CS-554 Lab 4

Jiaxian Xing 10439460

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 customizeable 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?

Store the log in the mongoDB it support any number of extra fields.

Use an form the submit the post request. The customizeable key value can be store in the json file format.

Query based on the one of key value pair. Using the mongo query function to find the entry.

See the log as the json file. It's easy to process and ready to parse.

expressJs as the server. It's light weight.

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?

Since the data structure is always same we can use the Relational Database.

ExpressJs server cause is easy to setup and cheap to running.

Email we can use the AWS SES service.

PDF maybe we can use the pdfmake(https://github.com/bpampuch/pdfmake) I founded online.

Templating can try to use the React. It's looks good.

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?

Streaming API in Twitter for Node.js (https://www.npmjs.com/package/twitter) can use to check the real-time data. Search API and checking the history data.

Put the app on the Cloud Service to beyond local precinct.

Using the automatic backup and automatic restart to make sure the stable. Aws cloudwatch, Gcp snapshot and Nodejs forever.

MongoDB for the use of database, and store the logging information. It's also can be used as long term storage.

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?

The Geolocation API allows the user to provide their location to web applications (https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API).

Redis for cache fast retrieval. And LRU to transfer the data in to MongoDB for long term storage.

Authentication for the user sign up and sign in.

Also save the user preference, RESTful API for the content management.