CS – 554 A Lab-3

Choosing Technical Stacks

Anchal Bansal
10478699

Scenario-1 Logging

For creating this logging system, I would be using NoSQL Database such as MongoDB to store the log entries. MongoDB is object-oriented, dynamic, distributed and can be scaled. It stores data as JSON objects following document-based model. To create a seamless logging system where logs can easily be saved and also queried, I would be using Apache Kafka that would be the distributed messaging service that would connect to Mongo DB using MongoDB's sink and source connecter for writing and reading the data. Kafka consists of three parts- a producer (that produces the messages), broker (that acts as intermediate between producer and consumer) and consumer (that consumes messages). Besides setting up these fields, Kafka needs a Zookeeper setup that helps in management tasks while running this distributing system and helps in maintaining configuration information and naming. Kafka stores each entry with timestamp. I can store the data in Mongo DB with source, timestamp, logText as mandatory fields for each log and customizable fields can be added for logs being added as per the requirement. In case, the data increases, we can easily increase the number of producers and consumers to handle large logs. Read and write in Kafka can be done with O(1) time. To submit log-entries, I would expose and API endpoint such as /api/createlogs which would be a POST request. Users can use this for entering their logs. To query the data, users will have access to GET API- api/searchLogs?search1=a&search2=b where users can give the search criteria as the parameters in the API and Kafka will run its filter method to retrieve the data from database. To view those log entries, users can do a GET request to api/logs?userid=myid which would return all their logs in a table format. This table can be created using HTML so that users can have a prettified look of their logs. I would use NodeJS and Express to create the web server. NodeJS can be easily scaled, gives better performance than other servers, both front-end and back-end code can be written easily with NodeJS. I would use ExpressJS framework for NodeJS which allows to define routes of application based on HTTP methods and URL's. Middleware functions offered by Express help perform additional tasks on request and response.

Scenario-2: Expense Reports

To store the expenses, I would use MongoDB since it is document based database where the data can be stored in JSON format. Data stored in the database would like below:

Id	User	isReimbursed	reImbursedBy	submittedOn	paidOn	amount
2565	John Cena	Yes	Jimmy Fallon	02/14/2022	02/20/2022	$5000
2566	Big Show	Yes	Tim Hans	01/14/2022	01/30/2022	$2000

I would use Node JS as my webserver since it can be easily scaled, gives better performance than other servers, both front-end and back-end code can be written easily with NodeJS. It can be customized according to the business requirements. I would use ExpressJS framework for NodeJS which allows to define routes of application based on HTTP methods and URL's. Middleware functions offered by Express help perform additional tasks on request and response. For handling emails, I would use npm package called 'Nodemailer'. Nodemailer makes it easy to send emails from the application. To set up a Nodemailer, a Nodemailer transporter needs to be created using any transport mechanism such as SMTP. By using sendMail() method of the created transporter, we can specify sender's address, list of receivers to whom message needs to be sent, text body and html body of the message. To handle the pdf-generation, I would use npm's wkhtmltopdf package since html can be easily converted using this package. The html format of the file can be provided to this and name of the output pdf file and page size can be specified in this to generate the pdf. Templating for the web-applications can be handled using express-handlebars. We can store the templates under views folder in the node application and render the template we want by choosing it from that views folder doing res.render. Handlebars provide minimum templating and logicless engine that keeps the view and code separate. They help us avoid repetitive code by compiling the final Dom structure of the site via logic, compiled by task runners like GULP.

Scenario-3: A Twitter Streaming Safety Service

For creating Twitter Streaming Safety, I would be using Filter Realtime Tweets API that returns the results of the tweets which match with one or more predicates specified by the user. It supports both GET and POST API. I would leverage the geo package offered by twitter which gives you endpoints to get information about a particular place as well as places near that location. Hence to expand this service beyond local precinct, I would use these API's to get information about other areas. This service expects an id that should be passed for getting information about that place and use GET geo/search to get information for expanding around my local area. To make sure that this service is stable, I would make sure that I test this service using load tests, smoke tests and use Continuous Integration and Testing for this purpose. Also, I would try to make this code clean, modular and without any duplication. Also, to make it stable, I would use more number of servers and host my application on them and use a load balancer such as nginx that would help in case any server is down. I would use Node JS as my webserver since it can be easily scaled, gives better performance than other servers, both front-end and back-end code can be written easily with NodeJS. It can be customized according to the business requirements. I would use ExpressJS framework for NodeJS which allows to define routes of application based on HTTP methods and URL's. Middleware functions offered by Express help perform additional tasks on request and response. For handling emails, I would use npm package called 'Nodemailer'. Nodemailer makes it easy to send emails alerts from the application. I will use npm package Twilio for sending SMS alerts by using its messaging API. I would use Mongo DB as my database for storing the data since it is object-oriented, dynamic, distributed and can be scaled. It stores data as JSON objects following document-based model. To create a seamless service where historical logs of tweets can easily be saved and also queried, I would be using Apache Kafka that would be the distributed messaging service that would connect to Mongo DB using MongoDB's sink and source connecter for writing and reading the data. Kafka consists of three parts- a producer (that produces the messages), broker (that acts as intermediate between producer and consumer) and consumer (that consumes messages). Besides setting up these fields, Kafka needs a Zookeeper setup that helps in management tasks while running this distributing system and helps in maintaining configuration information and naming. Kafka stores each entry with timestamp. I can store the data in Mongo DB with source, timestamp, logText as mandatory fields for each log and customizable fields can be added for logs being added as per the requirement. In case, the data increases, we can easily increase the number of producers and consumers to handle large data. Read and write in Kafka can be done with O(1) time. This would also help in handling real time streaming incident report. To help police to see their tweets real-time as they are getting parsed, I would write front-end that shows streaming online incident report using AJAX such that the webpage gets refreshed every 5 seconds. I will set a timer in my AJAX that updates this webpage with real-time tweets every 5 seconds since it would not reload the page and fetch the data in background from Kafka consumer. This is similar to how LinkedIn uses Kafka for storing their logs in real time. To store any media that comes with tweets, I would be using npm package called ImageMagick since this can be used for image manipulation and storage. I can also use npm package called node-media-server that helps in dealing with media if it's not just images. This media would be stored in mongoDB which can be done by creating package with Mongoose and data type can be specified as buffer. I would use NodeJS and Express as my web server technologies for this. NodeJS can be easily scaled, gives better performance than other servers, both front-end and back-end code can be written easily with NodeJS. I would use ExpressJS framework for NodeJS which allows to define routes of application based on HTTP methods and URL's. Middleware functions offered by Express help perform additional tasks on request and response.

Scenario-4 Mildly Interesting Mobile Application

For this mildly interesting mobile application, I would be using Google Maps API to handle the geospatial nature of the data. I would use JavaScript to write this API. Google Maps API provides a class called google.maps. This has different layers which are the objects on the map that consist of one or more separate items. The Maps API manages the presentation of objects within layers by rendering their items into one object. We can add a layer by using setMap() function. Class google.maps.DATA()allows to markup map with variety of overlays and allows user to choose how they wish their data to be represented. Using this we can also add GeoJSON data to the map using loadGeoJSON() method. We can use map.data.setStyle() method for specifying how the data should look. Since this is a mobile application, we can utilize front-end JavaScript framework such as React-Native to build its front-end for administrative dashboard that manages content. For storing images, I would be using npm package called ImageMagick that can be used for image manipulation and storage. For short term, fast retrieval I would use MongoDB so that it can be retrieved locally. For long term, cheap storage I would use cloud storage such as Amazon S3. Amazon S3 has the cheapest storage cost and has high data availability, performance and can be scaled according to the requirements. The data is stored in buckets which can be expanded till 5 TB each. S3 provides S3 GET to modify and process data as per requirement. I can use custom code to modify the data returned by standard S3 GET requests to filter rows and dynamically resize images .The database I will use is MongoDB since it is object-oriented, dynamic, distributed and scalable and follows document-based model. It provides high performance and availability. This media can be stored in mongoDB by creating package with Mongoose and data type can be specified as buffer for that. Since this application requires user to have an account to use the service, we could create a signup, login and logout pages using JavaScript handlebars and maintain their sessions by storing cookies using express-session package. We can use middleware functions to keep a check such that only authenticated logged in users can use the service at any time. The user details can be stored in a separate collection in MongoDB by encrypting the sensitive information such as passwords.