reason is its easy connection with SQL.

I would use Django-celery-beat in my web application. This will trigger every time an email appears and would help in generating pdf whenever user requests or user changes expense or expense request is triggered.

Handling PDF generation would be easy using python packages such as PyPDF2, xhtml2pdf, pdfrw, PyFPDF. These packages would help generating pdf from the provided template.

Django being a web application framework has very efficient template handling system. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. It also helps to insert logics in the template in very easy method. HTML can be loaded using Django template loader function and also use the same function to store the template.

Scenario 3: A Twitter Streaming Safety Service

In this scenario the assigned task is to build an application for local Police department that keeps track of tweets within a designated area and scans for keywords to trigger an investigation. The application includes many requirements such as a CRUD combination of keywords, that would trigger an investigation whenever the CRUD keywords are found in any tweets. This is to be notified to the officers where it sends user information and the tweet that was posted. In the CRUD keywords there are extremely strong keywords which are subject to be investigated on high importance should be sent in high priority to the officers. To provide a database which can show all the required investigation data and twitter log when required which also supports long term data storage. Parsing the tweets and streaming online incident will also be part of this application.

I will use Filtered stream API which can scan through real-time tweets. This API also supports searching using CRUD combination of keywords. It can track if any tweet includes such words that are required to trigger investigation.

To have more efficiency so that its expandable to other precincts I will build using MongoDB which will make my application more scalable with Realtime data with its document oriented non-relational database. Using MongoDB queries, it will make easier to read and store real-time data. One package called geolocation can also be used to expand beyond my precinct.

Code can collapse due to bad error-handling and class duplication. This are the main reasons when code can collapse due to bad inputs or some sort of different data to encounter. To avoid these issues, I will use continues integration and tests to ensure that the code works fine. I would make sure that all errors are perfectly handled and there are no duplicate classes present in the code.

NodeJS and Express JS would be my first option as my web server having the readiness and easy configuration with MongoDB would be useful for having a stable system. Using the security options provided by NodeJS and also having routes controlled using URL would be great option to handle such large application.

I would use MongoDB which is quite efficient with real-time data and is simple to read and can be dynamic. Using MongoDB queries and its feature to have functions behind them will be very helpful to trigger keywords.

The historical log of tweets can be either stored in MongoDB or Redis. Both of them provides very efficient base to store non structured data. While the requirement of the application is to store and read such unstructured tweets and comments provided by the user which can be supported by MongoDB. While Redis having distributed and in-memory key value will also help in storing such tweets.

I will use Polling and Web Sockets to handle real-time streaming incident. This would make it easier to prompt whenever an incident has to reported for investigation.

I would use something like Amazon web services, Amazon S3 to store media. These platforms are very efficient and cheap to handle storing such data types. Considering twitter high volume of media storage will be required which is only possible by using services like this.

I would use NodeJS and express JS which would be useful because of its easy configuration and security provided. Middleware can be used for processing requests and response of the application. It provides many dependencies that can be used to build the application. Using the efficiency of defining routes for particular URL can be used for such a huge application.

Scenario 4: A Mildly Interesting Mobile Application

The task assigned in this scenario is to build a web server side for a application where people can upload mildly interesting things and upload them. This application allows user to view these pictures in their geographical location. The backend should store user accounts and should have solution for curd users, with dashboard management.

To handle the geospatial nature of the data I will use MongoDB and its features. One which is GeoJSON which will help in storing such kind of data.

As it requires long term and cheap storage for fast retrieval and efficient data storage my best option will be cloud services. I will use AWS, Amazon S3 or Google cloud to store media having its easy configuration and cheap way to store such data.

I would write my API on Express JS which include very easy configuration with MongoDB. This would also include using middleware, react JS, and other features that are provided by NodeJS. We need a mobile application which can be designed using JavaScript framework which makes the development very easy and fast. It can also ensure that we have a stable server.

Seeing the scenario and requirements provided in the application development I would use MongoDB considering it high efficiency with unstructured data. NoSQL data can also increase scalability in such application which will ensure that the application can be updated and work for long periods. Also, it is very easy to configure MongoDB with express JS.