Scenario 1: Logging

Web server: Apache

Express.js: A web framework for handling an HTTP client request.

Middleware: Creating a Middleware function in the application file which will be used to generate logs for every user action.

MongoDB: Since we need each log to have a different structure (different fields), a non-structured DB like MongoDB is the best option to store the logs.

Simple custom form: To query the MongoDB, based on input filters from the user.

Whenever a user hits any route of the website, a middleware function will be triggered in which we can create a customized log entry containing common and customer-specific fields example: Username, Router name, Timestamp, etc., and store this log document into MongoDB.

Suppose a user wants to see his log entries then after proper authentication, the user can access the custom form. Once the user submits the form, it will hit the post route and execute the findLog(InputParams) function which will query the database for the given inputParams using the getall() function of MongoDB, fetch the data, and present it to the user in tabular format. This form will also have an option to add a custom field and its value to search for any customer-specific values.

Scenario 2: Expense Reports

I recommend using Node.js and Express.js to build the webserver. Since the structure of the data is constant, we should use an SQL database to store the expense data. For the User interface, we can create a form using HTML, CSS, Bootstrap, and validation can be done using jQuery. To handle all templating, I recommend using express handlebars which allows avoiding repetitive code. For Email functionality, we can use the Nodemailer module since it can be easily installed using npm and allows us to easily send emails from the server-side. We should include a button named “Reimbursed” in each expense record row while displaying the expense data in tabular form 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

Technologies and APIs:

- Twitter Search API
- the geolocation
- SMS-Service
- NodeMailer
- MongoDB
- Redis
- Express.js

We can create a simple form that will take keywords and their critical level and store it in DB. I recommend using the Twitter Search API to search the tweets. I suggest using the geolocation utils package to expand the search beyond the local precinct. To make sure the system is constantly stable I recommend keeping a secondary or backup server for media data, scheduling maintenance activity monthly, keep looking out for any official changes in the APIs. For the web server, I think we should use Node.js, Express.js as it is easy to configure and customize. I recommend storing the given trigger keywords with their critical level as key-value pair in MongoDB. This will help in alerting officers via text or mail based on the criticality level. For historical databases also I recommend using MongoDB. And using Redis to retrieve the historical log of tweets is the optimum solution. To handle real-time, streaming incident reports we can use a Web socket. To store the media files, I recommend storing the actual media files on an external database and only storing the metadata(name, filetype, time, date, incident_number, file_path) on MongoDB. Use SMS-service package to send an alert message via text message. Use Nodemailer to send an alert message via email.

Scenario 4: A Mildly Interesting Mobile Application

To handle the geospatial nature of the data I propose to use GeoJSON. GeoJSON is a standardized format for representing geographic data in JSON. We also need a geolocator package to get the user's current location and show the mildly interesting pictures in their geographical location. We need to create a signup and login page for users to create an account and login into the application. To provide authentication and authorization, I suggest using express session and cookies to store user login details and session data. Node.js provides readily available packages for these operations like express-sessions. CRUD operations can be easily done using MongoDB querying functions. an administrative dashboard can be easily created using HTML, CSS, WordPress, and Handlebars. And to populate the data, we can easily set up a route using express.js. I recommend storing images for a long-term basis in MongoDB using the multer package which is a middleware that allows uploading 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 cache it.