CS-554 Lab 3
Choosing Technical Stacks

Jiayin Huang 10477088

Scenario 1: Logging

In this scenario, I would choose MongoDB for storing log entries. MongoDB is suitable for handling unstructured data and can store documents with variable fields. MongoDB has cultivated a reputation as a versatile, flexible, non-relational database. It uses JSON, which is widely used across for frontend and API communication.

To allow users to submit log entries, I would create a RESTful API that allows user to send their log data to the server. The API should include secure authentication and authorization mechanism to ensure that only authorized users can access.

I would choose Elasticsearch to allow user to query log entries. Elasticsearch isa REST based search and analytics API for storing huge data and searching through it. It is also an incredibly fast full text indexing database. Searching is an inherently hard operation, in every single database. Elasticsearch is an advanced indexing platform that focuses in searching, rather than storage.

To allow users to see their log entries, I would create a web-based dashboard that allows users to search and visualize log data using JavaScript.

My web server framework would be Express in this scenario, it provides a simple and flexible API for building web applications and APIs. I will use Express to handle HTTP requests and responses, implement middleware for authentication and request handling and integrate it with MongoDB. It is easier to handle common tasks, such as routing and error handling.

Scenario 2: Expense Reports

For storing expenses, I would use SQL database. SQL databases are well-suited for handling structured data like expenses. Expenses have a fixed data structure that includes fields like id, user, isReimbursed, reimbursedBy, submittedOn, paidOn, and amount, and these fields can be easily defined and indexed in a relational database.

My web server would be Node.js with the Express framework in this scenario. Node.js is fast, scalable, and easy to use. It’s a popular choice for building web applications.

To handle email sending, I would use a third-party email API like SendGrid. This service provides a reliable and scalable way to send emails and can handle email templates and attachments like PDFs.

To handle the PDF generation, I would choose LaTeX library to generate high-quality PDFs from HTML templates. LaTeX is a content first approach, which can then be combined with a set of styles. It is a markup language, similar to HTML. Anytime your product must generate PDFs, LaTeX is a valid option. It is also one of the only options! LaTeX is effectively the language you can write PDFs in.

To handle templating for the web application, I would like to use React fronted framework. React can address the problems in building and managing complex web UIs. For front-end applications. It's component-based architecture using pure JavaScript. React provides a powerful and flexible set of tools for building dynamic and responsive user interfaces.

Scenario 3: A Twitter Streaming Safety Service

I will use Twitter Streaming API to get real-time updates of tweets containing the keywords. The Twitter Streaming API provides a constant stream of data, which is ideal for this use case.

To build a system that is expandable beyond local precinct, I would choose cloud infrastructure such as AWS or Microsoft Azure. With cloud infrastructure, I can easily deploy the system across multiple regions, which allows me to scale it up or down based on demand.

To make sure that this system is constantly stable, I will deploy continuous integration (CI/CD) pipeline to ensure that changes are tested, build, and deployed automatically. Also, I might implement a logging and monitoring system to identify and address issues in the system before they cause significant issues. Logging can help to track system events, while monitoring can help to identify system errors.

I would use React as the web server technology. React provide a scalable and secure way to build CRUD website and handle the email and text alerting systems.

For triggers, I would use MongoDB. It is well-suited for handling unstructured data like keywords and combinations. I can use MongoDB to store the combinations of keywords that trigger alerts, along with their associated information like alert priority and notification settings.

To storing the historical log of tweets, I will choose SQL since it is well-suited for handling structed data like tweet metadata. I can use SQL to store the historical log of tweets with users' metadata like timestamp, userId, and tweet content.

To handle the real-time streaming incident report, I will consider using Socket.io, which enables real-time communication between the sever and the client.

For storing media used by tweets, I would use a cloud-based storage service like Google Cloud Storage. Google Storage provides reliable storage for large amounts of data, and handle media files like pictures and snapshots of URLs.

Scenario 4: A Mildly Interesting Mobile Application

To handle the geospatial nature of the data, I will choose MongoDB as geospatial database, it provides support for geospatial data types and spatial queries, which is ideal for handling location-based data.

To store images for both long-term, cheap storage and short-term, fast retrieval, I would use a combination of cloud storage services like Google Cloud Storage, this will enable me to store images in a cost-effective and ensuring fast retrieval times.

I would use Node.js with Express framework to write my API in, this would provide a flexible structure for building the API endpoints for CRUD users and interesting events.

My database will be MongoDB, which can handle the unstructured data like interesting events and user profiles. It can provide a highly scalable, efficient solution for the Mildly interesting mobile application.