Scenario 1: Logging

Solution:

Before jumping on the tech stack let's see what logging is and why it is so important. So exactly what is logging? Logging is a way your code speaks to you. Example if a user sign-ins then how a system sends out a message of the user being logged in is what exactly logging is. There are alot of times when developers write a code and after a few days they have no idea why the code doesn't work, because they have no logging system in place. So it is very important to provide context and log it for the developer to debug it if something goes wrong. Most important thing to know is logs are temporary and not permanent. If they are not stored in your database, thus they have a short life span.

So for this lab seeing the requirements I will suggest Winston as the logger which is a logging library of node js. I will run npm install winston to use the library and require it. It has multiple transports and the log entries can be stored in a file "new winston.transports.File" or it can be shown in console, module, http, stream etc. I will be able to see the log entries in the file or in the console depending on the way you are storing the log entries for different environment variables. Winston also allows you to save the logs in mongoDB as it allows you to store the logs in the database. I have to install winston-mongoDB. Winston supports querying of the logs and you can find items logged between today and yesterday by using "logger.query". I will create an object called options and specify fields for querying. I will see all my logs using winston.stream and specify the start point for streaming. Express would be my web server for starting and then logging the entries. I will use express-winston as the package to run it using express-middleware.

Scenario 2: Expense Reports

Solution:
For the expense tracker the expenses will be stored in the **database** as documents. Under the collection Expenses with a user Id to specify which user's expense is the following. **Web server** to be used for this application will be HTTP server using the express framework of node js. Express dwells well along with the mongoDb and the templating using handlebars. **Nodemailer** a npm library will be used to handle the emails, which will make it easier. Pdfs will be generated using the **pdfmake**, a document generation library for server-side and client-side usage in pure JavaScript. Templating will be done using the **handlebars** for this application.

Scenario 3: A Twitter Streaming Safety Service

"Need to take care of RT incoming tweets using Apache Spark, Kafka and other Big Data Tools and Technologies"

I would use Twitter API v2 for getting access to the tweets. This is the official twitter api for developers and can be used in the local police department for keeping a track and scan for triggers.
For making it **beyond the scope of the local precinct** we can increase the area for monitoring the tweets. Twitter api offers the scope to use tweet location operators and profile location operators and increasing that scope will help to not limit ourselves to just our local police department. Since the twitter api

provides the data as json data mongodb would be the best database to be used to store the trigger words and the web server to be used for the application will be node server built using **express**. The triggers will be saved in the database and it will be permanently stored in the database so that for future reference the historical data can be found out using this db. **Mongoose** would be used to design a particular schema for tweets and the trigger words. For storing **media** 3rd party applications like microsoft azure or aws would be better as the images and videos would take quite a space considering we have to keep the historical data as well.

Streaming online incident report: If an incident occurs where the trigger is hit, that is if the trigger words are mentioned in the tweet by the user, then the user location and tweet location using twitter api will be returned and emailed to the police department. Email can be sent using **Nodemailer** a npm library, and sms text can be sent to the police department using **vonage api** considering the severity of the trigger

Stability of the application:

- Making sure the application is bug free by identifying the errors and solving them. Along with that good communication with customers is also important that the system is down. As well as fixing the system as soon as possible.
- A good memory management of the application that will ensure there are no unnecessary features taking up huge spaces.
- A rigorous testing of the app before as well as after the release and also after each update. As well as monitoring the application for performance and stability. Even logging the errors correctly every time one occurs.
- And the most important thing to ensure is taking user feedback and constantly being in touch with the users.

Scenario 4: A Mildly Interesting Mobile Application

Solution: This scenario tells us to create a web server side for the mobile application. The application is about taking pictures and uploading them so everyone can see. This application is very much specific to the geographical location and will show the pictures to the particular geographical location. Since the non-relational databases are much easier to implement I will use mongoDB for storing the **geospatial data** as it supports storing the query operations on it. A type of the **GEOjson** object and coordinates can be stored for showing the user of that particular geo location. Best way to store the images, both for long term, cheap storage and for short term, fast retrieval would be storing in 3rd party storage systems like **microsoft azure or aws**. I would personally not recommend file system storage as it will take up system's storage for quite a large extent. I would write my api in **express js** just because of flexibility, simplicity, extensibility and performance. Since it is a mobile application we can use node server using express along with **react native** for app development. And the database will **mongoDb** as mentioned above because of the GEOjson object and storing data via coordinates of latitudes and longitudes.