CS-554 Lab 3 Kevin D'sa CWID:20009000

Choosing Technical Stacks

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?

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?

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?

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?

Scenario 1: Logging

How would you store your log entries?

- I would store the log entries in a No SQL Database such as MongoDB as the logs should have some common fields but support any number of customizable fields. I would store common fields such as timestamp, user, log text, technology name followed by customizable fields

How would you allow users to submit log entries?

- User would submit log entries via POST request to an API endpoint Api/logs which would trigger the transports.MongoDB() command of Winston which would make an entry in the mongodb database

How would you allow them to query log entries?

- Users can query their logs using the query string parameter supported by express js using which the user can make specific query requests. Like ‘logger/page=n’

How would you allow them to see their log entries?

- The user can see the log entries using a simple get request which will query the mongodb database

What would be your web server?

- I would use Node JS Web server using Express js as the web server as it enables powerful querying and easy crud operations with MongoDB database. We would also use a npm package Winston which is a logging package that logs all the details that are important to the user

Scenario 2: Expense Reports

How would you store your expenses?

- Since the data will have fixed fields that means it will always have structured data, the best option would be to go with a SQL based database in this case I would use MySQL. This would help keep the data with the same data structure that is id, user, isReimbursed, reimbursedBy, submittedOn, paidOn, and amount.

What web server would you choose, and why?

- Express JS, as it is easy to set up and create routes, would be a good option to facilitate all the three requirements, that is pdf generation sending Emails and Database entry as it is supported by node js which would give us access to all npm packages

How would you handle the emails?

- NodeMailer, is a npm package which is a famous package for SMTP.
- NodeMailer works best with mailing systems like outlook or zoho mail for sending mass emails

How would you handle the PDF generation?

- PDFKit, is a npm package. pdf kit can be used for generating custom pdf. We can include images, transaction details as well as send the generated pdf via email which would be required and essential in this case scenario

How are you going to handle all the templating for the web application?

- We can use any Nunjucks for this web application it offers features like Extends that allows us to specify template inheritance and macro that helps us to define reusable chunks of content

Scenario 3: A Twitter Streaming Safety Service

Which Twitter API do you use?

- I would use the official twitter api so that the information I'm getting is official and correct Twitter API v2

How would you build this so its expandable to beyond your local precinct?

- I would build this in Node Js using MongoDB for the database, so this is scalable with Realtime data. In addition, NPM has multiple packages that we can use as per our requirements in the future

What would you do to make sure that this system is constantly stable?

- would use Continuous Integration and Tests to make sure the system is stable along with multiple test suites like smoke testing, load testing and integrity tests for the database. Automatic replication of the database would also be enabled.

What databases would you use for triggers?

- I would use MongoDB as the database as the type of data would be unstructured I would also user npm package "mongo-triggers" for triggers   it would function as a middle ware this will be invoked when data is inserted or when we use a find one and the middle ware finds the trigger word

For the historical log of tweets?

- MongoDb would be able to store the data using time stamp and assign a unique id to every tweet. That invokes the trigger

How would you handle the real time, streaming incident report?

- Every time the api is queried we can search for the trigger words. If we find that the tweet has the trigger word we would store it in mongodb for further review. In this case we could set up node schedule an npm package that gets invoked at a user specified time to scan through the tweets at regular intervals

How would you handle storing all the media that you have to store as well?

- As the twitter api responds with "data": { "attachments": { "media_keys": ["7_1138489597158199298"]……. .I would store the media key as a reference and query the media using its key and store the media file in fire base/storage

What web server technology would you use?

- I would use node js as the server-side scripting and Express as the server

Scenario 4: A Mildly Interesting Mobile Application

How would you handle the geospatial nature of your data?

- Using the NPM package react-native-community/geolocation. we can find the users current location. This package is for React Native so for this solution I will be using react native as part of the user interface for the mobile application. The npm package started above offers methods like getCurrentPosition() which would return the users current location. We would use this in the case when the user uploads an image to which we can store the latitude and longitudes and also to show the user pictures of mildly interesting things that are near the users current location

How would you store images, both for long term, cheap storage and for short term, fast retrieval?

- I would use firebase Storage @react-native-firebase/storage as it is a cloud storage solution, it is easy to implement and would be a long term and cheap solution another solution would be AWS-S3 in which aws offers multiple storage options and plans to suit the requirement

What would you write your API in?

- I would write the API in Node JS as it has access to npm that offers multiple packages that would be required for this scenario. we would use packages like nodejs-mobile-react-native to create a bridge between the react native application and the API

What would be your database?

- I would use MongoDB as the database to store user data and other information. Mongo db is the preferred option as it also has geospatial support. Hence we could use it to store the Images current location. It has multiple GeoJSON types like point and polygon. It also offers Geospatial Indexing