CS554A_Lab3
Name: Cheng Chen
CWID: 10473438

Scenario 1: Logging

The system running online will generate a large number of logs every day, and the logs will contain various information such as errors, warnings, and user behaviors. Usually, the system records log information in the form of text, which is highly readable and facilitates daily locating problems. However, after a large number of logs are generated, in order to mine valuable content from a large number of logs, further data storage and analysis are required. At this point I would choose to use MongoDB to store log entries. Because it has the following advantages: the form of JSON-style files, oriented to document storage; indexable to any key; rich query; attributes are easy to expand, etc. General logs include time, access source, user, access destination address, access result, system and browser used by the user, etc. We can set these fields as required. Each field will be stored in the same object as a key. E.g:

```
{
    _id: ObjectId('xxxxx123456789'),
    host:  “127.0.0.1” ,
    time: ISODate("2000-10-10T20:55:36Z"),
    user:  “example” ,
    path:  “/public/demo.js ” ,
    user_agent:  “xxxxxxxxx” ,
    other: {
            xx: xxx
        }
}
```

For allowing users to submit log entries, simple static HTML pages and AJAX can be used. In HTML, a login interface will be provided, and users can use the system only after login, which is the most basic. After logging in, there will be some sub-pages corresponding to different functions, such as submission log, query log, etc. In the page where the log is submitted, there will be a form. This form contains not only the basic field information of the log, but also provides customizable fields. For example, if you set an add button, AJAX will generate a new label to record the new key and values of the field after clicking. After submission, these custom fields will be stored in the “other” object as keys and values in the log object.

When querying, users will be able to query for each attribute, and at most two attributes can be queried at the same time. For example, you can view all logs of user A, or you can query all logs of user A on a certain day. At the same time, ordinary users are only allowed to access basic logs, and only administrators can see all levels of log information (such as some alarms, etc.).

The web server can use the cloud server of a large company, such as Amazon's. Because they have good security management capabilities, by purchasing complete security services, we will not worry about data loss or a large number of intrusions.

Scenario 2 ：Expense Reports

As a complete web application, it starts with a login page. After the administrator generates a user account in the background, the user will use the account password to log in to the web application.

After the user logs in, there will be a form for submitting reimbursement, where the user will submit some necessary information, such as: whether to reimburse, object of reimbursement, submission time, payment method and amount, etc. The data will be stored in the MongoDB database and the collection will have these keys: id, user, isReimbursed, reimbursedBy, submittedOn, paidOn and amount. After the user enters as required, the client will check whether the entered information meets the requirements, such as date format, whether there are special characters, etc. All information will be inserted into the database after meeting the requirements.

For generating PDF files, I will choose the JsPDF library, which will generate a PDF file in a specific format through the JsPDF library on the client side through validation and pass the content to the database, and download and save the PDF file on the client side. Here we are using AJAX to fetch the data and generate a complete table on the page, which does not have to be displayed. Write another function to set the size and other values of the PDF page. Next, you can generate a PDF and save it locally.

Next, we use AJAX to send the generated table to the server in the form of HTML statements. On the server side, we can use ASP.NET to send emails. Simple emails can be sent directly using the SmtpMail class. The SmtpMail class is the most basic class in the System.Web.Mail namespace. It is the core class that implements the function of sending e-mails. No matter how complex the sent e-mails are, they are ultimately sent out through the Send method in the SmtpMail class. SmtpMail.Send(from, to, subject, message). The four parameters in brackets represent the sender's email address, the recipient's email address, the email body and the email content, respectively. Here we can call the email address corresponding to the username through the database. Then read the mailto, mailfrom, mailsubject and mailBody in the form through the web page, and then assign to, from, subject and Body attributes of the mail object respectively, and finally call SmtpMail.Send to send the mail.

Regarding templates, since it's awkward to store templates in a file with your application code, and doing so clutters up your code, we'll show you how to read them from a separate file using Node's filesystem API .

Scenario 3: A Twitter Streaming Safety Service

Here I will use PowerTrack API from Twitter API. This is achieved by applying a PowerTrack filter language to match tweets based on various attributes including user attributes, geographic location, language, etc. As long as the relevant location information is set, it can be expanded to areas outside the local area. In the PowerTrack API request body, we can set the publisher, data format, date, date, title and various rules. And supports up to 1000 PowerTrack rules.

Next you can set up a trigger to retrieve it every 30 minutes or an hour. When a tweet that triggers the corresponding alert is found, an alert email will be sent to the specified mailbox

through SmtpMail in ASP.NET. At the same time, redis will temporarily store all the information of this tweet and install the set rules for security analysis to obtain its threat level. After analysis, the threat information is displayed in real time.

After the analysis is complete, all information about this tweet will be stored in the historical database. MangoDB can also be used as a database here, which is a database that is very suitable for storing JSON format. Here the user can mark the investigation status of this record and can also retrieve all the history records at any time. In addition, there will be a document in the database dedicated to storing log information. Including trigger records, processing records, query records and so on. All media in tweets (images, URL snapshots, etc.) will be stored in the database using GridFS. At the same time, we can also save these non-text information in the cloud server, such as the oss of aws, which can greatly reduce the burden of the local server. It is also convenient for subsequent expansion to police stations in other regions.

Streaming event reporting can use the open source Kafka to develop a streaming data platform, process and transform streaming data continuously and in real time, and open the results to the entire system. It is easy to integrate with streaming data platforms through common streaming frameworks such as Storm, Samza or Spark Streaming. These streaming data processing frameworks provide rich API interfaces that simplify data transformation and processing.

When the tweets that trigger the alert are analyzed and stored in the historical database, redis can delete the local cache, which can avoid the instability of the system due to too much data.

Scenario 4: A Mildly Interesting Mobile Application

After logging in, users can upload interesting photos they have taken, which will be stored in the database. We can choose to use MongoDB to store these images. MongoDB is a database based on distributed file storage, which is very suitable for storing image information. We can use GridFS to store images in the database. GridFS uses two collections to store files. A collection is chunks, which are used to store binary data of the file content; a collection is files, which is used to store the metadata of the file.

There is no doubt that these photo information contains geographic information such as latitude and longitude. Then we need to extract the latitude and longitude information from the uploaded photos to obtain the positioning. Here you can use exif-js API to get various photo information. Use EXIF.getTag(this, 'GPSLongitude') and EXIF.getTag(this, 'GPSLatitude') to obtain the photo storage information, and then obtain the latitude and longitude through a fixed calculation formula. At the same time of uploading, these longitude and latitude information will also be stored in the database as attributes, which is convenient for later search.

In addition, we can also temporarily save the captured image information in Redis for some processing. Here you can use the Georadius command of Redis. Georadius takes the given latitude and longitude as the center, and returns all the position elements contained in the key, and the distance from the center does not exceed the given maximum distance. You can also use Geohash to save the coordinates of the geographic location. At this time, we get the location information near the user, and then use this information as a keyword to find out

whether there are related pictures in the database. Short-term fast retrieval can be done in Redis. When the application is running, some information will be saved in Redis. When no relevant information is found in Redis, a complete retrieval will be performed through mangoDB.