# Use case diagram

Use case diagrams are a common way to communicate the major functions of a software system. A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

Use cases are nothing but the system functionalities written in an organized manner. Now another thing which is relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

So in brief, the purposes of use case diagrams can be as follows:
• Used to gather requirements of a system.
• Used to get an outside view of a system.
• Identify external and internal factors influencing the system.
• Showing the interacting among the requirements are actors.

## Symbols used in Use Case diagram:

| | | | |
|---|---|---|---|
| ⬭ | Use case | 웃 | Actor |
| — | Association | ▯ | System |
| «I» - - -> | Include | «E» <- - - | Extends |
| - - -> | Dependency | ◁— | Generalization |

## Use Case Diagram:

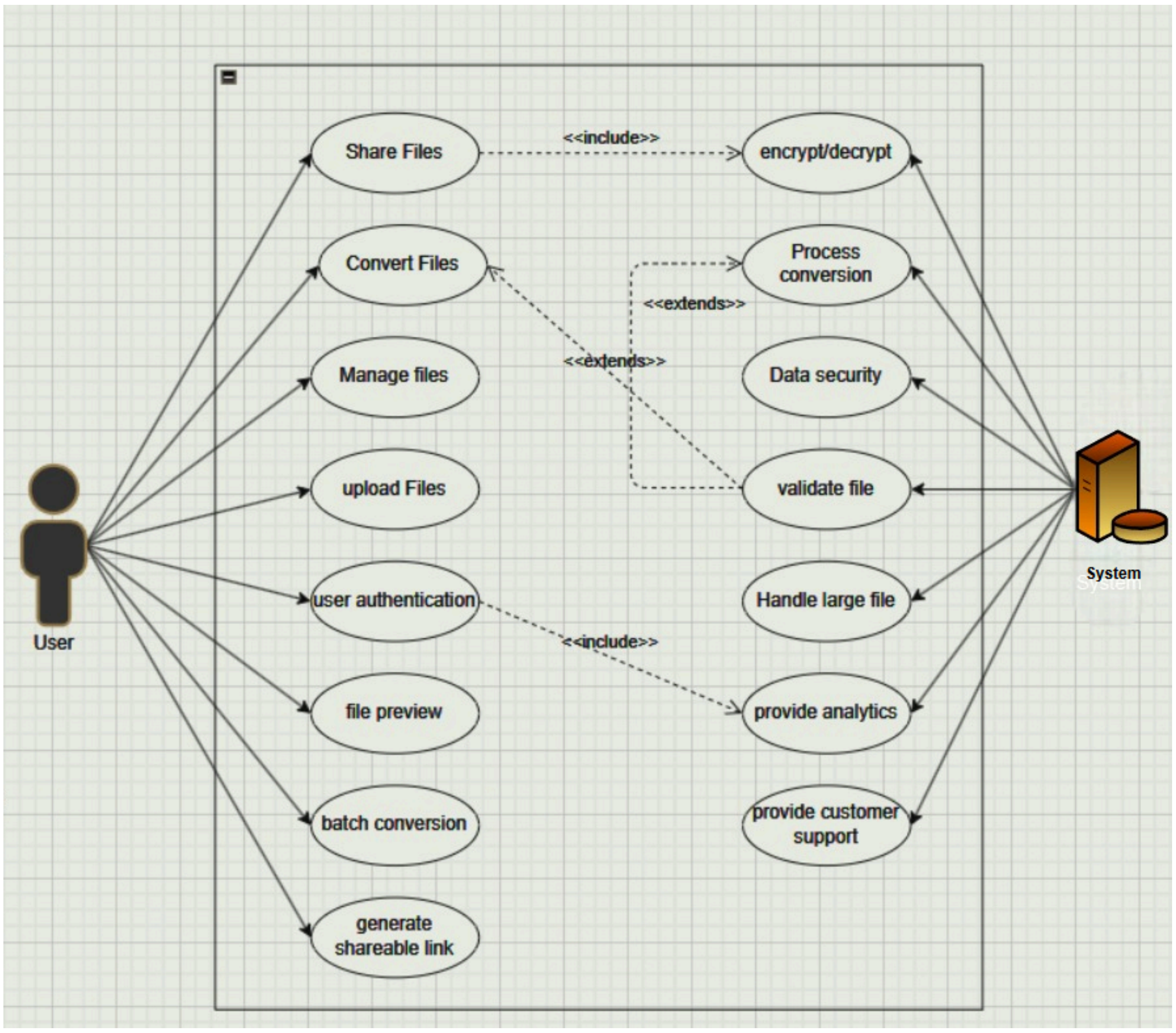
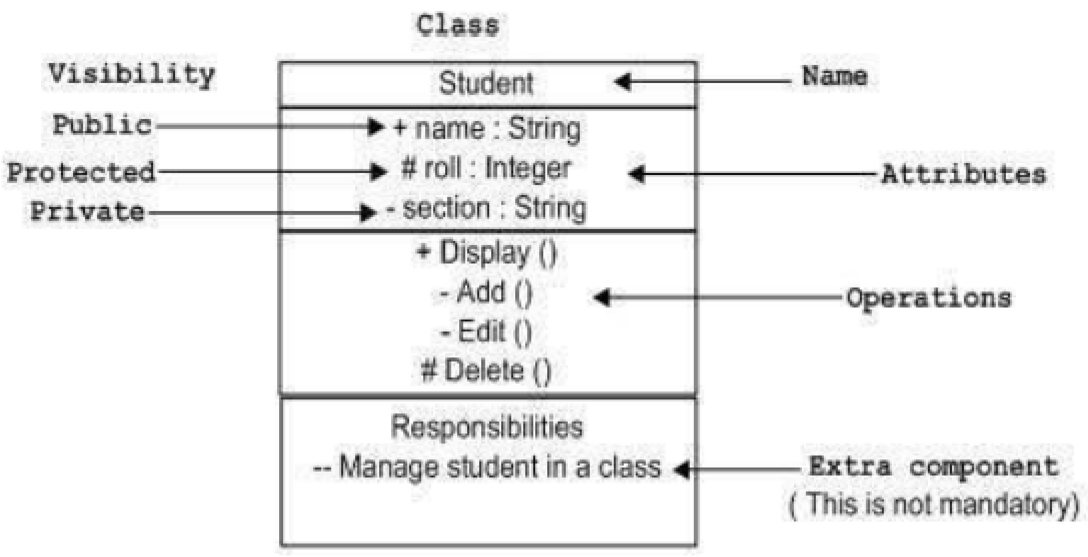


Diagram 2.1:Use Case Diagram

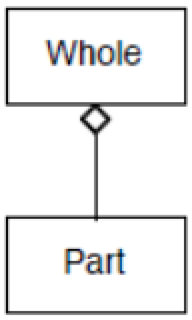
# Class diagram

## Symbols used in Class diagram:

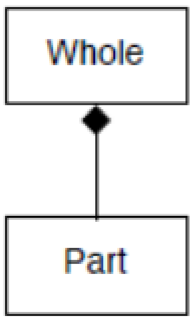
**Class**



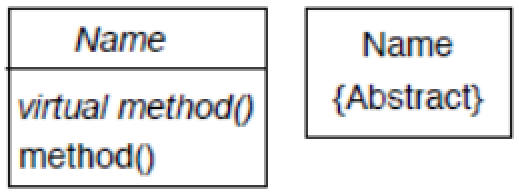
**Aggregation**



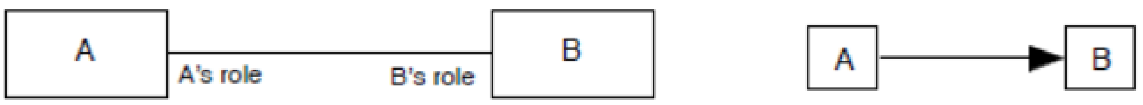
**Composition**



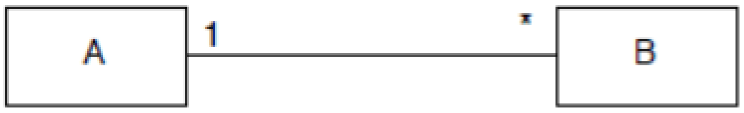
**Abstract Class**



**Association**



**Multiplicity in Aggregation, Composition, or Association**



## **Class diagram:**

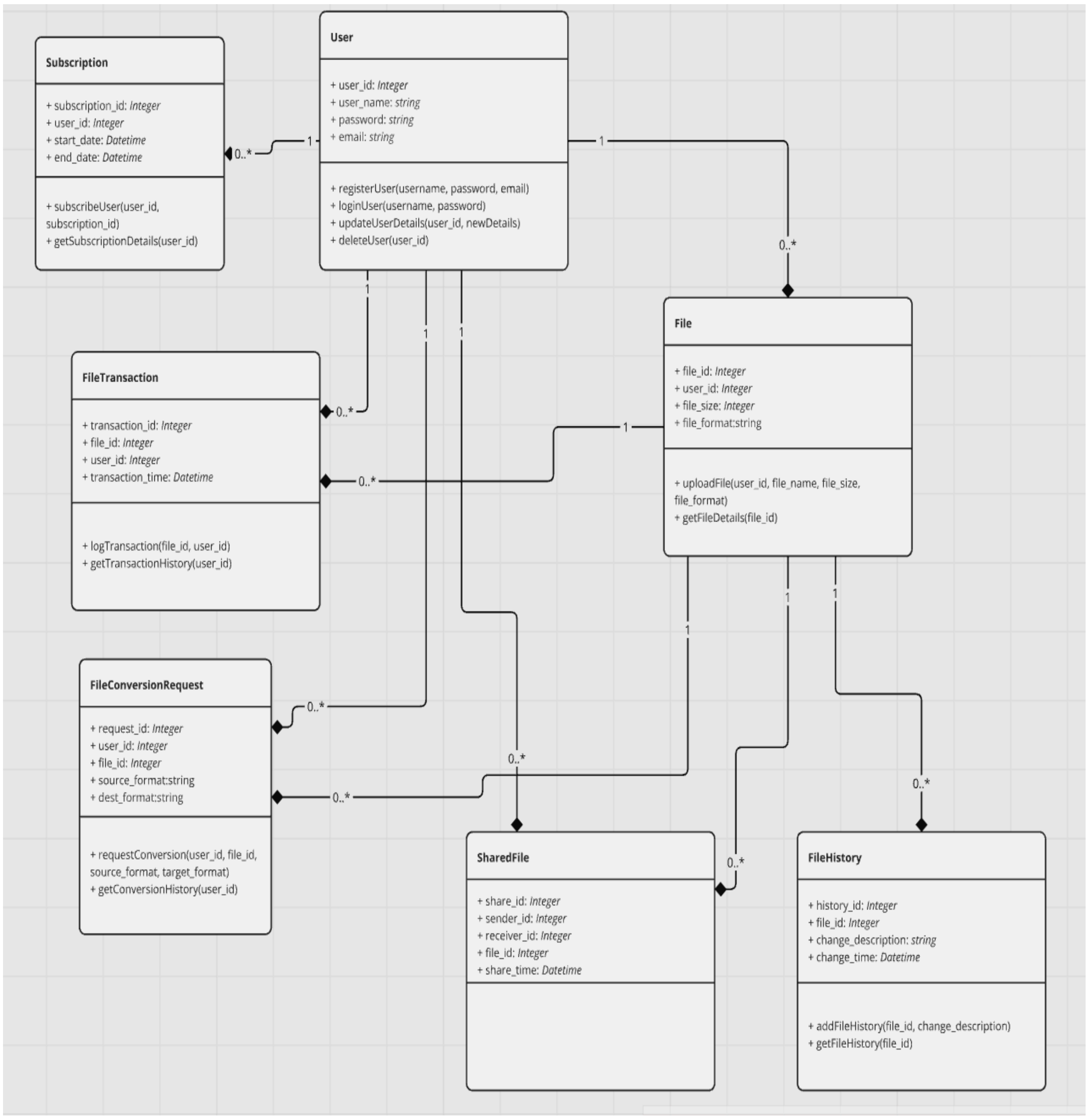
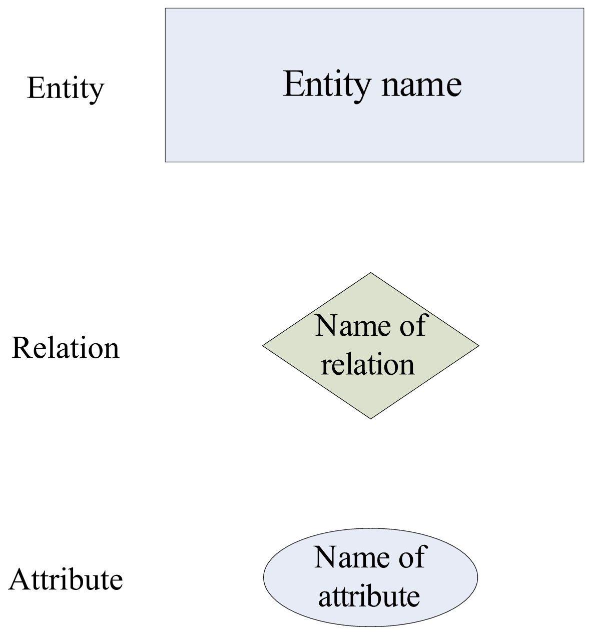

Subscription
+ subscription_id: Integer
+ user_id: Integer
+ start_date: Datetime
+ end_date: Datetime
+ subscribeUser(user_id, subscription_id)
+ getSubscriptionDetails(user_id)
User
+ user_id: Integer
+ user_name: string
+ password: string
+ email: string
+ registerUser(username, password, email)
+ loginUser(username, password)
+ updateUserDetails(user_id, newDetails)
+ deleteUser(user_id)
File
+ file_id: Integer
+ user_id: Integer
+ file_size: Integer
+ file_format:string
+ uploadFile(user_id, file_name, file_size, file_format)
+ getFileDetails(file_id)
FileTransaction
+ transaction_id: Integer
+ file_id: Integer
+ user_id: Integer
+ transaction_time: Datetime
+ logTransaction(file_id, user_id)
+ getTransactionHistory(user_id)
FileConversionRequest
+ request_id: Integer
+ user_id: Integer
+ file_id: Integer
+ source_format:string
+ dest_format:string
+ requestConversion(user_id, file_id, source_format, target_format)
+ getConversionHistory(user_id)
SharedFile
+ share_id: Integer
+ sender_id: Integer
+ receiver_id: Integer
+ file_id: Integer
+ share_time: Datetime
FileHistory
+ history_id: Integer
+ file_id: Integer
+ change_description: string
+ change_time: Datetime
+ addFileHistory(file_id, change_description)
+ getFileHistory(file_id)
1
0..*

Diagram 2.2: Class Diagram

# E-R Diagram

Entity-Relationship model is used to represent a logical design of a database to be created. In the ER model, real world objects (or concepts) are abstracted as entities, and different possible associations among them are modeled as relationships. We represent the attributes, entities and relation using the ER diagram. Using this ER diagram, table structures are created, along with required constraints. Finally, these tables are normalized in order to remove redundancy and maintain data integrity. Thus, to have data stored efficiently, the ER diagram is to be drawn as detailed and accurate as possible.

**Symbols used in ER diagram:**



**ER Diagram of the System:**

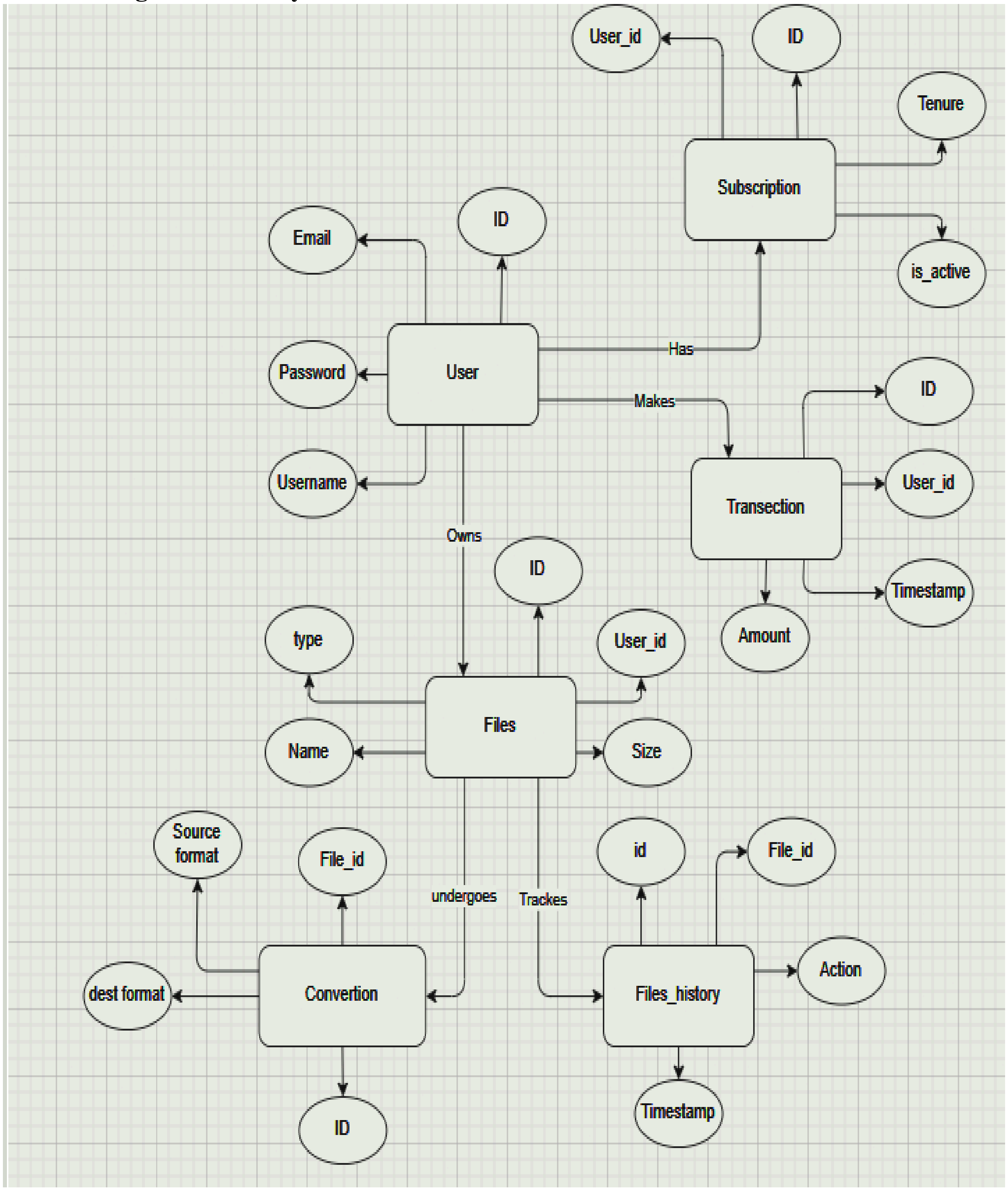
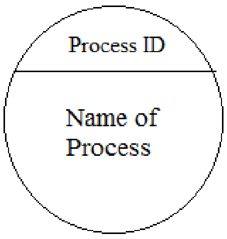
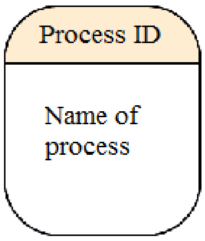
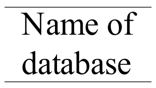
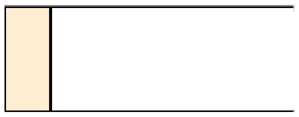
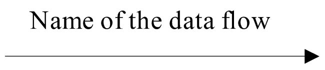
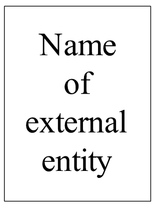


Diagram 2.3: E-R Diagram

## Data flow diagram

DFD provides the functional overview of a system. The graphical representation easily overcomes any gap between 'user and system analyst' and 'analyst and system designer' in understanding a system. Starting from an overview of the system it explores detailed design of a system through a hierarchy. DFD shows the external entities from which data flows into the process and also the other flows of data within a system. It also includes the transformations of data flow by the process and the data stores to read or write the data.

**Symbols used in Data Flow diagram:**

| | | | |
|---|---|---|---|
| Process |  | or |  |
| Data store |  | or |  |
| Data flow |  | | |
| External entity |  | | |

## Level 0

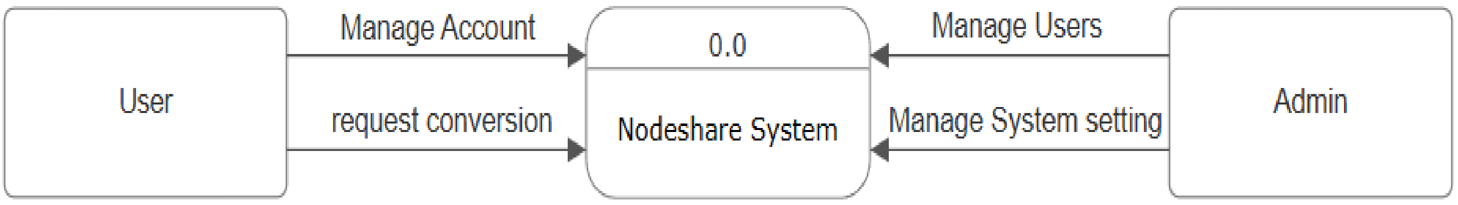


Diagram 2.4.1: Level-0 Data Flow Diagram

## Level 1

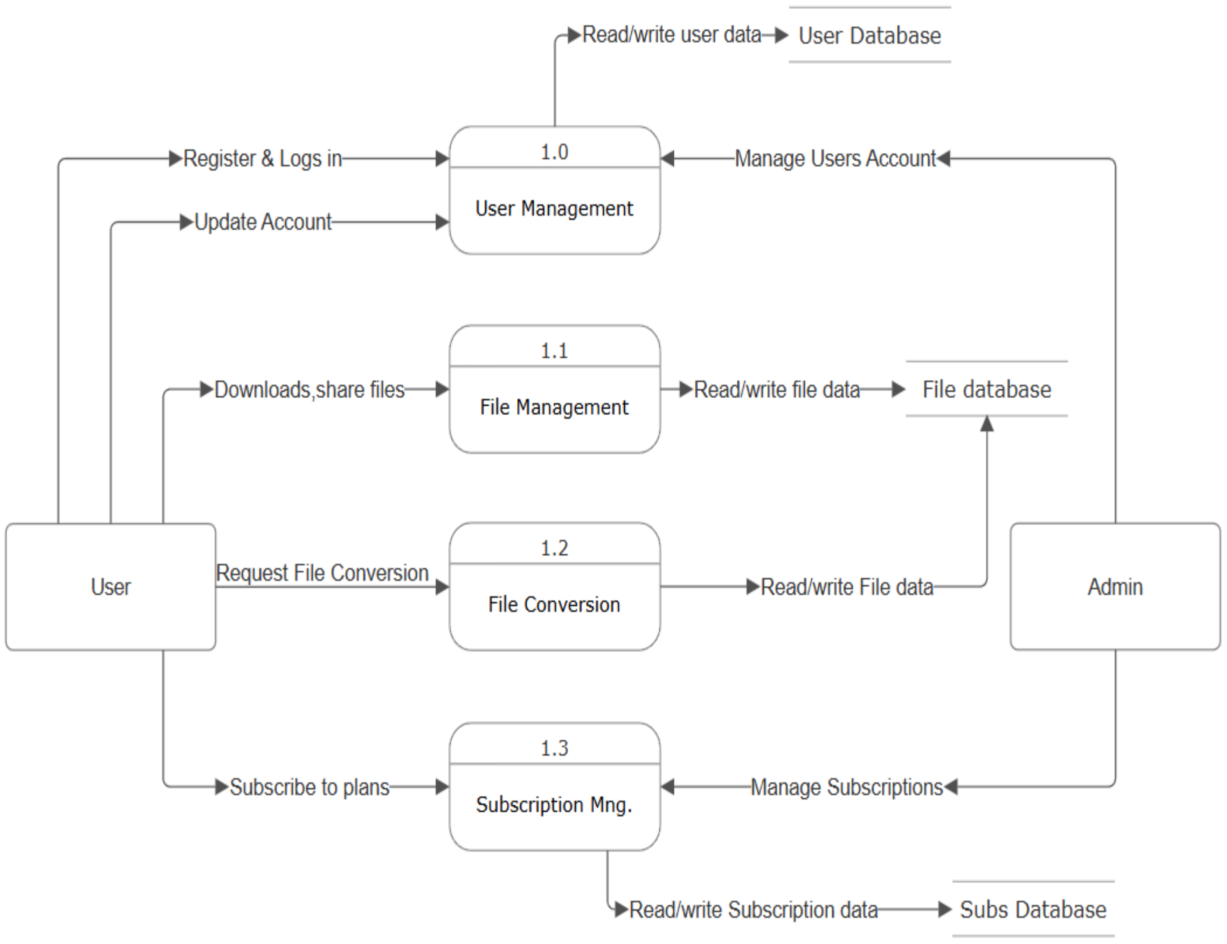


Diagram 2.4.2: Level-1 Data Flow Diagram

## Level 2

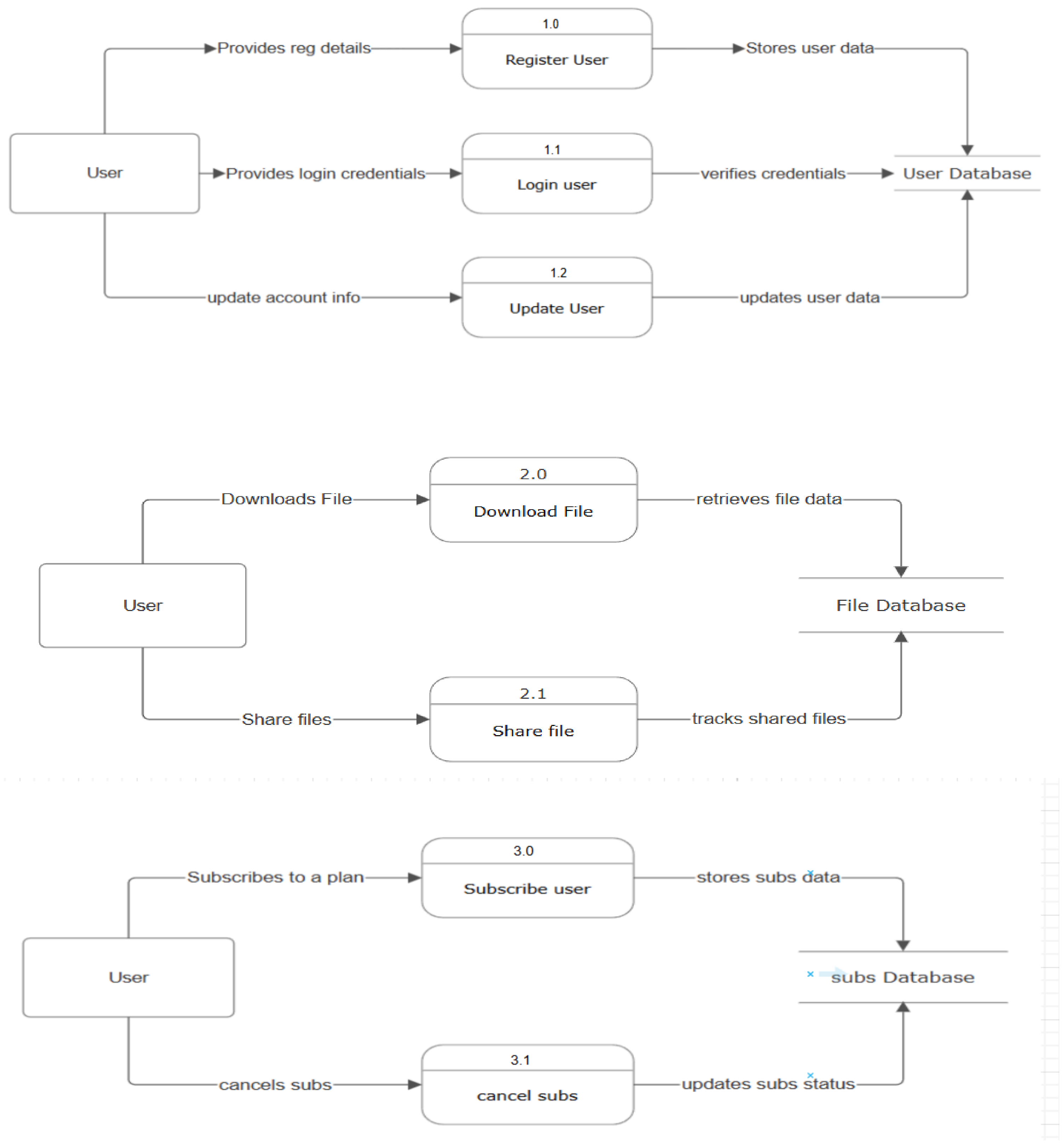


Diagram 2.4.3: Level-2 Data Flow Diagram

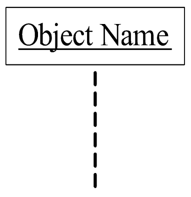
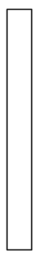
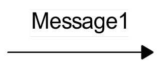
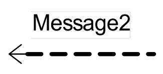
# Sequence diagram.

## Sequence diagram:

Sequence diagram represents the behavioural aspects of a system. Sequence diagram shows the interactions between the objects by means of passing messages from one object to another with respect to time in a system.

Sequence diagrams contain the objects of a system and their life-line bar and the messages passing between them. Objects appear at the top portion of the sequence diagram. Object is shown in a rectangle box. Name of the object precedes a colon ':' and the class name, from which the object is instantiated. The whole string is underlined and appears in a rectangle box. A down-ward vertical line from the object-box is shown as the life-line of the object. A rectangle bar on a life-line indicates that it is active at that point of time. Messages are shown as an arrow from the life-line of sender object to the life-line of receiver object and labelled with the message name.

**Symbols used in Sequence diagram:**

| | |
|---|---|
| Object |  |
| Life line bar |  |
| Call message |  |
| Return message |  |

## SEQUENCE DIAGRAM FOR SYSTEM:

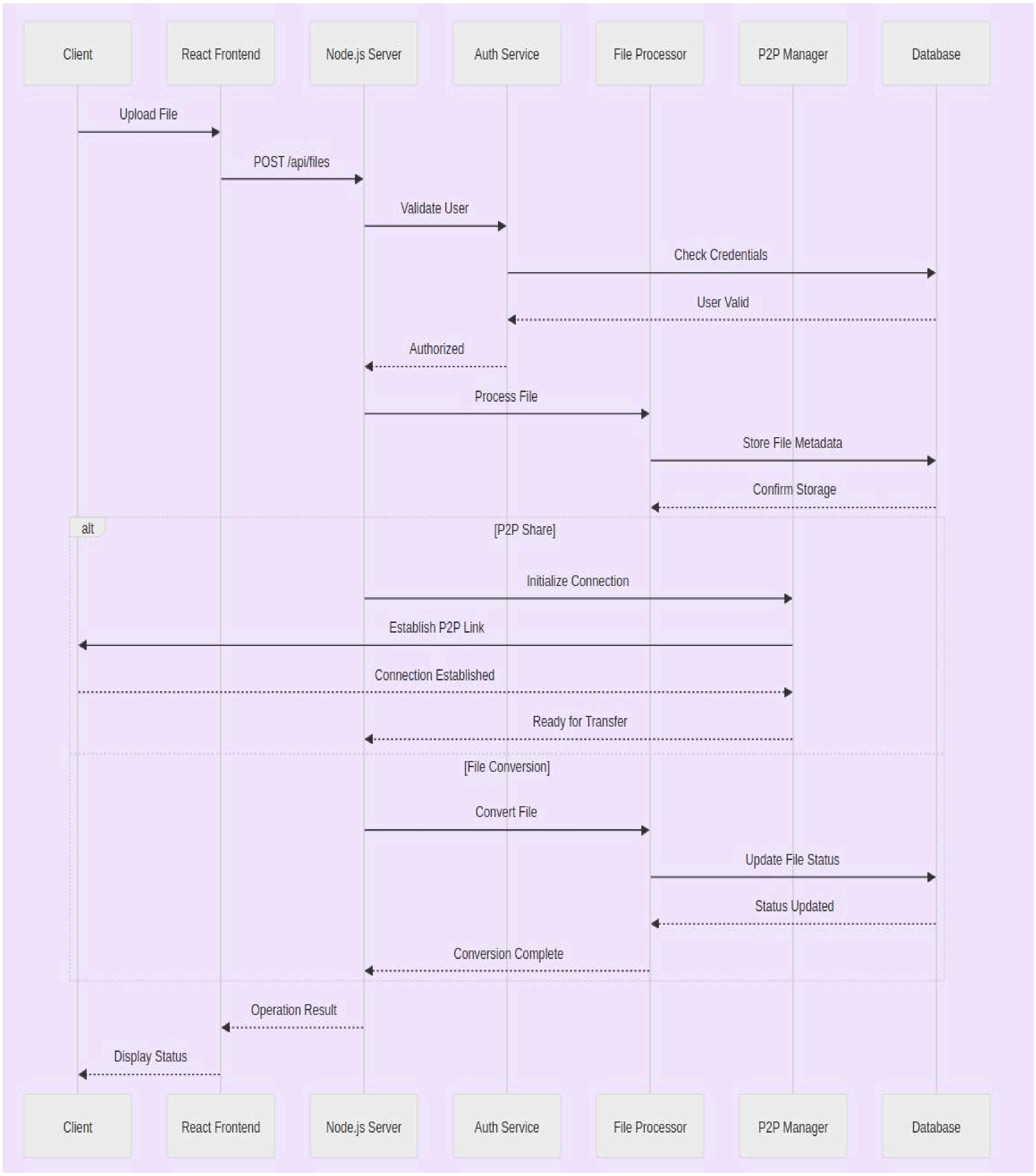


Diagram 2.5: Sequence Diagram

## State Diagram

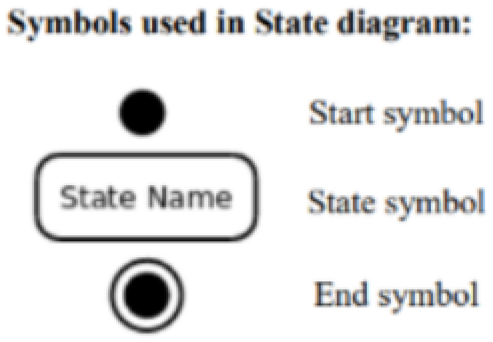
The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system. A State diagram describes a state machine. Now to clarify it, a state machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events. As the State diagram defines states it is used to model the lifetime of an object.

State diagram is one of the UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime. And these states are changed by events. So State diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

State diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of a State diagram is to model the lifetime of an object from creation to termination. State diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model a reactive system.

Following are the main purposes of using State diagrams:

- To model the dynamic aspect of a system.
- To model the lifetime of a reactive system.
- To describe different states of an object during its lifetime.
- Define a state machine to model states of an object



## STATE DIAGRAM FOR SYSTEM

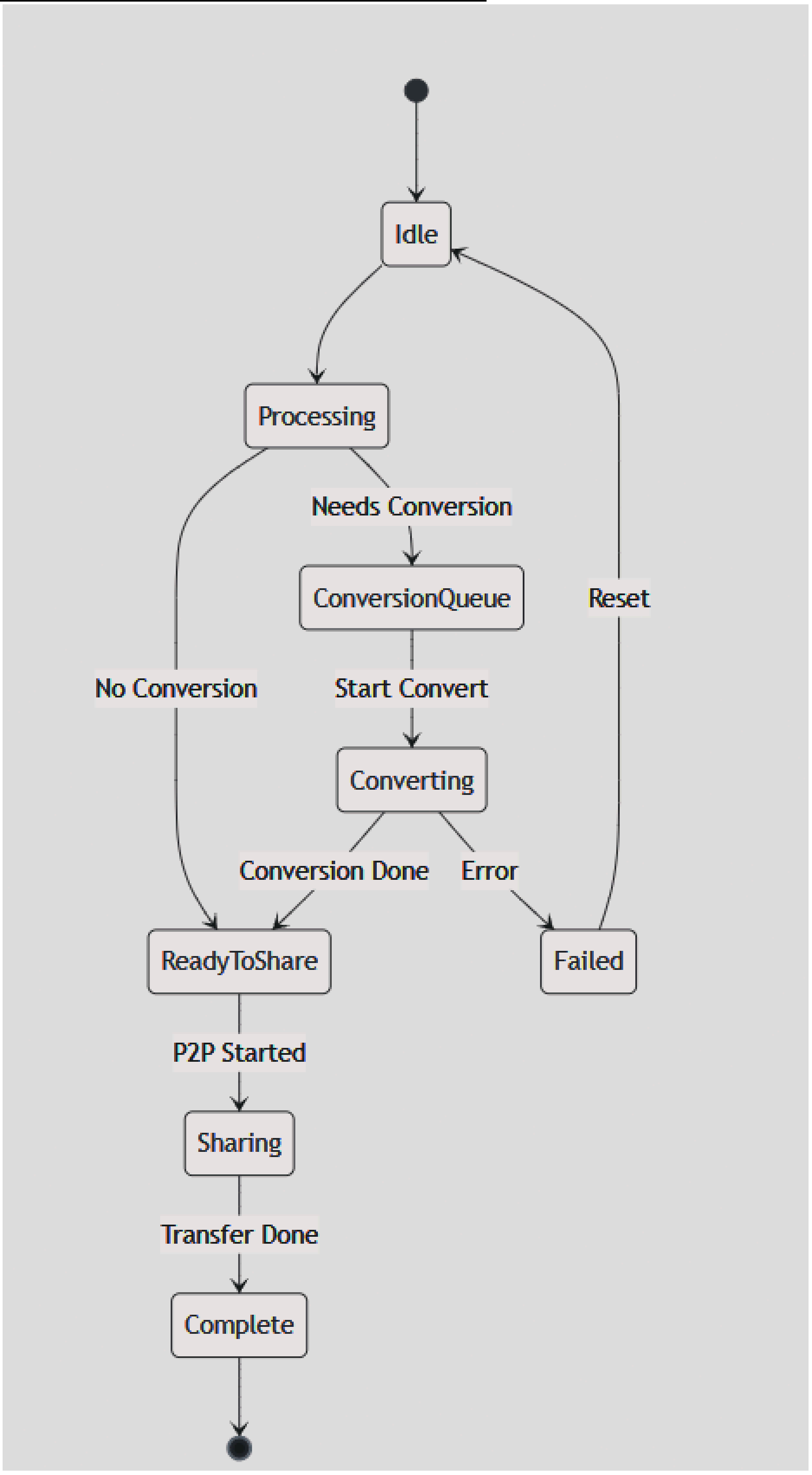
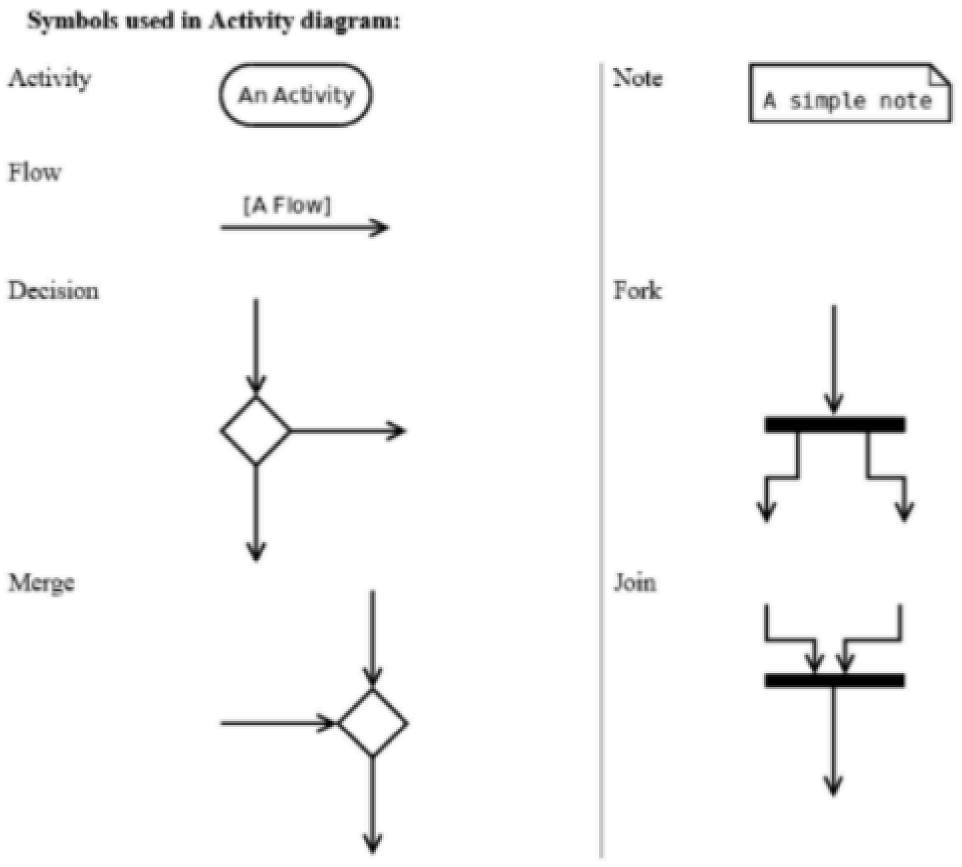


Diagram 2.6: State Diagram

## Activity Diagram

An activity denotes a particular action taken in the logical flow of control. This could simply be invocation of a mathematical function, alter an object's properties and so on. An activity is represented with a rounded rectangle, as shown in figure. A label inside the rectangle identifies the corresponding activity.

There are two special types of activity nodes: initial and final. They are represented with a filled circle, and a filled in circle with a border respectively. Initial node represents the starting point of a flow in an activity diagram. There could be multiple initial nodes, which means that invoking that particular activity diagram would initiate multiple flows. A final node represents the end point of all activities. Like an initial node, there could be multiple final nodes. Any transition reaching a final node would stop all activities. A flow is represented with a directed arrow. A decision node, represented with a diamond, is a point where a single flow enters and two or more flows leave. This is represented with a diamond shape, with two or more flows entering, and a single flow leaving out. Fork is a point where parallel activities begin. A join is depicted with a black bar, with multiple input flows, but a single output flow. Physically it represents the synchronization of all concurrent activities.



## ACTIVITY DIAGRAM FOR SYSTEM

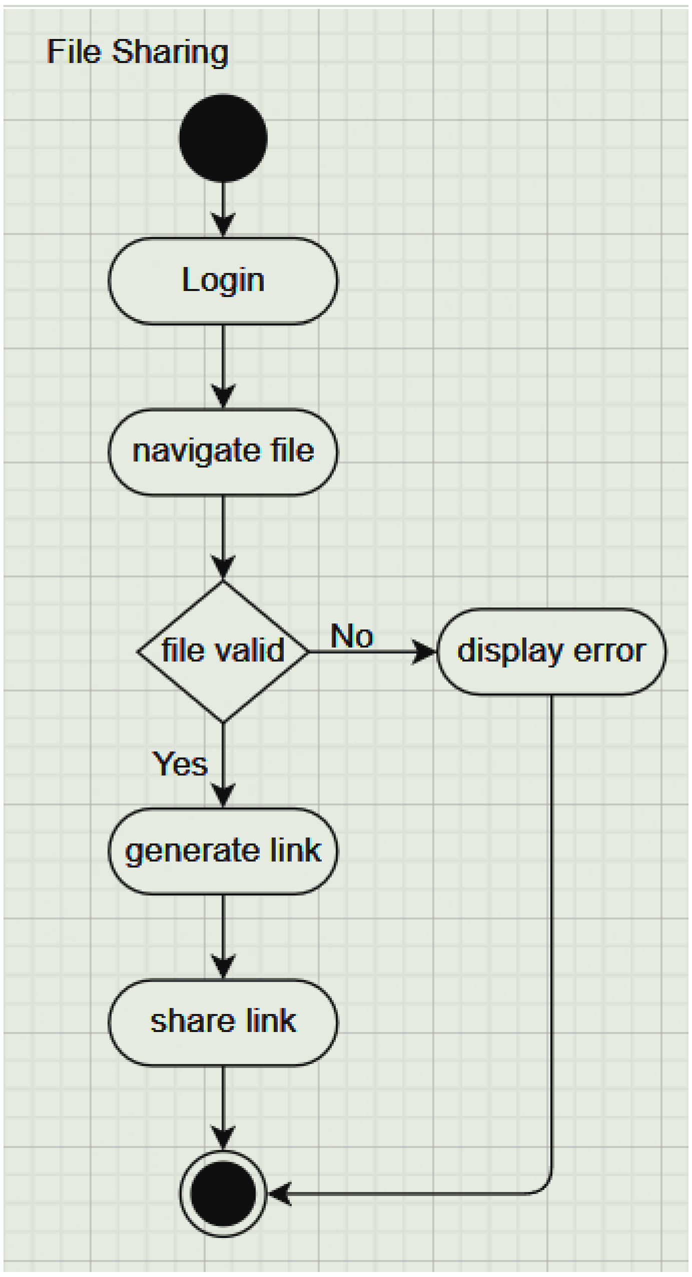


Diagram 2.7.1: Activity Diagram

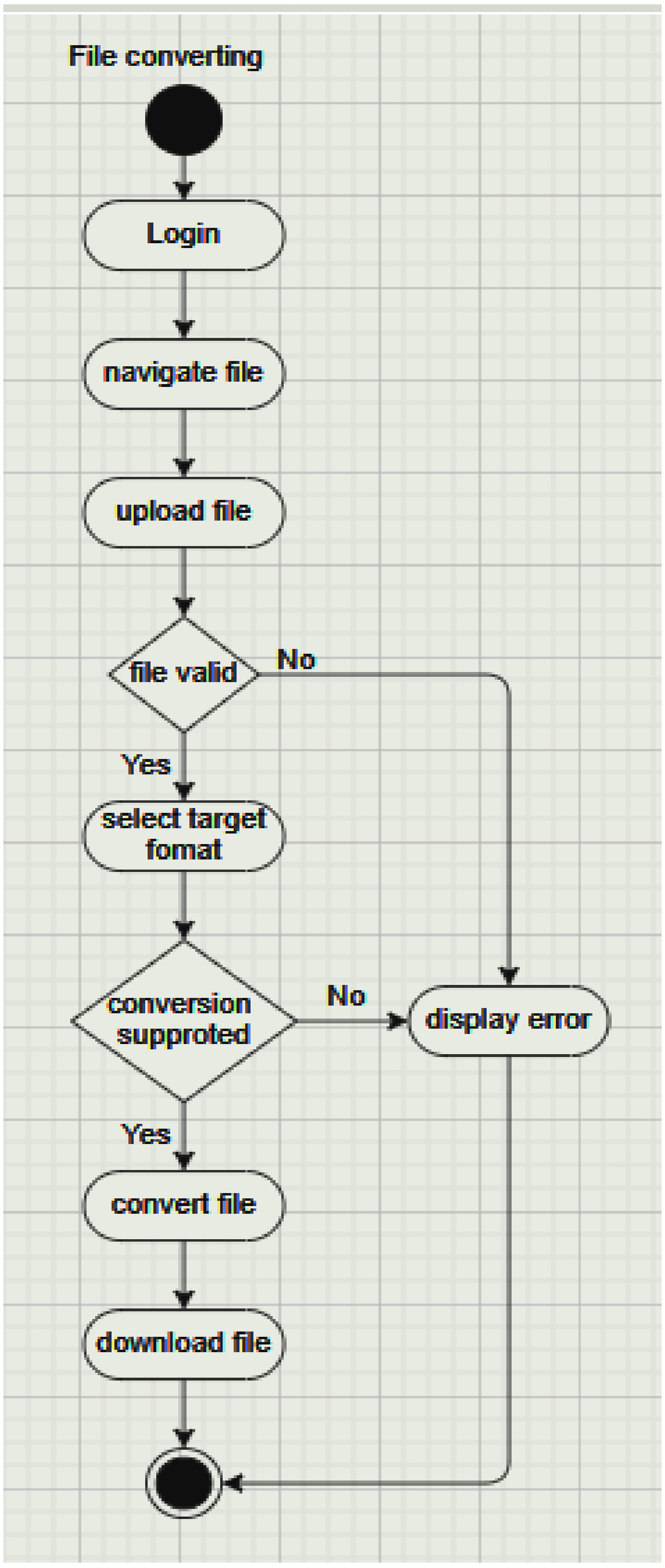
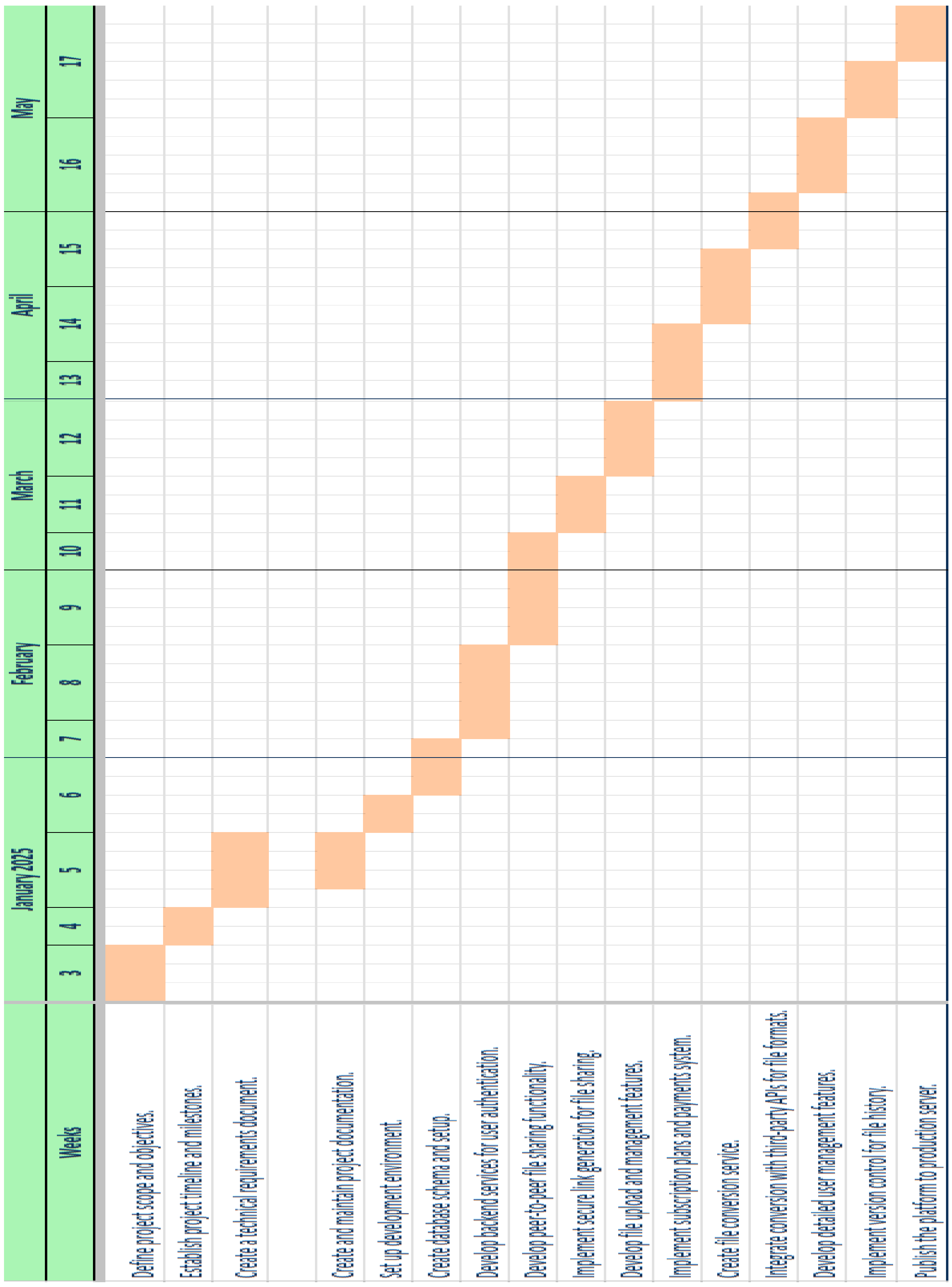


Diagram 2.7.2: Activity Diagram

## Timeline Chart describing the Project Scheduling.



|  | January 2025 |  |  |  | February |  |  | March |  |  | April |  |  | May |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Weeks | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| Define project scope and objectives. | 🟧 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Establish project timeline and milestones. |  | 🟧 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Create a technical requirements document. |  |  | 🟧 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Create and maintain project documentation. |  |  | 🟧 |  |  |  |  |  |  |  |  |  |  |  |  |
| Set up development environment. |  |  |  | 🟧 |  |  |  |  |  |  |  |  |  |  |  |
| Create database schema and setup. |  |  |  | 🟧 | 🟧 |  |  |  |  |  |  |  |  |  |  |
| Develop backend services for user authentication. |  |  |  |  | 🟧 | 🟧 |  |  |  |  |  |  |  |  |  |
| Develop peer-to-peer file sharing functionality. |  |  |  |  |  |  | 🟧 | 🟧 |  |  |  |  |  |  |  |
| Implement secure link generation for file sharing. |  |  |  |  |  |  |  |  | 🟧 |  |  |  |  |  |  |
| Develop file upload and management features. |  |  |  |  |  |  |  |  |  | 🟧 |  |  |  |  |  |
| Implement subscription plans and payments system. |  |  |  |  |  |  |  |  |  |  | 🟧 | 🟧 |  |  |  |
| Create file conversion service. |  |  |  |  |  |  |  |  |  |  |  | 🟧 | 🟧 |  |  |
| Integrate conversion with third-party APIs for file formats. |  |  |  |  |  |  |  |  |  |  |  |  | 🟧 | 🟧 |  |
| Develop detailed user management features. |  |  |  |  |  |  |  |  |  |  |  |  |  | 🟧 |  |
| Implement version control for file history. |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 🟧 |
| Publish the platform to production server. |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 🟧 |

## Project Modules

### 1. File transaction table
**Description:**This table will store details of file transactions (uploads, downloads, etc.)

| Sr. No. | Name | Datatype | Constraint | Description |
|---|---|---|---|---|
| 1 | transaction_id | Integer | Primary key | Unique transaction ID |
| 2 | file_id | Integer | Foreign key | Foreign key to Files.file_id |
| 3 | user_id | Integer | Foreign key | Foreign key to Users.user_id |
| 4 | transaction_type | Varchar(20) | Not null | Type of transaction (upload/download) |
| 5 | transaction_time | Datetime | Not null | Timestamp of the transaction |

Table 3.1: File transaction table

### 2. User Management Table
**Description**: This table stores the information about the users of Nodeshare.

| Sr. No. | Name | Datatype | Constraint | Description |
|---|---|---|---|---|
| 1 | user_id | Integer | Primary key | Unique user ID |
| 2 | username | Varchar(100) | Not null | User's username |
| 3 | password_hash | Varchar(100) | Not null | Hashed password of the user |
| 4 | email | Varchar(100) | Not null | Email address of the user |
| 5 | user_type | Varchar(20) | Not null | Type of user (e.g., free, premium) |

Table 3.2: User management table

### 3. File Conversion Requests Table

**Description**: This table stores details about file conversion requests made by users.

| Sr. No. | Name | Datatype | Constraint | Description |
|---|---|---|---|---|
| 1 | request_id | Integer | Primary key | Unique request ID |
| 2 | user_id | Integer | Foreign key | Foreign key to Users.user_id |
| 3 | file_id | Integer | Foreign key | Foreign key to Files.file_id |
| 4 | source_format | Varchar(20) | Not null | Format of the original file |
| 5 | target_format | Varchar(20) | Not null | Desired format for the converted file |
| 6 | request_time | Datetime | Not null | Timestamp of the conversion request |
| 7 | conversion_status | Varchar(20) | Not null | Status of the conversion (pending/completed) |

Table 3.3: User management table

### 4. Shared Files Tracking Table

**Description**: This table tracks files that are shared among users.

| Sr. No. | Name | Datatype | Constraint | Description |
|---|---|---|---|---|
| 1 | share_id | Integer | Primary key | Unique share ID |
| 2 | sender_id | Integer | Foreign key | Foreign key to Users.user_id (sender) |
| 3 | receiver_id | Integer | Foreign key | Foreign key to Users.user_id (receiver) |
| 4 | file_id | Integer | Foreign key | Foreign key to Files.file_id |
| 5 | share_time | Datetime | Not null | Timestamp of the file being shared |

Table 3.4: Shared Files Tracking table

## 5. Files Table (Referenced in Multiple Modules)

**Description**: This table stores details about the files shared or converted on Nodeshare.

| Sr. No. | Name | Datatype | Constraint | Description |
|---|---|---|---|---|
| 1 | file_id | Integer | Primary key | Unique file ID |
| 2 | user_id | Integer | Foreign key | Foreign key to Users.user_id |
| 3 | file_name | Varchar(255) | Not null | Name of the file |
| 4 | file_size | Integer | Not null | Size of the file (in bytes) |
| 5 | file_format | Varchar(20) | Not null | Format/extension of the file |
| 6 | upload_time | Datetime | Not null | Timestamp of file upload |

Table 3.5: Files table

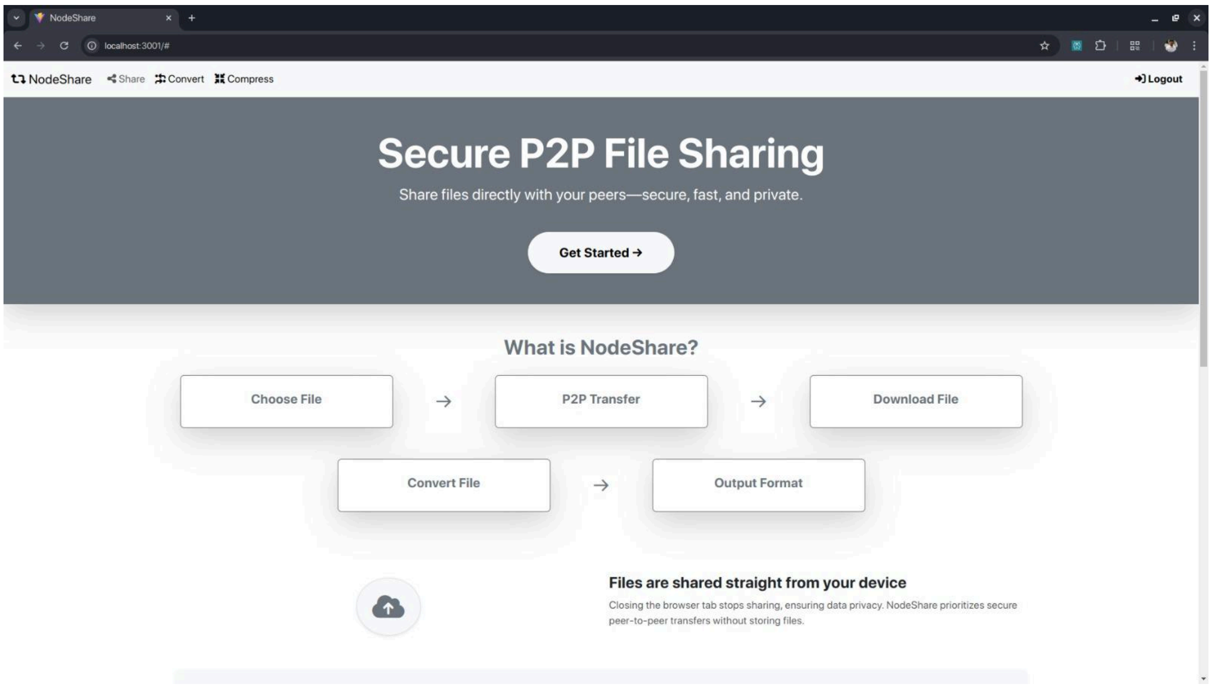
## 6. Subscription and Plans Module

**Description**: This module tracks user subscriptions for premium features like increased storage or faster conversion speeds.

| Sr. No. | Name | Datatype | Constraint | Description |
|---|---|---|---|---|
| 1 | subscription_id | Integer | Primary key | Unique subscription ID |
| 2 | user_id | Integer | Foreign key | Foreign key to Users.user_id |
| 3 | plan_name | Varchar(50) | Not null | Name of the subscription plan |
| 4 | plan_price | Decimal(10,2) | Not null | Monthly price of the plan |
| 5 | start_date | Datetime | Not null | Start date of the subscription |
| 6 | end_date | Datetime | Not null | Expiry date of the subscription |

Table 3.6: Subscription and plans table

## User Interface and Snapshot



Secure P2P File Sharing

Share files directly with your peers—secure, fast, and private.

Files are shared straight from your device

Closing the browser tab stops sharing, ensuring data privacy. NodeShare prioritizes secure peer-to-peer transfers without storing files.

Figure 3.1 – Home Page

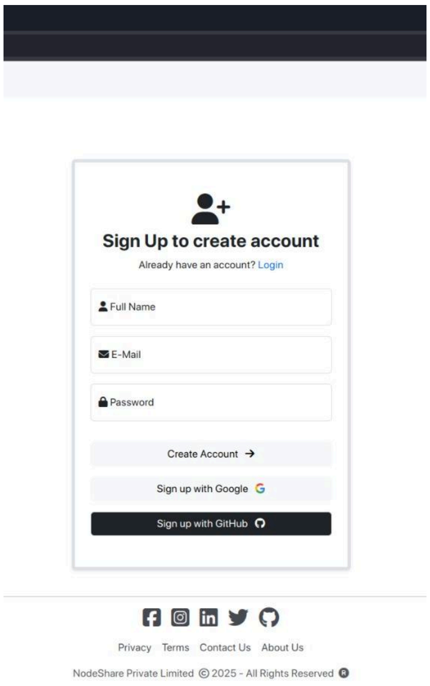

Sign Up to create account
Already have an account? Login
Full Name
E-Mail
Password
Create Account →
Sign up with Google
Sign up with GitHub
Privacy
Terms
Contact Us
About Us
NodeShare Private Limited © 2025 - All Rights Reserved

Figure 3.2 – Sign-up Page

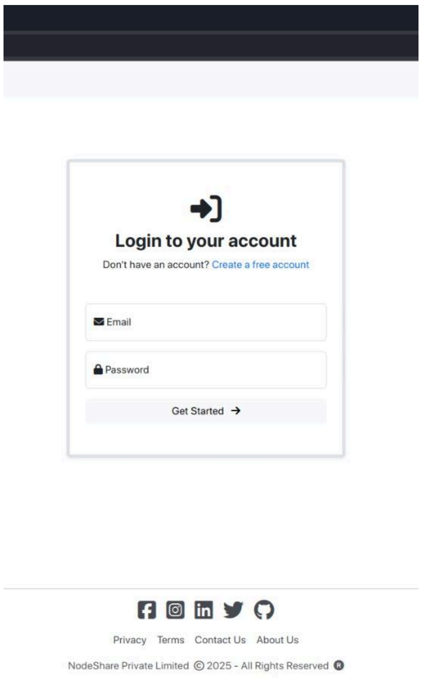

Login to your account
Don't have an account? Create a free account
Email
Password
Get Started →
Privacy
Terms
Contact Us
About Us
NodeShare Private Limited © 2025 - All Rights Reserved

Figure 3.3 – Login Page

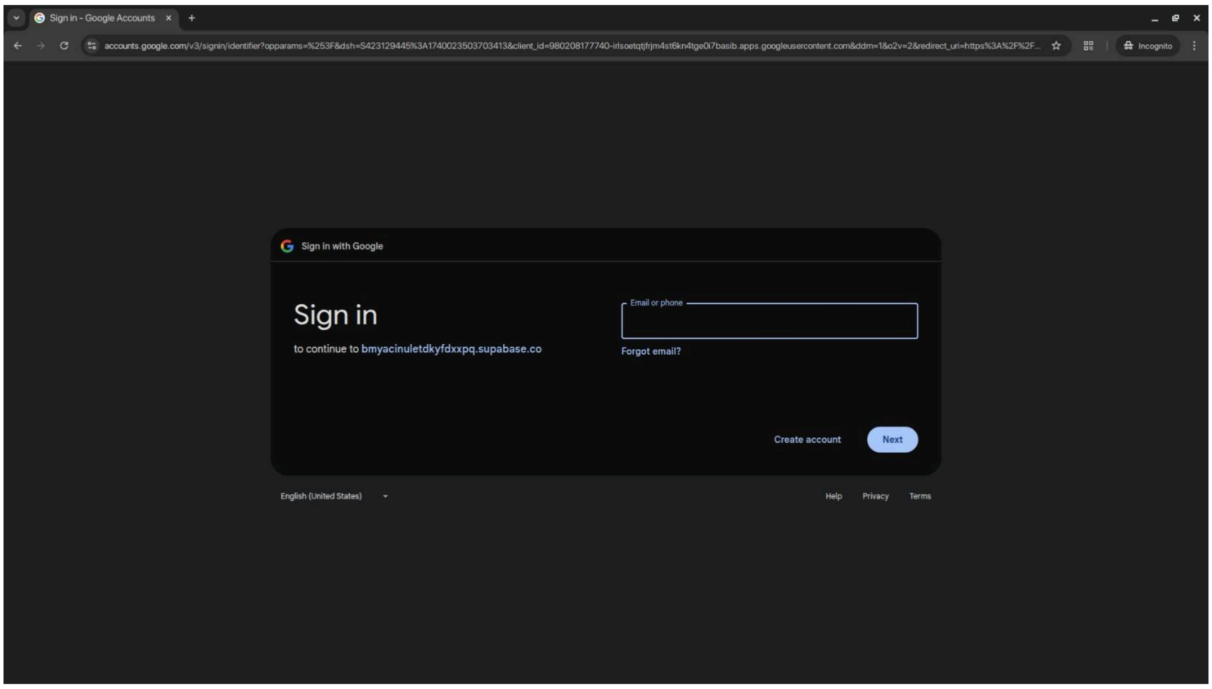

Sign in - Google Accounts
accounts.google.com/v3/signin/identifier?opparams=%253F&dsh=S423129445%3A1740023503703413&client_id=980208177740-irlsoetqtjfrjm4st6kn4tge0i7basib.apps.googleusercontent.com&ddm=1&o2v=2&redirect_uri=https%3A%2F%2F...
Incognito
Sign in with Google
Sign in
to continue to bmyacinuletdkyfdxxpq.supabase.co
Email or phone
Forgot email?
Create account
Next
English (United States)
Help
Privacy
Terms

Figure 3.4 – Sign up using Google

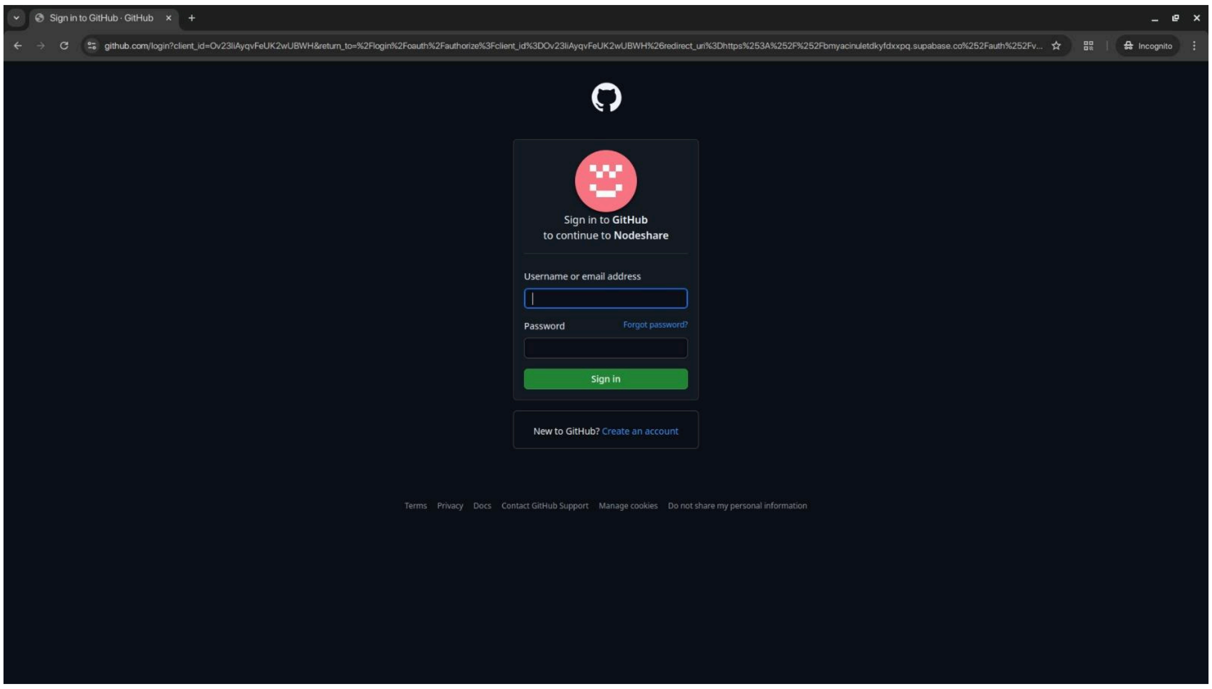

Sign in to GitHub · GitHub
github.com/login?client_id=Ov23liAyqvFeUK2wUBWH&return_to=%2Flogin%2Foauth%2Fauthorize%3Fclient_id%3DOv23liAyqvFeUK2wUBWH%26redirect_uri%3Dhttps%253A%252F%252Fbmyacinuletdkyfdxxpq.supabase.co%252Fauth%252Fv...
Incognito
Sign in to GitHub
to continue to Nodeshare
Username or email address
Password
Forgot password?
Sign in
New to GitHub? Create an account
Terms
Privacy
Docs
Contact GitHub Support
Manage cookies
Do not share my personal information

Figure 3.5 – Sign up using Github

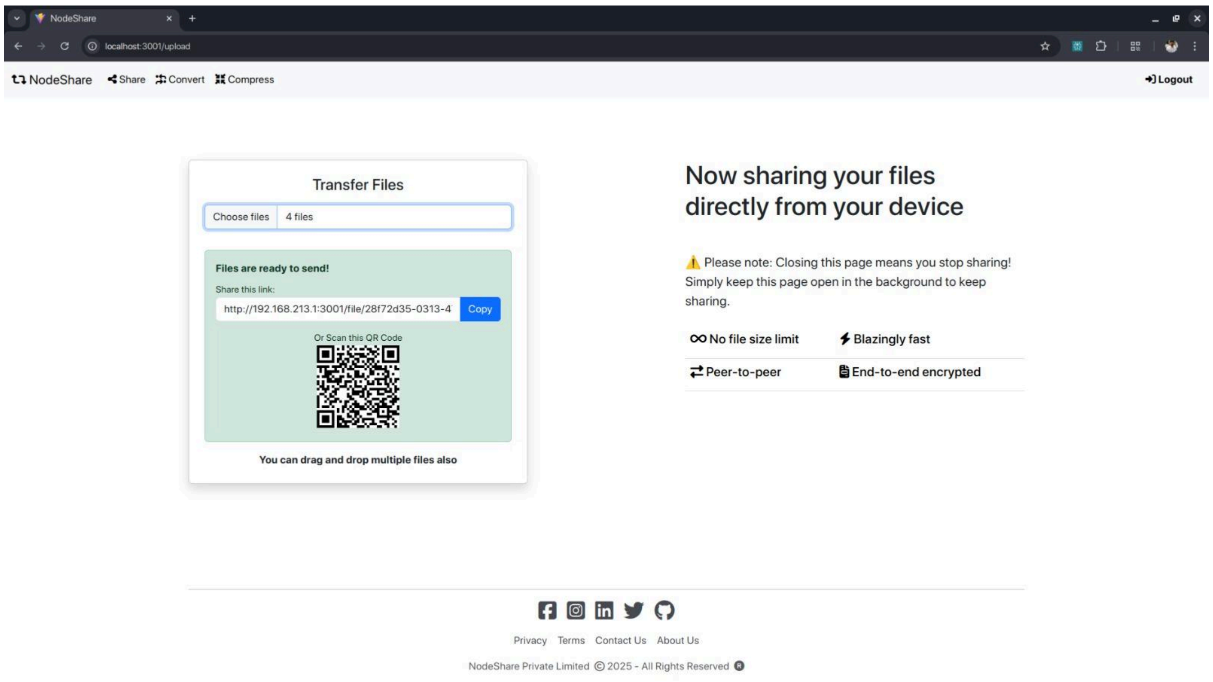

NodeShare
localhost:3001/upload
NodeShare
Share
Convert
Compress
Logout
Transfer Files
Choose files
4 files
Files are ready to send!
Share this link:
http://192.168.213.1:3001/file/28f72d35-0313-4
Copy
Or Scan this QR Code
You can drag and drop multiple files also
Now sharing your files directly from your device
Please note: Closing this page means you stop sharing! Simply keep this page open in the background to keep sharing.
No file size limit
Blazingly fast
Peer-to-peer
End-to-end encrypted
Privacy
Terms
Contact Us
About Us
NodeShare Private Limited © 2025 - All Rights Reserved

Figure 3.6 – File Transfer Module

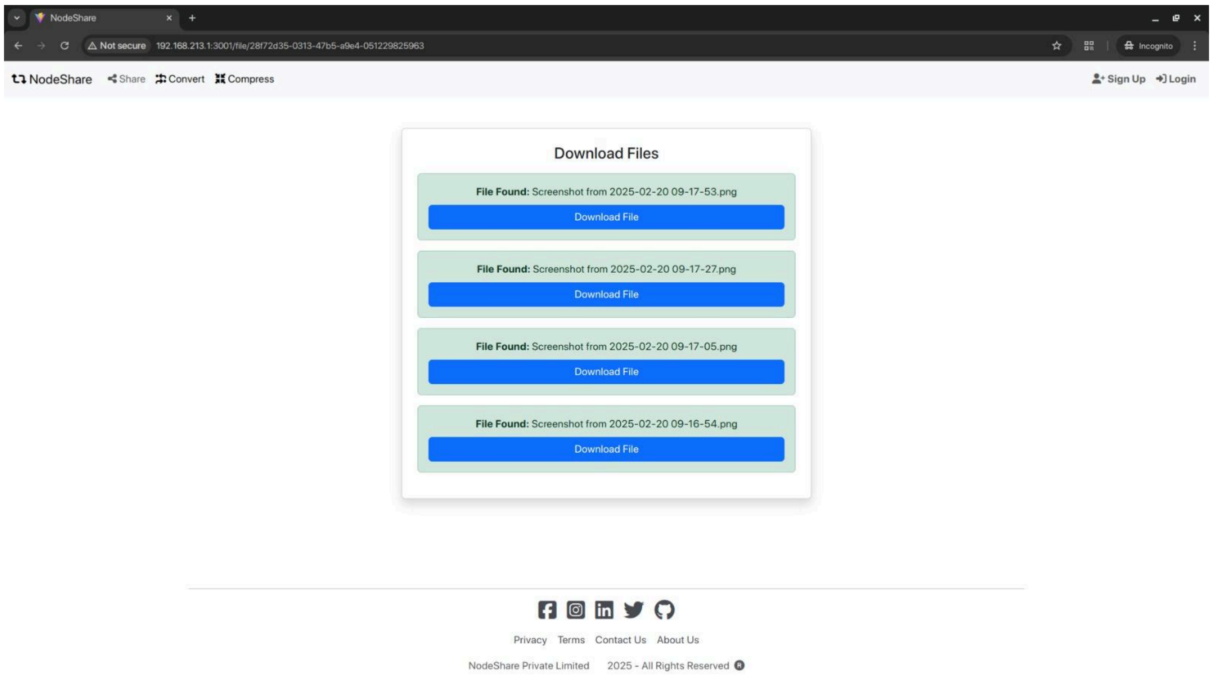

NodeShare
Not secure 192.168.213.1:3001/file/28f72d35-0313-47b5-a9e4-051229825963
Incognito
NodeShare
Share
Convert
Compress
Sign Up
Login
Download Files
File Found: Screenshot from 2025-02-20 09-17-53.png
Download File
File Found: Screenshot from 2025-02-20 09-17-27.png
Download File
File Found: Screenshot from 2025-02-20 09-17-05.png
Download File
File Found: Screenshot from 2025-02-20 09-16-54.png
Download File
Privacy
Terms
Contact Us
About Us
NodeShare Private Limited 2025 - All Rights Reserved

Figure 3.7 –File Download Module

Guide Remarks:

Guide Signature: