INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

CS4.409

Data Foundation Systems

Project Report: UDP Peer to Peer Secure Data Transfer

Submitted By:

Soumodipta Bose (2021201086)

Vishal Pawar (2021201025)

Mukul Kumar (2021201064)

GitHub Link: https://github.com/soumodiptab/Blogger/

Requirement:

Peer-to-peer data transfer is a technique used to transfer data from one computer to another without any intermediary server or a central coordinator. Secure peer to peer data transfer is crucial when the data being transferred is sensitive or confidential.

In this requirement document, we will discuss the architecture and design of a secure high-speed peer to peer data transfer system that uses UDP for data transfer and TCP for transfer quality control. The data transfer system will be built using NodeJS servers, and it will facilitate data transfer between two agents, the Data Foundation Agent, and the Partner Agent.

Data Foundation Server

The Data Foundation Server is the central server that stores and manages the data being transferred. The server has the following agent details:

- **Partner ID**: A unique identifier assigned to each partner to facilitate easy identification.
- **Partner Public Key:** A cryptographic key used to encrypt data sent to the partner.
- **Partner Private Key**: A cryptographic key used to decrypt data received from the partner.
- **Data Send/Receive Directory**: A directory where data is stored before and after transfer.
- **Host Name:** The name of the server hosting the Data Foundation Agent.
- **IP Address:** The IP address of the server hosting the Data Foundation Agent.
- **Port Details:** The port number that the Data Foundation Agent uses for communication.

Partner Agent

The Partner Agent is the agent that receives data from the Data Foundation Server. The partner agent has the following details:

- **Partner ID:** A unique identifier assigned to each partner to facilitate easy identification.
- **Partner Public Key:** A cryptographic key used to encrypt data sent to the partner.
- **Partner Private Key:** A cryptographic key used to decrypt data received from the partner.
- **Data Send/Receive Directory:** A directory where data is stored before and after transfer.
- **Host Name:** The name of the server hosting the Partner Agent.
- **IP Address:** The IP address of the server hosting the Partner Agent.
- **Port Details:** The port number that the Partner Agent uses for communication.

Data Transfer Request Setup

To initiate a data transfer, the sender and the receiver must have a data transfer profile that contains the following information:

- **Sender DT Profile:** The sender's data transfer profile contains the sender's details, including the Partner ID, Partner Public Key, Partner Private Key, Data Send/Receive Directory, Host Name, IP Address, and Port Details.

- **Receiver DT Profile:** The receiver's data transfer profile contains the receiver's details, including the Partner ID, Partner Public Key, Partner Private Key, Data Send/Receive Directory, Host Name, IP Address, and Port Details.
- **Transfer Configuration:** The transfer configuration includes the number of threads, IP addresses, ports, schedule time, number of files, total file size, and MD key.

Add Files

After the data transfer profiles and the transfer configuration have been set up, the sender can add files to the data send directory. These files will be transferred to the receiver during the data transfer session.

Schedule/Initiate Data Transfer Request

The sender can schedule or initiate the data transfer request. The schedule time is the time at which the data transfer should begin. The sender can also specify the number of files and the total file size to be transferred.

DT Profile Key Verification

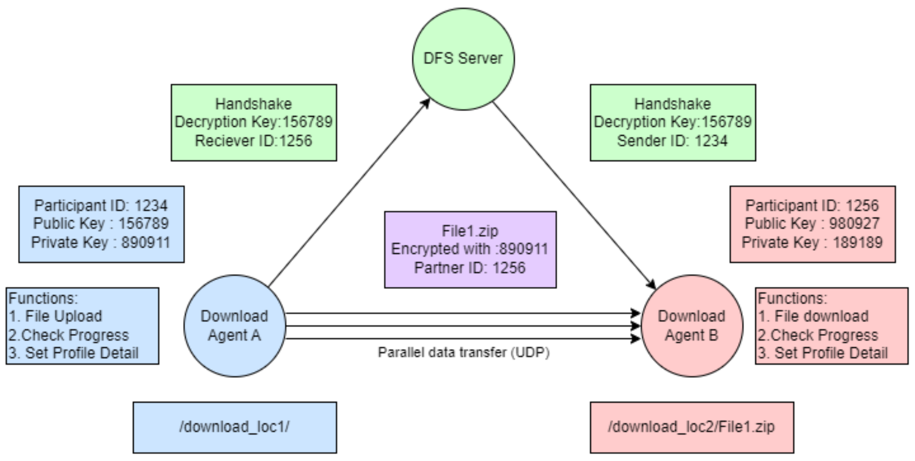
The Data Foundation Server verifies the sender and receiver's data transfer profile keys to ensure that the sender and receiver are authorized to access the data transfer session. If the verification is successful, the data transfer session can begin.

Initiate Data Transfer (Send/Receive)

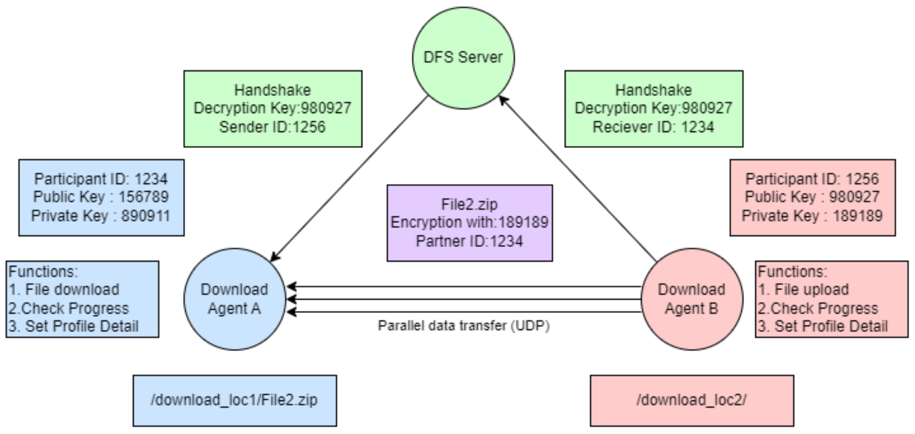
The data transfer session begins with the sender sending the data to the receiver using the UDP protocol.

Design:

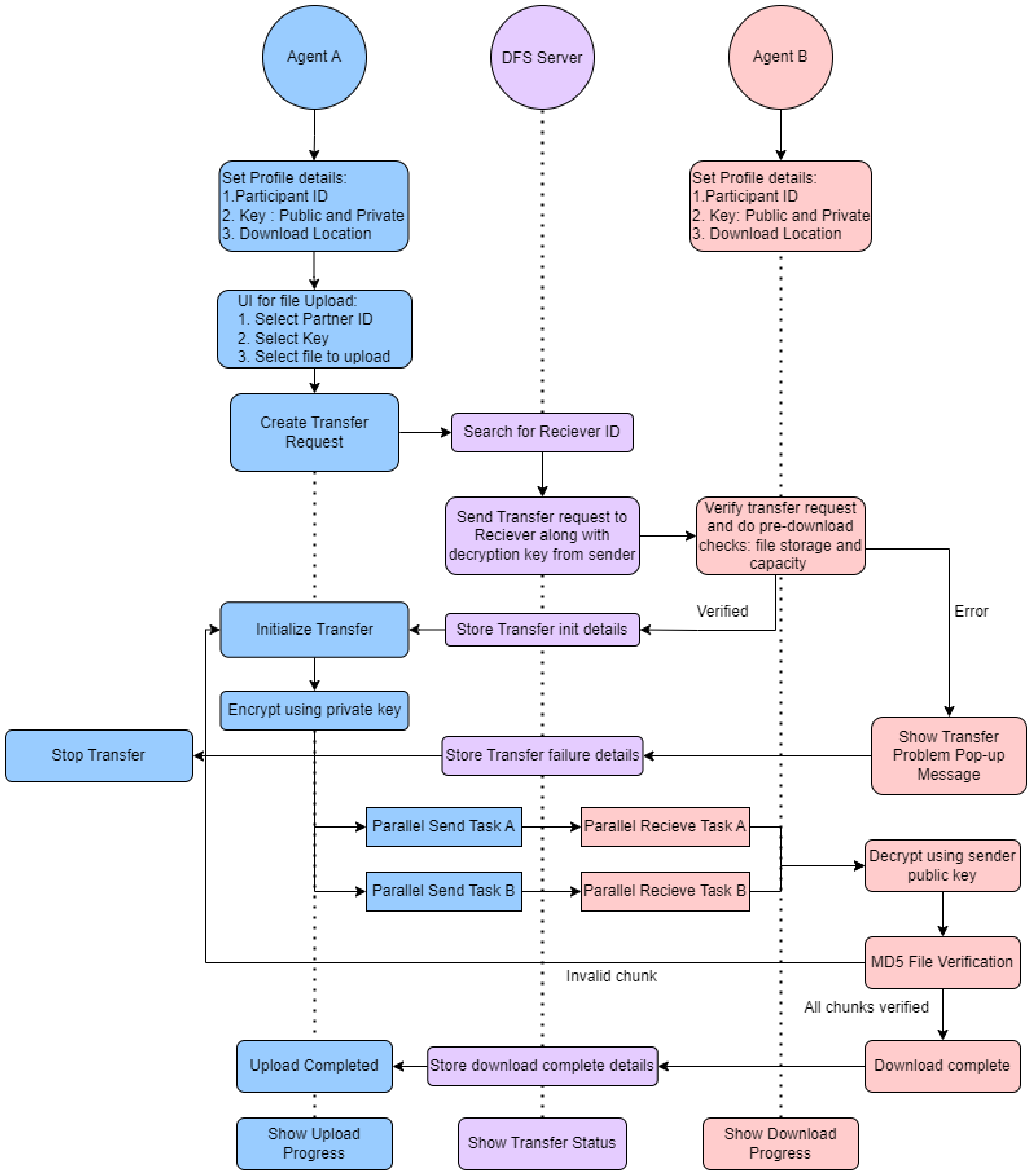
System Design : UDP Peer to Peer Secure Data Transfer



Transferring file1.zip from Agent A to Agent B



Transferring file2.zip from Agent B to Agent A


Agent A
DFS Server
Agent B
Set Profile details: 1.Participant ID 2. Key : Public and Private 3. Download Location
Set Profile details: 1.Participant ID 2. Key: Public and Private 3. Download Location
UI for file Upload: 1. Select Partner ID 2. Select Key 3. Select file to upload
Create Transfer Request
Search for Reciever ID
Send Transfer request to Reciever along with decryption key from sender
Verify transfer request and do pre-download checks: file storage and capacity
Verified
Error
Initialize Transfer
Store Transfer init details
Encrypt using private key
Stop Transfer
Store Transfer failure details
Show Transfer Problem Pop-up Message
Parallel Send Task A
Parallel Recieve Task A
Parallel Send Task B
Parallel Recieve Task B
Decrypt using sender public key
MD5 File Verification
Invalid chunk
All chunks verified
Upload Completed
Store download complete details
Download complete
Show Upload Progress
Show Transfer Status
Show Download Progress

Workflow of the System

Central Server (DFS server):

The Data Foundation Server (DFS) serves as a mediator between the sender and receiver during file transfers. It is responsible for managing the data being transferred and ensuring the security and integrity of the data. The DFS server maintains a list of institutes containing their peer ID, IP address, and secret key.

Institute List

Upon successful login, the sender and receiver can retrieve the institute list, which will be encrypted and can be decrypted using their respective secret keys. This list will contain all the institutes that have been registered with the DFS server. The sender and receiver can choose the institute to which they want to transfer the data.

Record of Activities

The DFS server will keep a record of all activities, including file transfers and requests. This record will be used for auditing purposes and to ensure that all transfers are completed successfully. The server will also maintain logs of any errors that occur during the transfer process.

Ongoing Transfer Monitoring

The DFS server will monitor ongoing transfers to ensure that they are completed successfully. The server will keep track of the progress of each transfer and will notify the sender and receiver if any issues arise during the transfer. In the event of a failed transfer, the server will provide error logs to help diagnose the problem.

Security and Integrity

The DFS server plays a critical role in ensuring the security and integrity of the data being transferred. The server uses encryption algorithms to secure the data during transfer and stores the data in an encrypted form to protect it from unauthorized access.

The DFS server uses the sender and receiver's secret keys to encrypt and decrypt data. This ensures that only the intended parties can access the data. The server also uses cryptographic hash functions to ensure the integrity of the data being transferred. Hash functions are used to generate a unique fingerprint of the data, which is then used to verify the data's authenticity.

Conclusion

The DFS server serves as a mediator between the sender and receiver during file transfers. It ensures the security and integrity of the data being transferred and keeps a record of all activities. The institute list maintained by the server allows the sender and receiver to choose the institute to which they want to transfer the data. The server also monitors ongoing transfers to ensure that they are completed successfully and provides error logs to help diagnose any problems. By using encryption algorithms and cryptographic hash functions, the server ensures the security and integrity of the data being transferred.

Sender:

Upon successful login, the sender can fetch the institute list from the DFS server. This list contains the peer ID, IP address, and secret key of all the institutes registered with the DFS server. The sender can initiate the transfer by selecting the receiver's ID from the list.

Once the receiver is selected, the sender can upload the file to the DFS server. The basic information about the file, such as UUID, filename, and file size, is sent to the DFS server. The server then forwards the request to the receiver through the DFS server. The receiver will receive a notification that a file transfer request has been initiated.

Parallel Chunk Transfer

The transfer occurs in parallel chunks to reduce the time taken for the transfer. The sender divides the file into smaller chunks, and each chunk is transmitted individually to the receiver. This process ensures that the file transfer is faster and more efficient.

Encryption of Data

The data is encrypted before transmission to ensure the security and privacy of the data being transferred. The encryption process is performed by the sender before the data is sent to the DFS server. The sender uses the receiver's public key to encrypt the data. Only the receiver can decrypt the data using their private key.

Monitoring and Tracking Downloads

The sender can monitor and keep track of all downloads. The sender can view the progress of the file transfer and see how much data has been transferred. The sender can also view the status of the download, whether it has been completed or not.

Conclusion

The file transfer process between the sender and receiver involves fetching the institute list from the DFS server, initiating the transfer by selecting the receiver's ID, uploading the file to the DFS server, forwarding the request to the receiver through the DFS server, and monitoring and tracking the download progress. The transfer occurs in parallel chunks to reduce the time taken for the transfer. The data is encrypted before transmission to ensure the security and privacy of the data being transferred. The sender can view the progress of the file transfer and see how much data has been transferred. The file transfer process is efficient and secure, and the sender can be assured that their data is safe during the transfer process.

Receiver:

Once the receiver logs in successfully, they can fetch the institute list from the DFS server. This list contains the peer ID, IP address, and secret key of all the institutes registered with the DFS server.

When a sender initiates a transfer request, the receiver will receive a notification through the DFS server. The notification will contain basic information about the file, such as UUID, filename, file size, and sender's information.

Accepting or Rejecting Transfer Request

The receiver can choose to accept or reject the transfer request. If the receiver accepts the transfer request, their response will be sent back to the DFS server, and the file transfer will start. If the receiver rejects the transfer request, the sender will be notified, and the transfer request will be cancelled.

Starting File Transfer

If the receiver accepts the transfer request, the file transfer will start. The data will be transferred in parallel chunks to reduce the time taken for the transfer. The data will also be encrypted before sending, ensuring the security and privacy of the data being transferred. The sender uses the receiver's public key to encrypt the data, and only the receiver can decrypt the data using their private key.

Monitoring and Tracking Downloads

The receiver can monitor and keep track of all downloads happening. The receiver can view the progress of the file transfer and see how much data has been transferred. The receiver can also view the status of the download, whether it has been completed or not.

Decrypting and Merging Data

After receiving all the chunks of data, the receiver will decrypt and merge the data. The data will be decrypted using the receiver's private key, and the chunks will be merged to form the original file. Once the file is complete, the receiver will be notified that the transfer is complete.

Conclusion

Receiving files through the DFS server involves fetching the institute list from the DFS server, receiving transfer requests containing basic information about the file, accepting, or rejecting transfer requests, starting file transfers, monitoring, and tracking downloads, decrypting, and merging data, and receiving notifications once the transfer is complete. The data is transferred in parallel chunks, and the data is encrypted before sending to ensure the security and privacy of the data being transferred. The receiver can be assured that their data is safe during the transfer process.

Data Transfer:

The transfer process begins once the receiver accepts the request to download the file or folder. The sender splits the selected file or folder into several smaller chunks for easier transfer. The sender then encrypts each chunk before sending to add an extra layer of security.

Parallel Transfer

To ensure a quicker transfer process, the sender transfers the chunks in parallel. This means that multiple chunks are transferred simultaneously, reducing the time taken to complete the transfer. On the receiver's side, multiple chunks are received in parallel to speed up the download process.

Decrypting Chunks

Each chunk received by the receiver is encrypted, and it needs to be decrypted before it can be merged back into the original file or folder. The receiver decrypts each chunk using the appropriate key. The decryption process ensures that the data transferred is secure and cannot be accessed by unauthorized parties.

Merging Chunks

Once all chunks have been successfully decrypted, they will be merged back into the original file or folder. The receiver's device will check each chunk to ensure that it has not been tampered with during the transfer process. Any corrupted chunks will be retransmitted by the sender.

Tracking Transfer Progress

During the transfer process, both the sender and receiver can track the progress of the transfer. The sender can see which chunks have been successfully transferred, and which ones are pending. The receiver can monitor the progress of the download, seeing which chunks have been received and decrypted successfully.

Conclusion

In conclusion, the transfer of files or folders between the sender and receiver involves splitting the file or folder into smaller chunks, encrypting each chunk, and transferring them in parallel. On the receiver's side, the chunks are decrypted and merged into the original file or folder. The transfer process can be monitored by both the sender and receiver to ensure its successful completion. The use of parallel transfer and encryption ensures a quick and secure transfer process, ensuring the safety and security of the data being transferred.

Data Encryption:

Data encryption plays a critical role in ensuring the security of information during transfer. By encrypting data, we can prevent unauthorized access and ensure that only the intended recipient can access the data. In the context of peer-to-peer data transfer using UDP and TCP protocols, encryption is particularly important to ensure that sensitive data is protected during the transfer process.

To ensure data security, each file or folder selected for transfer is divided into smaller chunks, which are easier to transmit and receive. Each of these chunks is encrypted using a secret key that is known only to the sender and receiver. The secret key is a unique string of characters used to encrypt and decrypt data. It is important to ensure that the secret key is strong and kept confidential, as any breach in the key's confidentiality can lead to the decryption of sensitive data by unauthorized parties.

The encryption process involves applying a mathematical algorithm to the original data, resulting in an encrypted version of the data. The algorithm used to encrypt the data is designed in such a way that it is impossible to reverse engineer and recover the original data without the key. The encrypted data is then transmitted over the network using the UDP protocol.

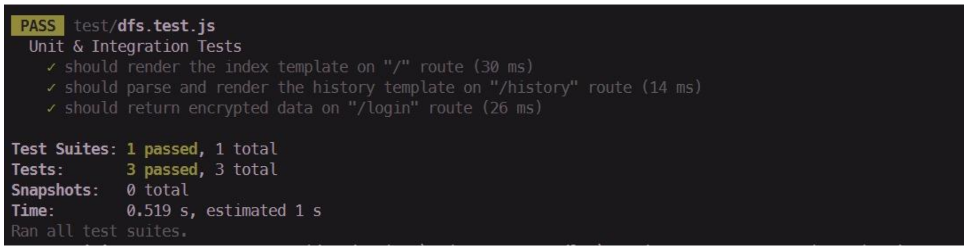
Once the receiver receives the encrypted chunks, they are decrypted using the same secret key that was used to encrypt the data. Decryption involves applying a reverse mathematical algorithm to the encrypted data, which returns the original data. It is important to ensure that the decryption process is performed accurately, as any errors can lead to corrupt data. The decrypted chunks are then merged back into the original file or folder, allowing the receiver to access the transferred data.

The use of encryption during the transfer process provides several benefits. Firstly, it ensures data confidentiality, as encrypted data can only be accessed using the secret key. This prevents unauthorized parties from accessing sensitive information during the transfer process. Secondly, encryption provides data integrity, as any tampering with the encrypted data will result in an error during decryption, ensuring that the data remains unaltered during the transfer process. Finally, encryption provides authentication, as only the sender and receiver possessing the secret key can access the data.

In conclusion, encryption plays a crucial role in ensuring the security of data during the transfer process in peer-to-peer data transfer using UDP and TCP protocols. By dividing the data into smaller chunks and encrypting each chunk using a secret key known only to the sender and receiver, we can ensure data confidentiality, integrity, and authentication. As such, encryption should be an essential consideration in any data transfer process to protect against unauthorized access and data breaches.

Unit and Integration Testing

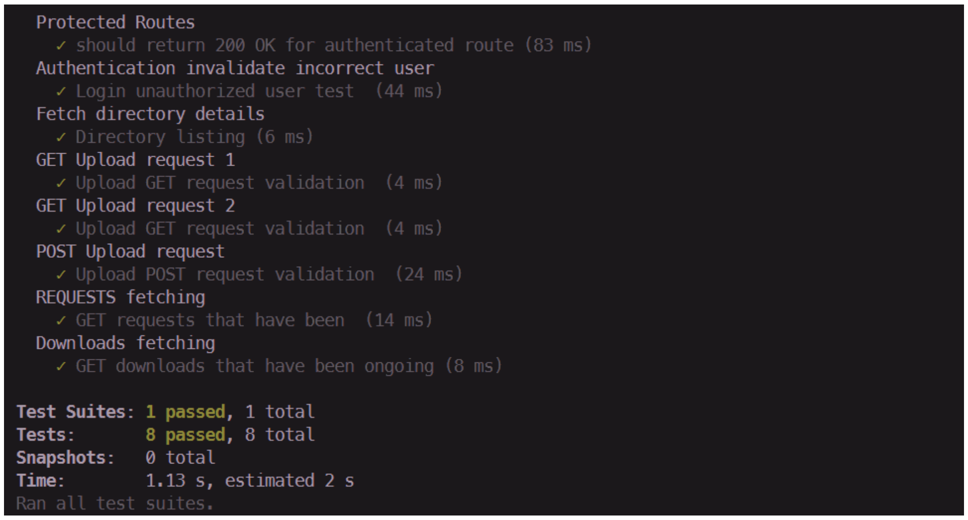
DFS Server Testing



```
 PASS  test/dfs.test.js
  Unit & Integration Tests
    ✓ should render the index template on "/" route (30 ms)
    ✓ should parse and render the history template on "/history" route (14 ms)
    ✓ should return encrypted data on "/login" route (26 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.519 s, estimated 1 s
Ran all test suites.
```

Downloader Testing



```
   Protected Routes
     ✓ should return 200 OK for authenticated route (83 ms)
   Authentication invalidate incorrect user
     ✓ Login unauthorized user test  (44 ms)
   Fetch directory details
     ✓ Directory listing (6 ms)
   GET Upload request 1
     ✓ Upload GET request validation  (4 ms)
   GET Upload request 2
     ✓ Upload GET request validation  (4 ms)
   POST Upload request
     ✓ Upload POST request validation  (24 ms)
   REQUESTS fetching
     ✓ GET requests that have been  (14 ms)
   Downloads fetching
     ✓ GET downloads that have been ongoing (8 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        1.13 s, estimated 2 s
Ran all test suites.
```

Code Traceability

Code traceability is an essential aspect of software development projects, ensuring transparency, maintainability, and accountability. In the context of the UDP Peer-to-Peer Secure Data Transfer project, code traceability plays a crucial role in documenting and understanding the system's implementation. The following sections outline the key areas where code traceability is applied in the project.

1. Code Version Control:

Using a version control system, such as Git, enables effective code traceability. Each code change, including bug fixes, feature enhancements, and new implementations, should be committed to the repository with meaningful commit messages. By examining the commit history, stakeholders can understand the evolution of the codebase, track modifications, and identify the responsible developers.

2. Detailed Comments and Documentation:

To ensure code traceability, it is essential to incorporate comprehensive comments and documentation within the codebase. Comments should describe the purpose, functionality, and intended behavior of the code sections. Additionally, documentation should be provided for complex algorithms, data structures, and external libraries or dependencies utilized in the project. This documentation aids in understanding the code's design choices and facilitates future modifications or updates.

3. Clear Naming Conventions:

Adopting consistent and meaningful naming conventions for variables, functions, classes, and modules enhances code traceability. Descriptive names that accurately represent the purpose of each component help developers easily comprehend the codebase. Moreover, maintaining a standardized naming convention across the project fosters cohesion and reduces confusion when collaborating with other developers.

4. Modular Code Design:

Applying a modular code design approach promotes code traceability by breaking down the system into smaller, self-contained modules. Each module should have well-defined responsibilities and boundaries, encapsulating related functions and data. By structuring the codebase in this manner, it becomes easier to track and understand the flow of data and operations throughout the system.

5. Comprehensive Testing:

Implementing an extensive test suite, including unit tests, integration tests, and system tests, contributes to code traceability. Tests should cover various scenarios, edge cases, and failure conditions. By maintaining a comprehensive test suite, it becomes possible to ensure that code changes do not introduce regressions and that the system behaves as expected. The test suite should be regularly executed and integrated into a continuous integration and continuous deployment (CI/CD) pipeline for automated testing.

6. Issue Tracking and Bug Reporting:

Utilizing an issue tracking system, such as GitHub Issues, allows for effective tracking of bugs, feature requests, and other code-related issues. When a bug is discovered or an enhancement is requested, it should be documented as an issue in the tracking system. This practice provides traceability by linking the code changes made to address the reported issues. Additionally, it enables stakeholders to monitor the progress of issue resolution and track the history of code modifications.

7. Dependency Management:

Managing project dependencies using a package manager, such as npm for Node.js, aids in code traceability. The project should maintain a well-documented list of dependencies, including their specific versions. This documentation ensures that the system can be reproduced accurately by other developers or in different environments.

By incorporating these code traceability practices into the UDP Peer-to-Peer Secure Data Transfer project, the development team can maintain a transparent and comprehensible codebase. This fosters collaboration, facilitates maintenance and debugging, and supports future enhancements or modifications to the system.