Assignment #8: Choosing Technical Stacks

Due on December 5th, 2019
Web Programming II
CS554WS—Fall 2019
Philip Barresi

Daniel Kadyrov

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?

Solution

Since the individual log entry has to support a ranging number of customizable fields, it would make sense to store the entries in a unstructured database like MongoDB.

A logging server will expect entries from different services which might be written in different languages. I think the easiest way to allow for log entry submittal, query, and see their entries would be through a request API.

I think an Express server will be sufficient for this task.

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?

Solution

Since the data structure remains consistent, it makes sense to use a structured database like MySQL or PostGreSQL. The web server I would use is Express since it will be able to handle the user interface, communication with database, and backend email sending.

I will handle emails through the Node package "Nodemailer". As much as I love LaTeX(even to the extent of writing this assignment in it), it potentially has a large overhead for this simple task and in previous work projects I have had a great experience using other PDF generator tools like wkhtmltopdf and ReportLab. I guess it all depends on how detailed and pretty the PDFs are required to be. These packages allow to program the templating.

Users will input the expenses through an online form.

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 `SMUHSP`).
- An email alerting system to alert different officers depending on the contents of the Tweet, who tweeted it, etc.
- 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?

Solution

Twitter offers an API that allows developers to search through and retrieve tweets as well as filter real-time tweets based on rules like keywords, locations and hashtags. (Twitter Rules and Filtering) has two different options based on payment, the enterprise "PowerTrack" API seems like the best choice since it allows up to 250,000 filters per stream, up to 2,048 characters in each and thousands of rules on a single connection. This API can be used to get a data stream of tweets based on the necessary keywords for the application and can be expanded to other keywords to service more areas and precincts. The API provides tweet details like content, media, tags, hashtags, and geographical information about where it was sent.

A worker will connect to the Twitter API through the NodeJS package "Twitter" and use the PowerTrack API to stream filtered tweets based on the targetted keywords. When a tweet matches the required keywords the script will send an email using "Nodemailer" and when a critical trigger is marked, a text using Nexmo or Twilio.

The server, which will be Express, will have a logging monitor to ensure that the system is stable. If there is an error, the system can alert the administrator or developers about the problem. The logging can display the incoming tweets as well as errors or a seperate web user interface can be created to visualize the stream of incoming tweets.

Since the required information for the tweets are consistent, a SQL (mySQL or PostGreSQL) database will have the lowest overhead for storage and searching abilities. While one table will store a historical log of tweets, another table will store values that users want to use as triggers. The Twitter API offers a URL to the image which the script can download, store in a file server (Amazon, Thinkmate, company servers etc) and store the location of the downloaded file in the SQL database.

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?

Solution

The handling of the geospatial data from the application depends on how it will be provided to the API. The ideal way to store geographical information would be through longitude and latitude. If the mobile application has been granted access to the phone's location services, the latitude and longitude information is provided. If the user wants to input their own geospactial information using street names or zip codes, the NodeJS package "Node-Geocoder" provides a way to convert it to longitude-latitude.

Long term image storage can be handled through a file data server service like Amazon, Thinkmate, or building a company owned file server system. Redis would be the best way to store photos for fast-retrieval.

The API would be written in Javascript using a Express server. Since the data will always be in the same structure, a SQL (mySQL or PostGreSQL) database would be the choice. Besides storing the user information and geospatial information, an address to the image file will also need to be stored.