**Network System Development**

**5CM510**

**Task Management System**

**Implementation Log**

**Student Number**

100723654

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, διάγραμμα, Σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματαServer Implementation

Phase 1

Organized the project, created a plan for all the classes and files. Created a data flow diagram. Modified it throughout development as needed.

Server Implementation

In the serverlib.py

* Started the server development with the ServerLib, a library for the server that contains the functionality for receiving and sending data. The file contains a class ConnectionHandler which contains all functionality.
* Created input and output thread safe queue buffers to receive and send data.
* Created two functions read and write as well as two threads readThread and writeThread that target each function respectively.

In the statemachine.py

* Added a enum class for states:
  + Start
  + LoggingIn
  + SignUp
  + Dashboard
  + AdminDashboard
  + Exit
* Created the standard protocol for message format communication.

JSON

DATA FORMAT:

{‘action’ : ‘login’,

‘message’: ‘SOMETHING’

‘username’ : ‘username' ,

‘password’:’password’}

When signing in or signing up the server will expect a dictionary with keys as action, username and password. When communicating in dashboard, it will expect ‘action’ ‘message’. Will use JSON format for transmitting the data.

* Added sign in and sign out functionality as well as authentication for user, admin, or wrong credentials.

Phase 2

In the serverlib.py

* Implemented packet header message verification. Added a 4 byte header, a network buffer for message accumulation and a messageInProgress bool to check message state.
* A message is sent with a 4 byte header that contains how many bytes the receiver should expect, example : ‘0011Helloworld’ 00 -11 means that the receiver reads the next 11 bytes to get the completed message.
* The receiver reads the 4 bytes into networkBuffer, convert the header to int to get the message length and accumulates the message bytes until the length is reached.

**Created encryption file and Server logger file.**

In the Encryption.py

* Implemented RSA encryption for communication
* md5 hashing for passwords
* Caesar cipher for usernames

In ServerLogger.py

* Created the logger.
* Set messages to be saved in server\_log.log
* Display the logs in the terminal as well

In the ServerLib.py

Added error handling due to errors during development.

* Added connection error handling
* Added socket timeouts
* Added graceful disconnections from clients
* Added buffer overflow protection

Added functions to get messages from the iBuffer and push messages into the oBuffer.

Encryption and decryption functions were added to automatically encrypt message before sending to clients and automatically decrypt messages while getting messages from the clients (iBuffer).

Early stages of encryption used RSA for all communication except username which used Caesar Cipher.

Phase 3

In the StateMachine.py

* Added dashboard functionality (CRUD operations)
* Added admin dashboard functionality
* Created functions to support these operations

Created new file Server.py, Database.py, Authentication.py

In the Server.py

* Created a Server class
* Implemented a “listen” function that listens for clients
* Created a listen thread
* Implemented RSA key exchange protocol
* Initialization of objects state machines, rsa keys, connection handlers.
* Added a function to call the state machine when a message is received by iBuffer

In the Database.py

* Created the database
* Created 2 tables, Users and Tasks
* Implemented functions for CRUD operations
* Implemented functions to add and view users.

In the authentication.py

* Gets the client’s credentials and matches them with the database

Phase 4

Had errors with RSA “plain text too long”.

In the encryption.py

* Implemented AES-GCM
* Now messages changed format to ciphertext, aesIV, authtag inside the ciphertext is all the message (action, message, username, password keys and their values)

In the server.py

* Modified the RSA key exchange to RSA keys then encrypt the AES key with the clients RSA public key and send it to client.
* Implemented validation for message length and public key and AES key

Finished the server having

Server.py

Serverlib.py

StateMachine.py

Encryption.py

ServerLogger.py

Authentication.py

Database.py

Tested the functionality of the server.

Fixed errors and bugs

**Client Implementation**

Created Client.py, ClientLib.py, ClientEncryption.py

Copy pasted most of the code from server to the client files. Made necessary changes in Client.py for the client to connect to the server and not listen for clients. ClientLib and ClientEncryption are almost identical to the server’s ServerLib and Encryption.

Created the ClientStateMachine.py

* Created states based on server’s states
* Created login/ signup functionality
* Created dashboard/admin dashboard functionality
* Implemented function for CRUD operations – Send the correct message to the StateMachine of the server.

Created the GUI.py

* Created login ui, signup ui and dashboard ui.
* Implemented CRUD operation functionality – Popups for Create/Edit task
* Implemented dynamic task viewing/ each task is encapsulated in a frame and creates 2 buttons edit/delete.
* Implemented Refresh functionality

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, ρολόι, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα**UI inspirations**

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει στιγμιότυπο οθόνης, ακουστικά, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματα

Near the end of the development, AES was added for stronger security and to fix errors of RSA such as “Plain text too long for encryption”.