Choosing Technical Stacks

For this lab, you will not be programming, but rather analyzing several scenarios and determining what technologies would work well together to solve these problems. For each answer, back up your reasoning with *why* you chose each technology, strategy, etc.

Scenario 1: Logging

I would use Node.js and Express.js to build the web server. Routing can be done easily, and I have mostly used Express.js when dealing with http requests from client side. User's activity of submitting a log entry is equivalent to POST request, user querying and viewing their log entries could be GET request.
Each log entry will be stored as a document in MongoDB as storing, retrieving, and manipulating documents fields can be done using MongoDB functions like create, update, find.
The log entries have common fields like log title, log body, user who posted I will create a form with these input fields. The form's onsubmit action will trigger a POST action to server which will hit the POST route and then using create function will store the log details into DB. The form will also have a add customize fields button which will allow the user to add any number of custom fields. This button's onclick action will render two input fields, one for field name and other for its value. After adding the fields, user can submit his final log entry and that will get stored in DB.
Viewing log entries can be simply done using GET request which will use the findAll function of MongoDB to retrieve all the documents present in MongoDB.
For querying log entries, I will give a form input field to enter their search key word. On submitting the key word, I will query the db to search the document fields have match the search team and then send back the documents that match the search terms.

Scenario 2: Expense Reports

I would use Node.js and Express.js to build the web server. I will store expense as a document in MongoDB since creating, retrieving, and updating documents is feasible and convenient to use for me. I would create a form using html, bootstrap and css. The form's onsubmit will trigger a POST request which will store the document in DB. To handle all templating, I will use express handlebars which allows to avoid repetitive code. To handle emails, I will use Nodemailer module since it can be easily installed using npm and can be required. Nodemailer allows to easily send emails from server side. There will be reimbursed button which will generate an expense report in html format, this html will be converted into pdf using the html-pdf-node plugin. This plugin can be installed using npm and works well with Node.js

Scenario 3: A Twitter Streaming Safety Service

For my web server, I would use Node.js, Express.js as it is easy to configure. I also prefer Node.js, Express.js as the website should support CRUD operations. MongoDb's query function makes it easy, and I am comfortable using MongoDB for CURD. For email alerting systems, Nodemailer can be used as it is compatible to use with Node.js. To send alert for critical triggers, I would use push notifications. This can be done using web-push package in Node.js. To retrieve the historical logs, I would use redis as it stores data in in memory and retrieval is faster. I would use the Twitter Search API to search the tweets. I will use geolocation utils package to expand the search beyond local precinct. To make sure the system is constantly stable will do continuous integration and testing of system. Frequent feedback from user and modifying the system accordingly, also help to keep the system stable. To handle real time, streaming incident report I will use Web socket. I would store all media in the MongoDB database.

Scenario 4: A Mildly Interesting Mobile Application

I will build the web application using Node.js, Express.js. To store the image data, I will use MongoDB. Since the user must have account to use the service, I will create a login and signup page. CRUD operations can be easily done using MongoDb querying functions. To provide authentication and authorization, I will use express session and cookies to store user login details and session data. Node.js provides readily available packages for these operations like express-sessions. To handle the geospatial nature of the data I would use GeoJSON. GeoJSON is a standardized format for representing geographic data in JSON. I would store images for long term basis in MongoDB using the multer package which is a middleware that allows to upload image files to the server. For short term and fast retrieval, I would use redis, as it will store the images data in memory and caches it.