Lab 3

Scenario 1: Logging

In order to store log entries, I would be using MongoDB along with Mongoose to have a DB schema and have fields such as ***logId, userId, logMessage, timestamp***. Users can submit log entries from the client-side by using a POST request. For querying logs, users can submit GET requests along with the search keywords for the log they want to see. To check their own queries, the user can submit a GET request with the URL ***http://localhost:3000/myqueries*** that will show the user their own logs based on their Id that will be stored in the session. For a web server, I will be using Express.js since it allows the user to create robust APIs in a cleaner way and it is lightweight and doesn't obscure the core Node.js.

Scenario 2: Expense Reports

In this scenario, I will be using MongoDB to store all the expenses in a JSON format, and in order to make sure that the data structure submitted is always the same, I will use Mongoose to create a database schema that will have ***id***, ***user***, ***isReimbursed***, ***reimbursedBy***, ***submittedOn***, ***paidOn***, and ***amount***.

```
{
    id: ObjectId,
    user: String,
    isReimbursed: Boolean,
    reimbursedBy: String,
    submittedOn: String,
    paidOn: String,
    amount: Number
}
```

For a web server, I will be using Express.js since it allows the user to create robust APIs in a cleaner way and it is lightweight and doesn't obscure the core Node.js. For sending emails, I will be using the Nodemailer as it focuses heavily on security and is a single module with zero dependencies. It allows us to use HTML content as well as plain text and supports attachments and embedded images. For security, it uses TLS/STARTTLS. It even supports OAuth2 authentication and different transport methods in addition to the built-in SMTP support. For generating PDFs, I will be using wkhtmltopdf, a Webkit to take HTML to print it to PDF. Here we can compose an entire HTML document complete with CSS and images and send it through wkhtmltopdf to be converted into a PDF. Finally, for templating I'll be using handlebars as it allows us to keep your HTML pages simple and clean, and decoupled from the logic-based JS files.

Scenario 3: A Twitter Streaming Safety Service

I will use the regular Twitter API along with Node.js to make actual requests to the Twitter API. In order to be able to expand beyond a local precinct, I will be using MongoDB which allows us to scale the database according to our needs. For web-server, I will be using Next.js as it is lightweight, supports hybrid static and server rendering, smart bundling, and even route pre-fetching. For storing the triggers, I will be using Redis since it stores the data on cache and it is much faster to search through it if it is stored on the cache. For storing the historical log of tweets, I will use MongoDB as it is flexible to use and allows us to store data in JSON format. For streaming incident reports, I will use React.js since it is declarative and component-based allowing it to update particular components in real-time and it even

supports Emailjs and Next.js. To store the images I will use AWS for persistent file storage. To send the emails, I will use Emailjs since it supports SSL and TLS and different authentication methods. It even supports asynchronous sending of emails as well as multiple attachments.

Scenario 4: A Mildly Interesting Mobile Application

To handle the geospatial nature of my data, I will be using the location available in the metadata of the image captured and have a key in my database that stores that particular location. In order to display the images in their geographical location, I would fetch the location of the user from their device and make an API call using Express.js that would return the interesting images that match the location of the user. For storing the images, I would use AWS S3 as it is cheap for the long term as well as short-term storage and allows for fast retrieval of data. Finally, my database will be in MongoDB since it is lightweight, and is flexible to use.

{

id: ObjectId,

image_name: String,

location: String (containing the latitude and longitude in numbers)

}