Michael Reilly

Professor Hill

CS-554-A

11/15/21

I pledge my honor that I have abided by the Stevens Honor System.

Scenario 1: Logging

In this scenario, you are tasked with creating a logging server for any number of other arbitrary pieces of technologies.

Your logs should have some common fields, but support any number of customizable fields for an individual log entry. You should be able to effectively query them based on any of these fields.

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?

I would store the log entries of this system in a MongoDB collection. This is beneficial because by default mongodb collections do not need to have the same set of fields and the data type for a field can differ across log entries within the collection. The user can submit the log entries by making them into a javascript data object, with the fields and their corresponding values. From there this can be passed into MongoDB using GraphQL. GraphQL can use a mutation to insert the various log entries into the MongoDB collection with the various fields. In order to query the log entries of the MongoDB collection is by having a GraphQL querying the necessary components from the collection. This would allow for a variety of queries based on what the user is trying to collect and get returned to them. In order for the MongoDB collection to stay secure, I would require users to authenticate login credentials to get access to data within the server. Once they have this authentication then people would be able to see the log entries as they have access to search for logs within the mongodb collection. The web server I'd use with MongoDB is Express. The main benefit to using Express in this case is that it helps simplify node JS programming and make developing this scenario much easier as MongoDB and Express are both part of the MEAN stack.

Scenario 2: Expense Reports

In this scenario, you are tasked with making an expense reporting web application.

Users should be able to submit expenses, which are always of the same data structure: id, user, isReimbursed, reimbursedBy, submittedOn, paidOn, and amount.

When an expense is reimbursed you will generate a PDF and email it to the user who submitted the expense.

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?

I would store the expenses as data within a MongoDB collection. This can then have the expenses stored with the same fields within the whole collection. This collection can be stored on an Express server. The benefit to using Express server in this case is that it is part of the MERN stack with MongoDB and thus it is simple to implement them together for various web applications such as this one. To handle the emails, I would query the users field to find their email from that, and then make an ajax request to send an email. These emails can be stored in the MongoDB as the values of users rather than some other form of authentication such as a username. In order to generate the PDF, I would use LaTeX. LaTeX is a tool made specifically for creating and compiling PDFs, so it would be the perfect option for handling the PDF generation. The templating for the web application would be handled using React. Using React components can then template all that is needed within the web application, and would also be beneficial to use in this case as this implementation can then follow the MERN stack for web applications.

Scenario 3: A Twitter Streaming Safety Service

In this scenario, you are tasked with creating a service for your local Police Department that keeps track of Tweets within your area and scans for keywords to trigger an investigation. This application comes with several parts:

- An online website to CRUD combinations of keywords to add to your trigger. For example, it would alert when a tweet contains the words (fight or drugs) AND (SmallTown USA HS or SMUHS).
- An email alerting system to alert different officers depending on the contents of the Tweet, who tweeted it, etc.
- A text alert system to inform officers for critical triggers (triggers that meet a combination that is marked as extremely important to note).
- A historical database to view possible incidents (tweets that triggered an alert) and to mark its investigation status.
- A historical log of all tweets to retroactively search through.
- A streaming, online incident report. This would allow you to see tweets as they are parsed and see their threat level. This updates in real time.
- A long term storage of all the media used by any tweets in your area (pictures, snapshots of the URL, etc).

Which Twitter API do you use? How would you build this so it's expandable 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?

The Twitter API that I would use for this use case is the Twitter Firehose. The reason for this is that Firehose will return 100% of all tweets that match the criteria the police are searching for. In order to make this application expandable beyond local precincts is to use Apache Hadoop. The reason for using Apache Hadoop in this application is that it is useful for distributing huge data across many machines, or in this case many different departments as well, while still being able to perform big data analytics. In order to make sure the system is constantly stable is using a cloud-based server to keep all the data less maintenance than it would be if kept in-house. This can be done through various means and in this case I would use Amazon AWS. Amazon AWS would thus also be the web server technology. Due to the sheer amount of data it is more beneficial to outsource it to a cloud server than keep it within the precinct and then try to maintain it on their own. I would use Firebird and MySQL databases for database triggers. Firebird allows for multiple-row triggers while MySQL allows for both Data Definition Language and Data Manipulation Language triggers, which together should be able to cover all database triggers needed for the data being collected by the Firehose API. For the historical log of tweets I would use Microsoft SQL Server for log triggers. Microsoft SQL Server is beneficial as it allows for the same trigger to be attached to multiple operations and old and new table values are kept as separate tables so the logs are separated from current values. To handle the real time, streaming incident report I would use Splunk. Splunk can work from any machine both in-house or on the cloud and enhanced security features including incident reviews. All this data could again be stored in the cloud, rather than being maintained in-house. Thus storing the sheer amount of data that will be collected can be done with AWS. The web server technology that would be used would be Amazon AWS. Amazon AWS is both scalable and reliable with large amounts of data so it would be a perfect web server to use for this data to keep it both stable as well as secure compared to any in-house web server.

Scenario 4: A Mildly Interesting Mobile Application

In this scenario, you are tasked with creating the web server side for a mobile application where people take pictures of mildly interesting things and upload them. The mobile application allows users to see mildly interesting pictures in their geographical location.

Users must have an account to use this service. Your backend will effectively amount to an API and a storage solution for CRUD users, CRUD 'interesting events', as well as an administrative dashboard for managing content.

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?

To handle the geospatial nature of the data that is collected I would use Turf.js. Turf.js is beneficial for geospatial nature because it is both modular and fast and thus would be able to handle the geospatial aspects of the data well with Node.js. To store the images, since we have to think about both long term, cheap storage as well as short term, fast retrieval, I would store the images in an outside data store. One such example of this would be Amazon S3 as it is very cheap per gigabyte for the amount of storage that is necessary, while also being a cloud storage system and thus being able to hold an extremely large amount of data. I would write the API as a REST API in javascript with Express and Node.js. This will allow for the Express server to handle all the image data so it can be stored easily and quickly, and also be retrieved quickly. The database that I would use is a MongoDB database. This would follow with the MEAN stack as a logical and easy to use database along with Express and Node.Js as a good way to create a custom stack for this scenario as if need be MongoDB is also able to store image data if the need arose, as MongoDB is able to store a large variety of data types.