**Android Application Development in Backend Side**

Instructor : Andrew Kaplan

Research Paper by Walt Cho

DigiPen Institute of Technology (GAM 450)

**Table of Contents**

Introduction……………………………………………………………………………………1

Network file System (NFS)……………………………………………………………………2

Relational Database Management System…………………………………………………….4

Encryption of User Data ………………………………………………………………………6

POST / GET Methods………………………………………………………………………….9

Tech Stacks…………………………………………………………………………………...10

Conclusion…………………………………………………………………………………....10

Reference……………………………………………………………………………………..11

**Introduction**

In our Project BOJA, users will evaluate their favorite movies, and the BOJA application will recommend them to users based on the data. The more users evaluate the BOJA application, the bigger the data will be, and the more accurate the recommendation will be. I was in charge of the backend part here, and I'm going to give a backstory of the knowledge and skills needed to construct this efficient backend. At the back-end, two servers are running. One is for managing user data, connection for the client, and the other for managing machine learning servers. This is configured like this using a low tier of AWS servers, and we'll explain how we're going to deal with this in the back. The application also has a total of three tables, configured as RDBMS for quick query search within these three tables. Finally, I will talk about how to protect user data by sending packets.

* Network File System(NFS)
* Relational Database Management System
* Encryption of User Data during the connection

**Network File System (NFS)**

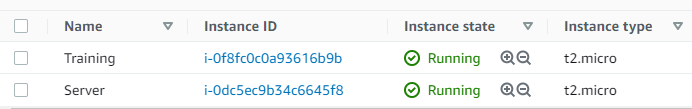
Network File System (NFS) is generally considered a state in which two servers share a single folder. The reason for using this is that two servers can read and write a single file. Originally, we decided to configure our project on one server, but the performance of the server was low, so we couldn't run the two servers at the same time. A simple illustration shows Figure 1.1 below.



<**Figure 1.1**>

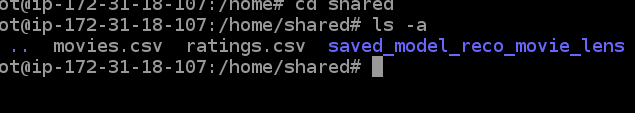
As Figure 1.1 shows, it is a very simple structure. One folder is shared by two servers, which would be a good option when using a low-performance server, or when dividing two servers. Our project will play with two \*.csv files that ML servers need to learn and the \*.pb files that servers need to learn and then read as dockers, and two servers will read and update them. The Back-end server will receive data from users and update \*.csv files, and the ML server will read the \*.csv files, and will update the \*.pb file. Then the updated \*.pb file, the Back-end server will give users updated information.

One of the features offered by AWS is the Elastic File System (EFS), which is the concept of NFS. This will tell you how to easily configure NFS between two instances. Figure 1.2 shown below is two servers that are currently being used in our project.



<**Figure 1.2**>

The servers must leave the ports open to connect. Our server alone has left 3,000 ports open to connect with clients. For EFS, port 2049 must be left open. With port 2049, you can specify a folder and mount it.

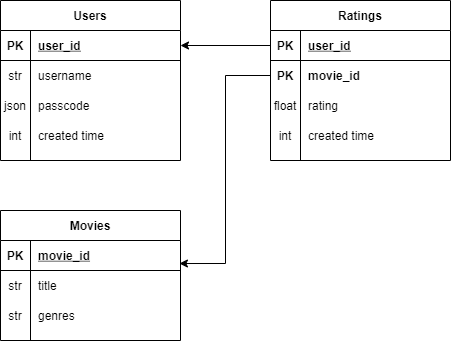


<**Figure 1.3**>

Referring to Figure 1.3 above, there are two \*.csv files and one ML folder in a folder named shared. This folder is shared within two servers. This makes it easy for us to share files. All functions can be used while maintaining the performance of the server.

**Relational Database Management System (RDBMS)**

Currently, this BOJA project has three tables, each named Ratings, Movies, and Users. And this database made by MySQL. The reason why I chose MySQL is because it is an open license, so it can be used free of charge, can handle huge tables quickly, and I chose it with support for various languages. First of all, it is SQL, so it is optimal for configuring RDBMS. Figure 2.1 shown below is a simple schema within my current database.

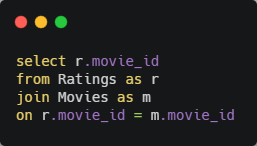


<**Figure 2.1**>

Last semester, mobile games I made only needed user-related data, so I chose NoSQL, which can be expanded faster, but this time they were configured with SQL because of other data (Movies, Ratings) in addition to users' data. In particular, the Ratings table has user\_id and movie\_id at the same time, making it easier and faster to know which movie users rated. With 50k data currently stored in this Movies table and 100k data stored in Ratings' table, fast query retrieval is essential for us. Let me show you how to search the query as an example.

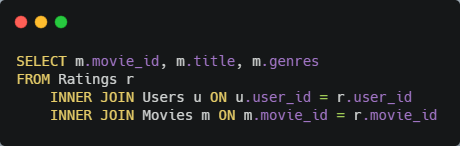
In general, it would be common to extract data from each table and compare it with data from other tables. But it has the disadvantage of being slow. So we're going to use the advantages of SQL and RDBMS to solve everything in a single query.

* Pull all the ratings of a particular movie

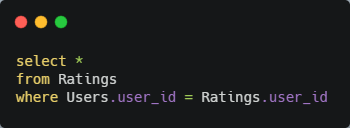


<**Figure 2.2**>

* Get all movie data evaluated by the user



<**Figure 2.3**>



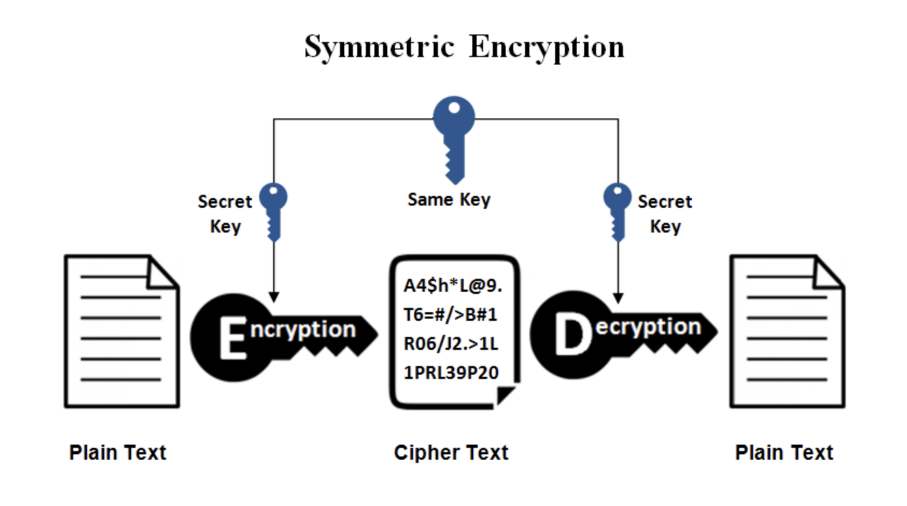
<**Figure 2.4**>

These queries allow for quick searches and simple use. Because of the advantages of RDMBS, it is advantageous to configure RDBMS using SQL when many tables and datas exist.

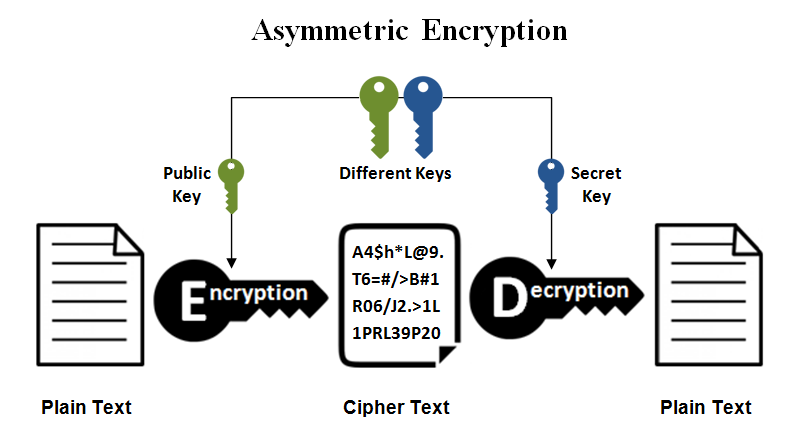
**Encryption of User Data**

It is natural to keep users' data safe. Of course, we should use https communication rather than http communication, but if developers can know all the user data, that will also be a problem. If the user's ID and password are stored in the database, all information will be leaked when the server is hacked. So we should also protect the information in the database.

I have adopted the AES algorithm here, where Hashing generally allows fast access through fast searches, but hackers can also do this, allowing fast data retrieval. The AES method can be seen as a symmetric key method, and although it may be a little less safe than the asymmetric key method, it has the advantage that users can easily decode passwords even if they lose them.

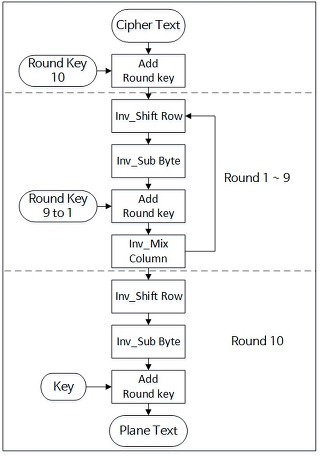
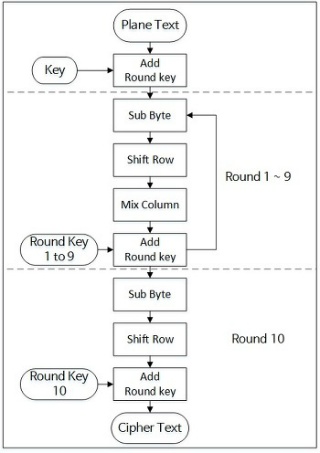


<**Figure 3.1**>



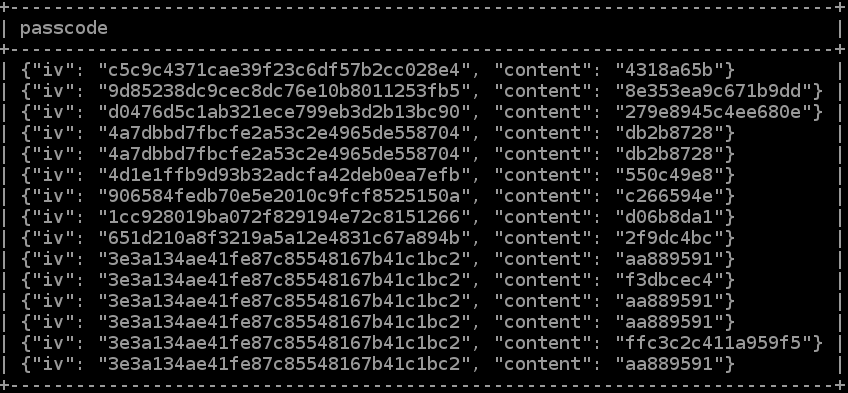
<**Figure 3.2**>

We also used exactly the AES-256, which is the most complex algorithm in the AES method, which is excellent for security. But this has the disadvantage of being a little slow.



<**Figure 3.3**>

Figure 3.3 above illustrates how to encrypt and decode, respectively. Since we used 256bit, we can think of it as a total of 14 rounds. It is also necessary to set the public key for this. However, the public key only shares it with each other, but it is not open to real users. When this process is finished, two values are returned: iv and content are stored in the database in json form. Figure 3.4 below shows how it is actually stored within our database. Any hack can leak that information, but it won't be easy to decode.



<Figure 3.4>

**POST / GET Methods**

1. **/users(POST)**

* This function is for creating new user. When the user signs up, the user enters their username, passcode, and confirmed passcode. Two username and passcode will be sent with encryption by this method.

1. **/users(GET)**

* This function is for getting users data. If the client wants to user’s data, they can get a data only user\_id because of security using this method

1. **/ratings(POST)**

* This function is for posting new rates. When the user evaluates various rates, this function will be called. They will be send with user\_id, movie\_id and rate by each moive.

1. **/ratings(GET)**

* This function is for getting ratings for specific user. When the client want to know all ratings from specific user, this function will help you. The user\_id parameter need.

1. **/signin(POST)**

* This function is for signing in, they will enter their username and passcode which was already registered. This method will be get two parameters, username and passcode.

1. **/likedMovies(GET)**

* This function is for getting all movies that specific user liked. Using this application, the data will be recorded in db. And when the user wants to see their movies that liked before. This method will be help users.

1. **/start(GET)**

* This function is for getting recommend movies. When the user goes to home menu, the user can see 4 canvases with movie data. This method will be get user\_id with parameter.

**Tech Stacks (Back-end)**

* Node.js
* MySQL
* AWS(Ec2, EFS)
* Pm2, Nginx
* Python 3.6.5

**Conclusion**

This project is the first machine learning-based application to challenge, so there were many hardships. Especially, I spent a lot of time on unexpected things. For example, I didn't know that machine learning uses a lot of cpu usage, and because of this, I found that when I used a low-priced server, I couldn't execute two at the same time. NFS was a concept that I didn't even know about, but it was a method that was urgently searched and solved. But I think it's right that these results gave me more knowledge. Also, the encryption method was just hashing last semester, but I found that fast searching is not necessarily good. I know that fast searching can be good for the backend work, but it is also good for hacking. Because the hacker can search our data with fast searching. so I tried other algorithms in a hurry, and this was a pretty tough task. I also spent a lot of time making good query statements to prevent multiple searches within the database. Because duplicate searches also produce many brackets, good query statements have also helped to write readable code.

**Reference**

[**https://dailyworker.github.io/AES-Algorithm-and-Chiper-mode/**](https://dailyworker.github.io/AES-Algorithm-and-Chiper-mode/)

[**https://www.w3schools.com/nodejs/nodejs\_mysql.asp**](https://www.w3schools.com/nodejs/nodejs_mysql.asp)

[**https://aws.amazon.com/efs/**](https://aws.amazon.com/efs/)

[**https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc**](https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc)

[**https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-2.html**](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-2.html)

[**https://en.wikipedia.org/wiki/Relational\_database**](https://en.wikipedia.org/wiki/Relational_database)

[**https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm**](https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm)

[**https://stackoverflow.com/questions/47720876/encrypting-in-node-js-using-aes-256-ctr-and-decrypting-in-golang**](https://stackoverflow.com/questions/47720876/encrypting-in-node-js-using-aes-256-ctr-and-decrypting-in-golang)

[**https://gist.github.com/chris-rock/993d8a22c7138d1f0d2e**](https://gist.github.com/chris-rock/993d8a22c7138d1f0d2e)