Lab 6

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?

Answer:

Tech stack for this scenario EXPRESSJS, MONGOOSE, HTML, HANDLEBARS, REST, REDIS

To store the log entries a NoSQL database like Mongoose can be used, Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box. The advantage of using mongoose is high performance and availability. Users would submit log entries via POST request /logs which can be maybe via a HTML form and the data will be stored in the database. Users can easily query for logs via a GET request /logs they can also view specific details as well by providing keys when making the GET request. Users will be able to see their log entries in a table like format which can be done using templating, the data to be displayed will be queried from the database. The web server which I will be using is the node express web server which can handle CRUD operations really well. For this scenario I can go a little bit further and maybe implement redis for caching the logs entries table as well.

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?

Answer:

The tech stack that I will be using for this scenario are NODEJS, HTML, CSS, HANDEBARS, POSTGRESQL, LATEX along with various nodejs packages.

I would store the expense as tables in Postgresql database, the table would be named as Expense table which will contains attributes such as `id`, `user`, `isReimbursed`, `reimbursedBy`, `submittedOn`, `paidOn`, and `amount`.

So, for the web server I would choose nodejs and would use node-postgres as it provides easy integration with postgresql with its framework. For emailing Nodemailer is a good option it will send out an email of the generated PDF expense (reimbursed report). For handling the pdf generation, PDFKit is a JavaScript pdf generation library for node which is quite seamless and easy to use. Since I am already using nodejs for this application I would prefer using express handlebars for the templating.

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 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?

Answer:

The tech stack that I will be using for this scenario are Django, MongoDB, Python, AWS S3.

Twitter provides various types of access level API's if possible, I will try to get the V2 Access Level Academic research which contains over 10 million tweets per month and it also provides some advance search operators. To make sure that the application is expandable, I will build a system that will

keep on pulling new data continuously and try to generalize the server so that the other precinct can also use it efficiently and easily. Can store the data in cloud as well for easy accessibility. For making sure that the system remains constantly stable , I will perform proper testing and maintenance and making sure that the technologies which are being used are available for all the precinct. For the tech stack I will be using Django as web server. For triggers I will use the redis sub-pub and will store the data in MongoDB for historical logs of the tweets. To handle real time streaming incident reports web sockets are the best option as they enable the sever and client to send messages to each other at any time. For storing media I will be using AWS S3 which can store large volume of data for cheap and still be secure at the same time. Will be using Django as web server since there are a lot of libraries available and we are dealing with a lot of data.

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?

Answer:

The tech stack that I will be using for this scenario are Google Maps API, NodeJS, Mongoose, AWS S3, Redis, REST.

To handle the geospatial nature of the data I will use google maps API. Since this is a mobile application if the user provides the location access, I can map

the data according to the location. I will be using node GeoIP- lite. This package gives all the details of the location. The user can have the option to manually enter the location with the fields such as street name and zip code. For storing the images and other media content I would use Amazon S3 since it is fast and reliable or else there is also option for Mongodb GridFS. For short term and fast retrieval, I can use redis as well. I will be writing my API in Nodejs for making the CRUD. The admin can monitor the data via the dashboard and block off any inappropriate content. I would be using mongoose as my database. Using mongoose, I can store all the details of the location very easily and it can be accessed as needed.