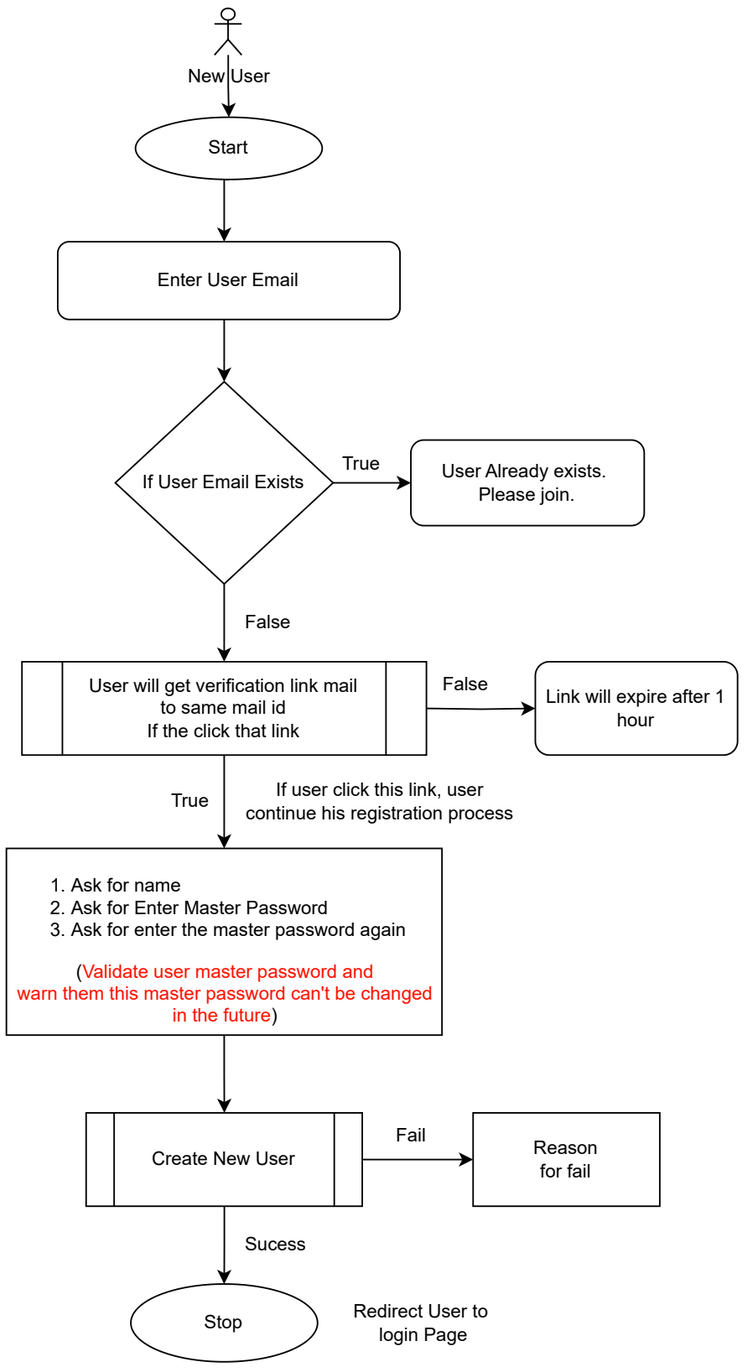
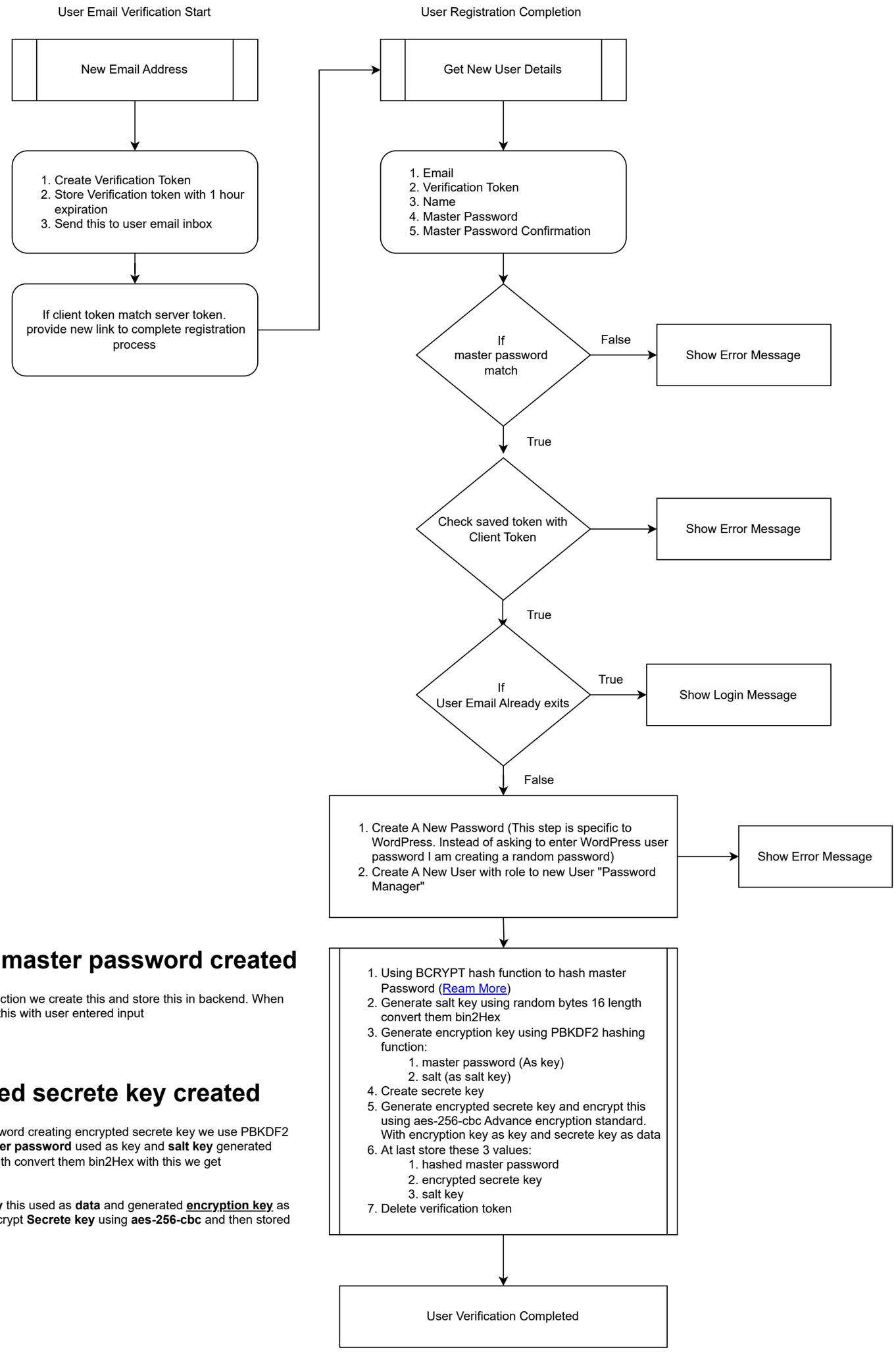
Registration

This diagram show how user flow in frontend



Registration Process at backend

This diagram show user flow in server and database



How BCRYPT works

1. Salting: A random salt is generated and added to the password.
2. Key Stretching: The combined password and salt are subjected to a computationally expensive process, involving multiple iterations determined by the work factor. This significantly increases the time required to crack the password.
3. Hash Generation: The result of the key stretching process is used to generate a fixed-length hash.
4. A typical Bcrypt hash looks like this:$2a$10$vI1UF0Nt3JAuhVo0O2QzpuF6j3LUdAu9u01L9qLJ6519rHt00j4u.

How PBKDF2 works

1. Salting: A unique salt is generated for each password. This prevents rainbow table attacks.
2. Iteration: The password and salt are combined and fed into the PRF (often HMAC). This process is repeated a specified number of times (iteration count).
3. Key derivation: The output of the final iteration is used to derive the key.

How AES-256-CBC Works

1. Initialization Vector (IV): A random value is generated to initialize the encryption process.
2. Block Division: The plaintext is divided into blocks of a fixed size (usually 128 bits for AES).
3. XOR with Previous Ciphertext: The first plaintext block is XORed with the IV. Subsequent blocks are XORed with the previous ciphertext block.
4. AES Encryption: Each XORed block is encrypted using the 256-bit AES key.
5. Output: The encrypted blocks form the ciphertext.

How hashed master password created

By using BCRYPT hash function we create this and store this in backend. When user login we will compare this with user entered input

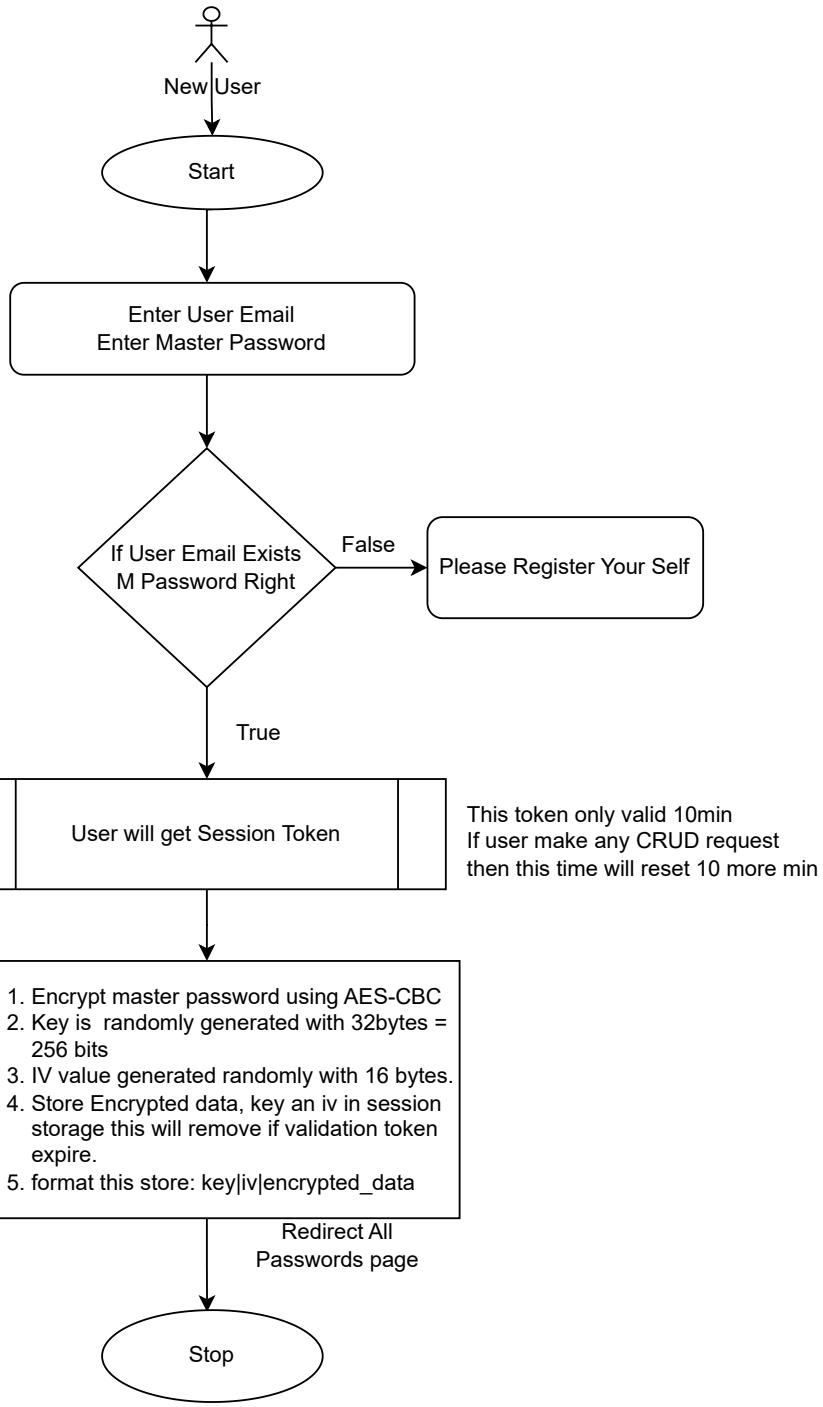
How encrypted secrete key created

Unlike hashed master password creating encrypted secrete key we use PBKDF2 hashing function here **master password** used as key and **salt key** generated using random bytes 16 length convert them bin2Hex with this we get **"encryption key"**

Now we create **Secrete key** this used as **data** and generated **encryption key** as key with these value we encrypt **Secrete key** using **aes-256-cbc** and then stored in backend.

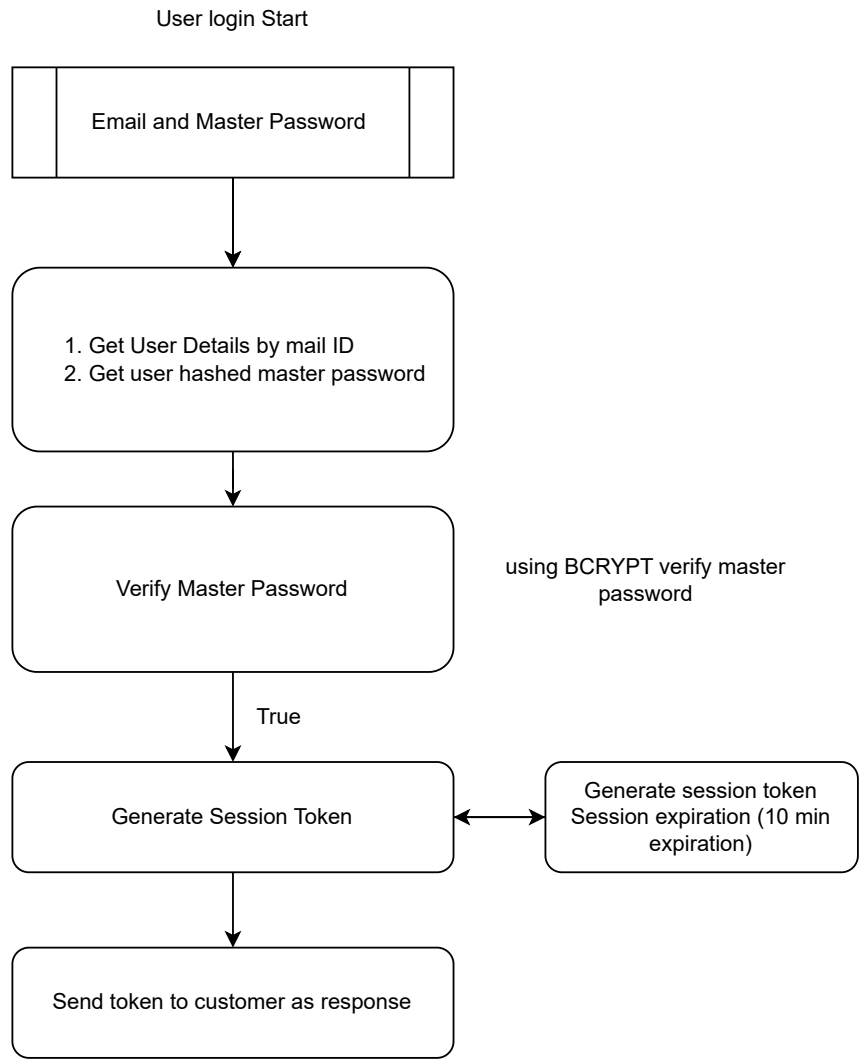
Login

This diagram show how user flow
in frontend



Login Process at backend

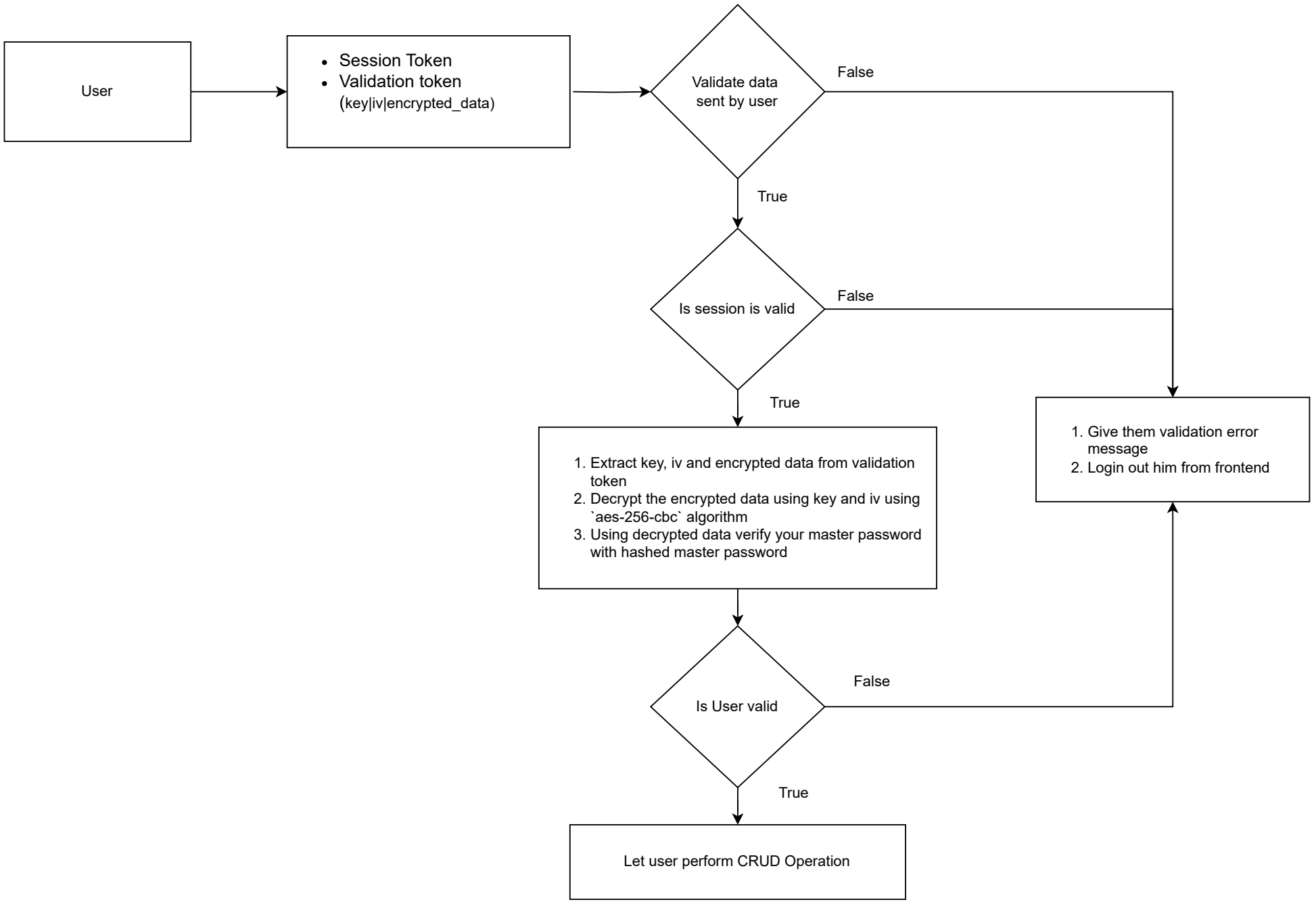
This diagram show user flow in server and database

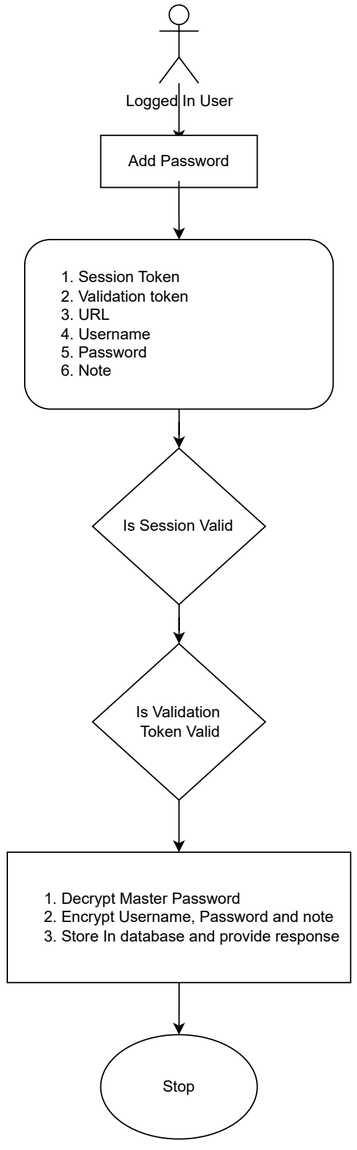


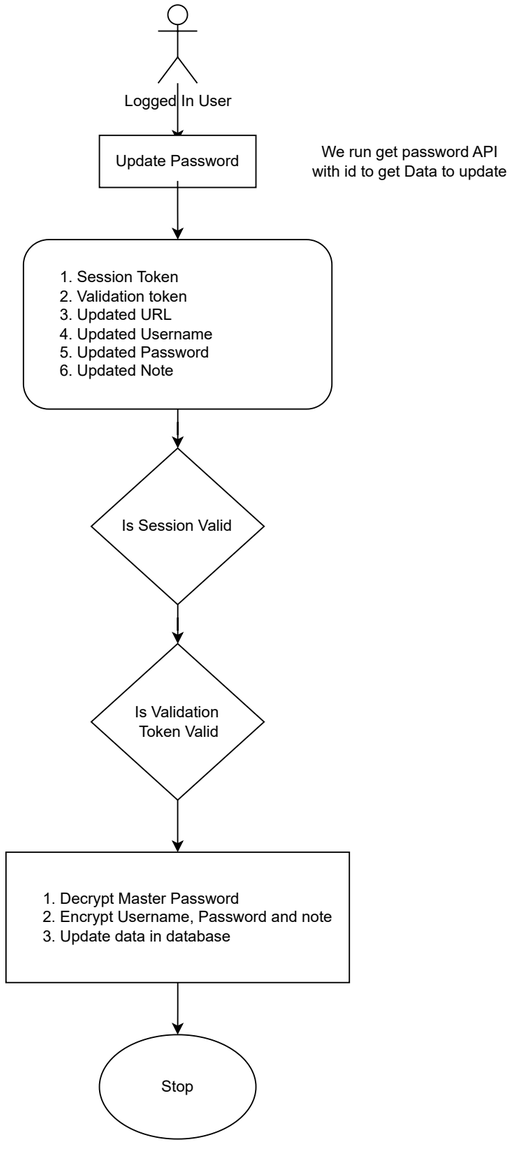
User Validation Process

If user do any operation in Password Manager we have validation use before doing any data manipulation

Validation token generated in frontend using `aes-cbc` algorithm this will sent in body or header on every API hits







We run get password API with id to get Data to update