Scenario 1: Logging

How would you store your log entries? How would you allow users to submit log entries? How would you allow them to query log entries? How would you allow them to see their log entries? What would be your web server?

The best logging solution in my opinion is to use winston which is available as a npm package. While it is possible to build a logger on your own in NodeJS, it is highly tedious and when there is an already amazing library like winston available used by a lot of developers, it is best to leverage it as it also provides a lot of fantastic features like profiling, streaming, and querying which caters to a lot our requirements in the given scenario.

There are two ways in which I would handle log submissions. The first is when an error occurs in the application. By default, all application features which are errors produced by the application or the features which cause erroneous behaviour would automatically get logged. And therefore, winston is great as it provides a lot of in-built methods to capture data such as timestamps. Secondly, as for the log submissions by the user, the best way would be to provide a front-end form so that the user can submit their requests and provide a screenshot of the application which would help in querying the logs.

This brings us to log querying which can be done using winston. winston provides great querying support like Loggly which is another great logging technology provided by SolarWinds but it's too pricy and hence not included as our technology of choice in this scenario. The users can be provided with a user interface which they can use to query the requred entries or data as per their requirements.

winston also provides an in-built storage technique called 'transports' which helps us store our logs as per our choice like in a file system or in a database or in the cloud. There are 4 built-in methods which winston provides: Console, File, HTTP and Stream. Apart from the built-in ones, there are a lot of contributors which provide storage extensions for MongoDB, Amazon CloudWatch, Redis or CouchDB.

I would also use ExpressJS as my web server as it provides great simplicity over traditional NodeJS http module from which a server is difficult to create all by ourselves.

Scenario 2: Expense Reports

How would you store your expenses? What web server would you choose, and why? How would you handle the emails? How would you handle the PDF generation? How are you going to handle all the templating for the web application?

While NoSQL databases like MongoDB or Apache Cassandra are the best fit with Node or JS applications, in this case the use of a relational database like Postgres would make a lot of sense as the problem states clearly that the incoming fields would always be of the same type i.e., only the following fields, id, user, isReimbursed, reimbursedBy, submittedOn, paidOn, and amount would be submitted. This means the data would grow vertically and not horizontally and relational databases are best databases for vertically growing data as opposed to horizontally growing data for which NoSQL databases are best. I would also use ExpressJS as my web server.

For the purposes of generating pdfs, I would use a library like jsPDF which is the go-to node library for pdf generation. We can handle different paper orientation sizes as well as it's a very easy to use library rendering high quality pdfs. One of the downsides is the lack of support for UTF-8 characters but this can easily be overcome using custom ttf glyphs easily available on the internet.

nodemailer is my library of choice when it comes to email services. While there is a better option in terms of usage and features available in MailChimp which can be used by developers by installing a NodeJS wrapper called node-mailchimp, the fact that you might have to pay as your subscribers grow doen't make it a default option for me to use. Instead, sending emails using nodemailer package is the best open-source option in my opinion. If pricing is not an issue, I would instead use MailChimp for its ease of use and additional features.

For templating, there is a great node package available called jspdf-invoice-template which is an extension of jsPDF and is perfect in our scenario as this is specifically for invoices and expense reports.

Scenario 3: A Twitter Streaming Safety Service

Which Twitter API do you use? How would you build this so its expandable to beyond your local precinct? What would you do to make sure that this system is constantly stable? What would be your web server technology? What databases would you use for triggers? For the historical log of tweets? How would you handle the real time, streaming incident report? How would you handle storing all the media that you have to store as well? What web server technology would you use?

I would use the official Twitter Search API for this purpose as we need to keep a track of all the tweets. As this scenario requires urgency in delivery of emails because of nature of our application being threat analysis, I wouldn't use a package like nodemailer which is open source and may cause delay in delivery of important emails. Instead, I would use a paid and premium service like Postmark, which is a proven email service for its reliably high delivery speeds and efficiency. It can be integrated into our application using it's official npm package called postmark.

I wouldn't have chosen MongoDB as it wasn't a good database for storing historical data as the problem with NoSQL databases is that as data is constantly updated the document has to adjust for random parameters and as the application scales so does the data and performing CRUD operations becomes inefficient over time as a single document can only store a limited amount of data and new documents will have to be created very often as we would encounter large amounts of data everyday with different datatypes like texts and media. But this has changed with introduction of MongoDB's Time-Series Data collection. This was specifically developed by MongoDB to store time-sensitive and historical data. This is like a regular Mongo database but the underlying technology in the collection takes care of the problems I mentioned above, and it is highly performant and optimized to scale huge amounts of data. Hence, I would use MongoDB as my database with Time Series as my collection.

I would use ExpressJS as my web server and would also use one of Lambdas or Inferdo as my face detection API and node-rtsp-stream library to handle real-time streaming of incidents.

Scenario 4: A Mildly Interesting Mobile Application

How would you handle the geospatial nature of your data? How would you store images, both for long term, cheap storage and for short term, fast retrieval? What would you write your API in? What would be your database?

Google Maps Web API is the best technology to handle geospatial data. There are great cloud storage options for storing data. While Google Cloud Storage and Amazon Webservices are hugely popular and well-known names, products like pCloud and Icedrive also offer cheaper solutions for various types of storages like media. While I personaly use Icedrive mainly because it's very cheap and its retrieval times are super-fast, pCloud are veterans in this field and provide a slightly more expensive but reliable storage options as well. For database storage MongoDB is the best fit in my opinion as it’s easy to use and has high scalability.

Using technologies like ReactJS, we can provide users with a front-end dashboard where they can perform CRUD operations with ease.