Name: Priya Gupta
Assignment: Assignment 05

Scenario 1: Logging

How would you store your log entries?

Because of the volume of data, the logging can generate, I will prefer to store it in any data lake, like Amazon s3, HDFS etc. So here, I'll go with Amazon s3 cloud server (because of its cost effectiveness and accessibility), and I'll use json file format because:

1. The biggest benefit of **logging in JSON** is that it's a structured **data format**.
2. This makes it possible to analyze **logs** using any of the open source big data tools and NoSQL databases which will provide an opportunity to query the log using SQL like queries

How would you allow users to submit log entries?

Will use Winston API for collecting all users submitted log entries.

Because it offers:

1. JSON formatting, coloring log output and the ability to fiddle with formats before they are posted off to your end log locations.
2. Winston adds another interesting layer, which is log levels. Log levels are metadata for logs. Log levels tell you the severity, ranging from as severe as a system outage to as small as a deprecated function. The reason for having log levels in the first place is so you can quickly see which logs need attention. Log levels also allow you to configure environments to log the correct amount of data, with amounts of detail.

How would you allow them to query log entries?

For querying log entries, Winston supports querying of logs with different database. Here I would prefer to use cloud mongoDB because it supports huge volume data and traffic. And it is a document data model which is a powerful way to store and retrieve data that allows developers to move fast as compared to relational database.

How would you allow them to see their log entries?

There should be an option given to download the generated log files in PDF format, which will allow user to see the Winston log files.

What would be your web server?

I will go with Express.js as a web server. It offers:

1. **Scale application quickly**: The first benefit of using Express.JS for backend development is that you would be able to scale your application quickly.
2. **Can use same language to code Frontend**: Another benefit of using Express.JS is that you would be able to do the code of both frontend and backend with the help of using JavaScript.
3. **Supports Caching**: Express.js supports the caching feature, and the advantage of the catch is that you would not have to re-execute the codes again and again. Moreover, it will help web pages to load faster than ever.

Scenario 2: Expense Reports.

How would you store your expenses?

I would allow user to upload an excel sheet in the given format of **id, user, isReimbursed, reimbursedBy, submittedOn, paidOn, and amount.**
Then will import csv data in mongoDB using fast Fast-CSV module. Reasons to choose mongoDB:

- It provides high performance, high availability, and automatic scaling.
- It is extremely simple to install and implement.

What web server would you choose, and why?

I will go with Node.js server. Because it allows to write server-side applications in JavaScript. It's also lightweight, efficient, and its ability to use JavaScript on both frontend and backend opens new avenues for development

How would you handle the emails?

After some research to handle emails in node module I will preferably go with **nodeemailer**.
Some of the reasons to choose nodeemailer are:

- It is a single module with zero dependencies.
- It’s Heavy focus on security.
- Unicode support to use any characters, including emoji.
- Use HTML content, as well as plain text alternative

- Add Attachments to messages
- Embedded image attachments for HTML content – your design does not get blocked.

How would you handle the PDF generation?

For PDF generation part I will use LaTeX.
Reasons:

1. LaTeX is designed for producing beautifully typeset mathematics. Not only are the equations and mathematical symbols beautifully rendered, but LaTeX also does an exceptional job at handling visual components such as fonts, spacing, and line breaks.
2. LaTeX has a coherent package system that makes it relatively easy for users to write extension packages to provide additional features.
3. It is very easy and user friendly.
4. LaTeX files contain markup language that enables them to be readily converted to other outputs (e.g., PDF or HTML), allowing you to change or share your document more easily than if it was in another format.

How are you going to handle all the templating for the web application?

As for templating in web application I would prefer EJS because it goes well with node and mongo and it offers:

1. faster than Jade and handlebars.
2. really smart error handling mechanism built right into it. It points out, the line numbers on which an error has occurred so that one don't end up looking through the whole template file wasting time in searching for bugs.
3. Simple template tags: <% %>.
4. Custom delimiters (e.g., use <? ?> instead of <% %>)

And pdf templating will be handled by LaTeX itself.

Scenario 3: A Twitter Streaming Safety Service

Which Twitter API do you use?

I will use Tweet lookup GET method (GET /2/tweets) to return information about a Tweet or group of Tweets, specified by a Tweet ID.
The response returns one or many Tweet objects, which deliver fields such as the Tweet text, author, media attachments, and more.

This endpoint can also be used to receive up-to-date details on a Tweet, verify that a Tweet is available, or update stored details following a compliance event.

How would you build this so its expandable to beyond your local precinct?

Application scalability is the potential of an application to grow in time, being able to efficiently handle more and more requests per minute (RPM).

To make system expandible I will use Amazon Elastic Compute Cloud (Amazon EC2), it provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 eliminates need to invest in hardware up front, so one can develop and deploy applications faster.

Elastic Load Balancing is used to distribute incoming traffic of system across EC2 instances. This increases the availability of application. Placing instances in multiple Availability Zones also improves the fault tolerance in system. If one Availability Zone experiences an outage, traffic is routed to the other Availability Zone.

Reasons:

1. It has Virtual computing environments, known as instances
2. It provides preconfigured templates for your instances, known as Amazon Machine Images (AMIs), that package the bits you need for your server (including the operating system and additional software)
3. It provides preconfigured templates for your instances, known as Amazon Machine Images (AMIs), that package the bits you need for your server (including the operating system and additional software)
4. It has secure login information for your instances using key pairs (AWS stores the public key, and you store the private key in a secure place)

What would you do to make sure that this system is constantly stable?

System stability is the measurement of overall system performance, accessibility, and usability. It includes ensuring uptime in components such as Web and database servers, but it goes beyond that. I will make sure to follow below points to make system constantly stable.

1. **Define Stability**: Define and establish what the department can consider a stable computing environment, including server metrics and their effect on UX. They may include both a Recovery Time Objective (RTO), the maximum time tolerable without access to the application, and a Recovery Point Object (RPO), the maximum data loss that can be tolerated.

2. **Wo Management Policies**: Create and enforce a strict, well-defined change management process so failures don't occur when something is modified. This includes hardware and network configuration, patch installation and software version upgrades.
3. **Enforces End to End Test Procedure**: Common sense suggests higher quality software results in greater uptime. But make sure your company implements proper testing procedures to ensure quality across the board.
4. **Network Map and Monitor**: Slow or compromised communications can appear as an outage, directly affecting stability. Know what's out there on your global network: physical and virtual servers, network infrastructure, which ports are open, where vital communications occur and where your weak points are. The best way to do this is visually, using tools that help you interpret complexity at a glance.
5. **Server Monitoring**: To avoid downtime, you need to know when an issue occurs as it occurs, with enough insight to fix the issue quickly.

What would be your web server technology?

I will go with Express.js as a web server. It offers:

1. **Scale application quickly**: The first benefit of using Express.JS for backend development is that you would be able to scale your application quickly.
2. **Can use same language to code Frontend**: Another benefit of using Express.JS is that you would be able to do the code of both frontend and backend with the help of using JavaScript.
3. **Supports Caching**: Express.js supports the caching feature, and the advantage of the catch is that you would not have to re-execute the codes again and again. Moreover, it will help web pages to load faster than ever.

What databases would you use for triggers? For the historical log of tweets?

- For database I will probably choose redis for the **triggers** because of its speed. Redis Pub/Sub can be used to implement triggers. **Redis Pub/Sub** implements the messaging system where the senders (in redis terminology called publishers) send the messages while the receivers (subscribers) receive them. The link by which the messages are transferred is called channel. In Redis, a client can subscribe any number of channels.

- And for the **historic log of** tweets I will use **mongoDB** because it goes very well with the node server and it is open source.

How would you handle the real time, streaming incident report?

To handle the real time, streaming incident report I will choose **WebSocket.**

Reasons:

1. WebSocket is a naturally full-duplex, bidirectional, single-socket connection. Using WebSocket, HTTP request becomes a single request to open a WebSocket connection and reuses the same connection from the client to the server, and the server to the client.
2. It reduces latency. For example, unlike polling, WebSocket makes a single request. The server does not need to wait for a request from the client. Similarly, the client can send messages to the server at any time. This single request greatly reduces latency over polling, which sends a request at intervals, regardless of whether messages are available.
3. It makes real-time communication much more efficient. One can always use polling (and sometimes even streaming) over HTTP to receive notifications over HTTP. However, WebSocket saves bandwidth, CPU power, and latency. WebSocket is an innovation in performance.
4. WebSocket is all about Simplicity.

How would you handle storing all the media that you have to store as well?

To handle storing all the media I will go for any cloud enterprise like AWS s3. because cloud data can be easily accessible and shared.

Reasons to use AWS S3:

1. It provides Reliable and Durable Data Storage.
2. It has Enhance Security features.
3. It is very Simple to use.

What web server technology would you use?

For web server technology I will go with node.js.

Reasons:

1. Great performance! Node was designed to optimize throughput and scalability in web applications and is a good solution for many common web-development problems (e.g. real-time web applications).
2. Code is written in "plain old JavaScript", which means that less time is spent dealing with "context shift" between languages when you're writing both client-side and server-side code.
3. It is simple to learn and rich in framework (React, Angular, Vue, Ember).
4. Multiple modules (NPM, Grunt, etc.) and supportive community
5. Ability to keep data in native JSON (object notation) format in your database
6. Good for data streaming, thus for audio and video files, as example

Scenario 4: A Mildly Interesting Mobile Application

How would you handle the geospatial nature of your data?

I will use **Leaflet**, which is the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 39 KB of JS, it has all the mapping features most developers ever need.

Leaflet is designed with simplicity, performance, and usability in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code that is a joy to contribute to.

How would you store images, both for long term, cheap storage and for short term, fast retrieval?

To store images for both long term cheap storage and short-term fast retrieval, I will prefer any cloud enterprise like GCP Bucket or AWS s3. After storing images on cloud I will:

1. store local path of the image in the database.
2. store bucket path in environment variable.
3. And then dynamically generate url to load and access the image.

Reasons to use AWS S3:

1. It provides Reliable and Durable Data Storage.
2. It has Enhance Security features.
3. It is very Simple to use.
4. Cost effectiveness.

What would you write your API in?

I will use Node.js, which is a perfect approach to implement REST API Proxy and to comply with all performance requirement.

Reasons:

1. **Easy to write API**: There are a lot of ready to use modules to work with pure HTTP(s), REST API, Web Services, Sockets, etc. That can be used both to create API and to implement interaction with existing applications.
2. **Streaming support**: It is easy to stream results back to the client of API as they are getting from existing applications.

3. **Monitoring possibilities**: It is easy to get events on request/connection life cycle and collect metrics on API usage in Node.js.
4. **Authentication support**: Authentication strategies like OAuth, OpenID, Basic and others are available through passport.js, everyauth and other modules and can be applied easily to API.
5. **Lightweight, fast and scalable:** Node.js allows you to build fast, scalable API Proxy capable of handling a huge number of simultaneous requests with high throughput.

What would be your database?

For database I will choose **PostGIS**, an **open source, freely available spatial database extender** for the PostgreSQL Database Management System. So PostgreSQL (Postgres) is the database and PostGIS is like an add-on to that database.

In a nutshell **PostGIS adds spatial functions** like **distance, area, union, intersection, and specialty geometry data types to PostgreSQL**. Spatial databases store and manipulate spatial objects like any other object in the database.

The data is stored in **rows** and **columns**. Because PostGIS is a spatial database, the data also has a **geometry column** with data in a specific coordinate system defined by spatial reference identifier (SRID)**. It can answer** like 'how close is the nearest shop', 'is this point inside this area' or 'what is the size of this country'.