Troy Tomasch

February 16, 2022

Lab 3

Scenario 1

I would use Elasticsearch as one of the primary components for this tech stack. Storage of the log entries would be managed using Elasticsearch. Elasticsearch is document based rather than relational so it will more easily allow for flexible logs to be submitted as the logs will not always have the same fields. One of the main benefits of Elasticsearch is that it allows for full-text searching which will make it much easier to search through the data and to find logs that one is looking for. This allows users to search through the entries simply through a search bar as Elasticsearch is able to find logs through this full text searching option. In order to submit these log entries, I would set up a site with a form on it to submit the logs. I would set up this site using Express.js and Node.js. Node has many libraries related to form validation that would allow the form to easily be developed. Furthermore, developing the site using these technologies would allow multiple routes to be developed for the user to submit a log as well as to view a log entry that they searched for. Express.js would serve as the web server in this setup as it allows for easy integration with Node.js and is easy to set up. This would be extremely user friendly as well as most users know how to enter data into a form as well as use a search bar to search for content.

Scenario 2

I would use Node.js to set up the web application and the server would run on Express.js. This would allow developers to use node modules to make tasks easier. For example, one of the tasks it would make easier would be sending the email. To send the email, I would use the Nodemailer module which would allow an email to be sent to the user that requested it as well as allow the PDF of the expense report to be attached to the email. This is an easy way to implement the email sending as we can simply include the node module and utilize it. In order to store the expenses for the sight, I would use the MySQL database. The data in this scenario is always structured the same way so using a relational database such as MySQL is feasible. This allows the data to be easily accessed and sorted as the data can conform to the relational database model. In order to generate the PDF to email to the user I would use LaTeX. LaTeX makes it easy to output the data from the expense report to a PDF as LaTeX outputs to a PDF. This PDF could then be attached to the email that is sent back to the customer. For the templating in the web application, I would use mustache.js. Mustache.js is a zero-dependency templating language which allows the web application to be lightweight.

Scenario 3

I would use the Twitter API in order to pull data about tweets from Twitter. I would query the Twitter API for tweets containing the keywords that were implemented on the site. The site, written with Node.js, would allow users to input keywords into a form that would then be stored in a MongoDB database for easy retrieval. This would allow it to be expandable to other precincts as they could just enter keywords that apply to their precinct and the site would retrieve tweets from the Twitter API that contain those keywords. MongoDB could also help with this as it could store the keywords for each precinct in a different document or even collections in the database to be easily accessed. For the email and text alerting system, I would use node modules which is one of the benefits of writing the site in Node. I would use the Nodemailer module to send email alerts to officers based on the content of the tweet and the Twilio module to send officers texts based on combinations that have been marked as extremely important. I would also use MongoDB to log these tweets that triggered alerts. There would be a collection in MongoDB for storing all tweets that would allow them to be retroactively searched through. There would also be a marker on each tweet to indicate whether it had triggered an alert. This would allow tweets to be sorted by whether or not they cause an alert. For the real time updates, I would use the Firebase Realtime database as the tweets could be stored here temporarily as they were parsed and then displayed live on the streaming site. The site would update live using the Firebase database and then all of the data could be stored long term in the MongoDB database. Additionally, AWS would be used to store any media that was found in the tweets as it is a reliable cloud option that can store a large amount of data. The media stored on AWS could then be pulled later and displayed if necessary.

Scenario 4

I would write my API in Node.js so that I have access to node modules to help with aspects of the project such as the geospatial location data. Node is also lightweight and scalable to allow the API to be built out quickly and expansively. For the database, I would use MongoDB as it allows the data to easily be separated into collections so the users would have a collection as well as the ‘interesting events’. Furthermore, MongoDB also allows the storage of GeoSON objects to be able to handle the geospatial nature of the data. This will allow queries to be done based on geospatial data about the user’s current location to present them with the posts from their area. For storing the images for the interesting events I would use a combination of CloudFlare and AWS storage. I would use CloudFlare to cache the images and be able to quickly retrieve them once the user has seen them and may access them again fairly soon. For long term storage, I would use AWS, this will allow a large amount of storage in the cloud and the images can easily be pulled down from a simple get request and then displayed to the user. Additionally, I would employ ImageMagick to format the photos users upload as well as formatting them again before they are displayed to the user.