Christopher Cousillas

Scenario 1
Logging:

To store all logging entries I would use Mongodb as it would allow for a document of logs that can easily be queried with their fields. They would each have unique IDs that can be queried through. If a user were to log their entries they could use a form providing fields to enter the information. This can be done with React as it can handle form submissions for the front end. Each of these log entries that are being sent through the form would also be sent to mongodb and stored in a document of other log entries. Their entries could be displayed on the web page nicely as well by sending the data through React with various different elements. If users wanted to view specific logs and query individual log entries I would add a search function as a component using React that will be able to query the data and display what the user wants. Backend functionality could be achieved by using express for the routes with node.js. This can handle any functional operations that are to be done on the collection.

Scenario 2
Expense Reports:

To store the expense reports the usage of mongodb would be useful to have as there could be a document storage of them. Submissions to this application will be handled by React taking in all fields in a form to be processed. Express will process this data by querying information in the database and storing the expenses. Express and node.js will host this application as it is easy to use and functional. Using LaTeX can handle the processing of email structuring and allow for PDF generation of the expense report along with many other functionalities for users to read easily. Doing this with node.js will be possible. A template I could use is a React component using the HTML that can be used for the PDF format.

Scenario 3
Twitter Streaming Safety Service:

The twitter API that will be used is Twitter API v2 which will allow us to fetch data on both users and their posts. Using the twitter API we can check the location of where the tweet was posted, making this expandable and usable anywhere. To Make sure the traffic on the service is minimal, we could possibly limit api calls or even limit usage of the service for users. Users will also be able to query a certain number of tweets which would have a limit as well, in order to lower the amount of unwanted calls on the server. The backend server of this application would be made using express and node.js. Mongodb can store a document of triggers which can be used to match against tweets to check for any triggers in those tweets. For any historical logs of tweets, another document can be made that can be queried through. To handle the real time streaming report I would use Socket.io with React that will handle the communication between the server and client. Media storage can also be made easy with mongodb as it can store media of up to 100MB. The technology stack that would be used in this application is MERN: Mongo, Express, React, and Node.

Scenario 4
A Mildly Interesting Mobile Application

The login service for this application can be done using a React front end with a form and authentication using express and Monogodb. Mongodb has geospatial operations built into it which can be used to handle the geospatial nature of the data in this application. The geospatial data would be stored using geoJSON that would allow for geo objects to easily read. Long term storage of these images would also be made possible using mongodb in documents where all images can be stored from a user. For fast retrieval access to this data I would use redis as any cached images would be displayed much faster. I would write the API in javascript using node.js. The general database for this application for all the storage would be mongodb, as with all the other storage in the application.