Distributed File System

Ryan Kane – 1332 5208 – Computer Science and Business
https://github.com/kaner94/distroDFS

INTRODUCTION

In this report I will be detailing how I completed the assignment to create a Distributed File System in Haskell. We were required to split the system into a number of sub-systems. As such, for each sub-system I will explain my idea for how the system would work, how I implemented the system and how the system works. If I feel any system can be improved I will also include that information.-

The system is made up of the following sub-systems:

1. Authentication System
2. File System
3. Caching System
4. Directory System

In the following pages you will find these sub-systems described as explained previously.

AUTHENTICATION SYSTEM

I intended for the Authentication System to provide a wide range of services. Firstly, it would allow users to Sign Up to the system. It would encrypt passwords for new users, as well as decrypt passwords and other data when necessary. The Authentication System would be used to set a token for use in the encryption and decryption. Users would also be able to request a Token from the system.

Add User

The user is most likely going to first use this system when they wish to sign up. This occurs by using the ‘addUser’ method. The user will pass a ‘name’ and ‘password’ when calling this method. These will both be used in the addUser method to create an instance of the ‘User’ datatype. This instance of User will then be posted to the MongoDB by calling the ‘postUser’ method. The password will be encrypted before it is posted to the database. All data must be converted to BSON before it is posted to the MongoDB to ensure it works correctly.

Encryption and Decryption

The encryption in this system is based off a simple Caesar cipher. The encryption function relies on the Key. This is simply a datatype that contains an integer which is determined by the Authentication system. To encrypt a string, the system uses the ‘Map’ function combined with ‘Ord’ to convert each character in the string to a hexadecimal value. ‘Map’ is then used again to augment each hexadecimal value by the predetermined value of the Key. The Map function is used one last time with the ‘Chr’ function to convert the array of hexadecimal values back into characters.

Decryption is performed by completing very similar steps. The encrypted string is converted to hexadecimal values, the Key is reapplied to the values and then the hexadecimal is converted back into a string.

Get Token

Before a user can access files, they are required to retrieve a token from the Authentication system. This is performed by using the ‘getToken’ method.

When making this call, the user is required to pass a Username and Password. The system will check the MongoDB to ensure that the username is in the system and the password does indeed match. If this check is successful, a token will be returned to the user. Otherwise, this request will fail.

FILE SYSTEM

The purpose of the File System is to allow the user to store files in the database, as well as download these files to store locally.

Post File

This is the primary function of the system. The user calls the ‘postFile’ method and passes the contents of the file as a parameter. The system will encrypt the contents of the file using the aforementioned Caesar cipher and store this in the MongoDB.

I also attempted to post a local file to the database. I was successfully able to read the contents of the locally stored file and operate on that data, however I was unable to store it in the database.

GetFile

I attempted to allow users to access the MongoDB and save the contents of a DB entry to a local file. Unfortunately I was unable to get this working.

COMMANDS

The File System, as previously mentioned, is split into multiple subsystems. To run each system, run the following command:

stack exec systemDFS-exe

where ‘system’ is the system that you are currently in (ie) cache, file, auth.

To Create User:

curl -X POST -d '{"name": "JoeBloggs", "password": "sample"}' -H 'Accept: application/json' -H 'Content-type: application/json' http://localhost:8080/addUser

To Get Token:

curl -X POST -d '{"name": "JoeBloggs", "password": "sample"}' -H 'Accept: application/json' -H 'Content-type: application/json' http://localhost:8080/getToken

To Insert File:

curl -X POST -d '{"fileContents": "Contents to add to DB"}' -H 'Accept: application/json' -H 'Content-type: application/json' http://localhost:8080/postFile