**Batch: T6**

**Practical No. 9**

**Title of Assignment: Study and implementation of node.js**

**Student Name: Parshwa Herwade**

**Student PRN: 22510064**

Perform following problem statements using Node.js

Problem Statement 1: Database Connectivity using SQL or Oracle

• Write a Node.js program that connects to an Oracle/SQL database, retrieves data

from a table, and displays the results.

ANS.**Technologies used:**

* **Node.js**: JavaScript runtime for server-side scripting.
* **MySQL/Oracle**: Relational databases.
* **mysql2/oracledb package**: Node.js modules to connect to MySQL/Oracle databases.

**Steps to set up:**

1. Install the required database module:
   * For MySQL: npm install mysql2
   * For Oracle: npm install oracledb

const mysql = require('mysql2');

// Create a connection to the database

const connection = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'password',

database: 'your\_database'

});

// Connect to the database

connection.connect((err) => {

if (err) {

return console.error('error connecting: ' + err.stack);

}

console.log('connected as id ' + connection.threadId);

});

// Query the database

connection.query('SELECT \* FROM your\_table', (err, results) => {

if (err) throw err;

console.log(results);

connection.end();

});

Problem Statement 2: Middleware (Express.js)

• What is middleware in Node.js, particularly in the context of Express.js?

ANS.**Technologies used:**

* **Node.js**: Server-side framework.
* **Express.js**: Web application framework for building APIs.

**Explanation:**

* **Middleware**: Functions that execute during the lifecycle of a request to the server. Each middleware function can execute code, modify the request/response objects, and call the next middleware function in the stack.

• How do you create custom middleware in Express.js?

ANS. const express = require('express');

const app = express();

// Custom middleware

function myMiddleware(req, res, next) {

console.log('Middleware executed!');

next(); // Pass control to the next middleware

}

app.use(myMiddleware);

app.get('/', (req, res) => {

res.send('Hello World');

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

• Explain how middleware is executed in order in an Express.js application.

ANS.**Middleware execution order:**

1. Middleware functions are executed sequentially, in the order they are defined.
2. If a middleware function doesn't call next(), the request is halted.

Problem Statement 3: File System (fs) Module

• How do you read and write files using the fs module in Node.js?

ANS.**Technologies used:**

* **Node.js**: Server-side scripting.
* **fs module**: Provides an API for interacting with the file system.

**Reading and writing files:**

**const fs = require('fs');**

**fs.readFile('example.txt', 'utf8', (err, data) => {**

**if (err) throw err;**

**console.log(data);**

**});**

* **Reading file asynchronously**:

fs.writeFile('example.txt', 'Hello World!', (err) => {

if (err) throw err;

console.log('File has been written');

});

• What is the difference between fs.readFile() and fs.readFileSync()?

ANS.**Difference between fs.readFile() and fs.readFileSync():**

* **fs.readFile()**: Asynchronous, non-blocking.
* **fs.readFileSync()**: Synchronous, blocking

• How can you check if a file or directory exists in Node.js?

ANS. fs.access('example.txt', fs.constants.F\_OK, (err) => {

console.log(err ? 'File does not exist' : 'File exists');

});

• How do you handle file operations in an asynchronous manner?

ANS. const fs = require('fs/promises');

async function readFileAsync(filePath) {

try {

const data = await fs.readFile(filePath, 'utf-8');

console.log(data);

} catch (error) {

console.error('Error reading file:', error);

}

}

readFileAsync('example.txt');

Problem Statement 4: File Upload and Download API

Develop a file upload and download API using Node.js and Express. The API should allow

users to upload files (e.g., images, documents) and download them later.

• Create an API to upload files to the server.

• Implement routes to retrieve and download files.

• Ensure proper error handling (e.g., file size limits, invalid file formats).

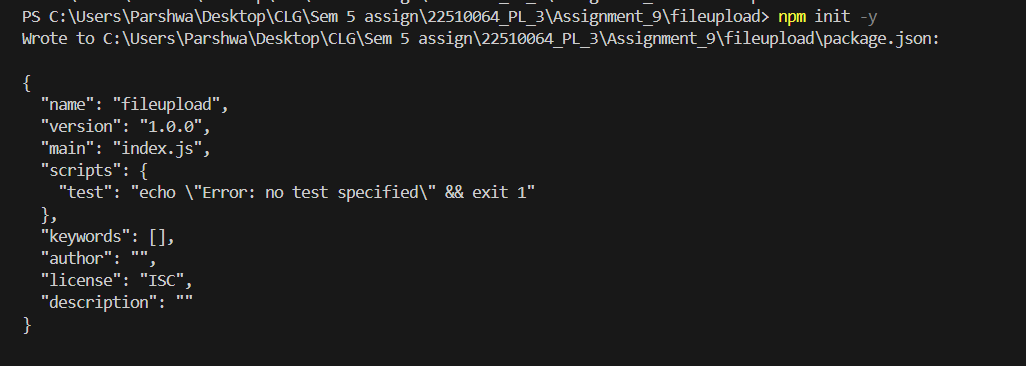
• Implement file versioning to allow multiple uploads of the same file name

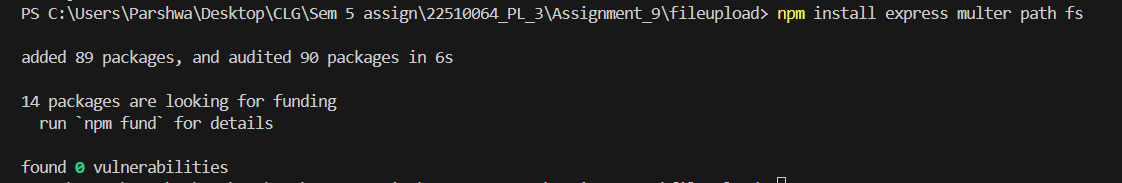
without overwriting.

ANS. **Technologies used:**

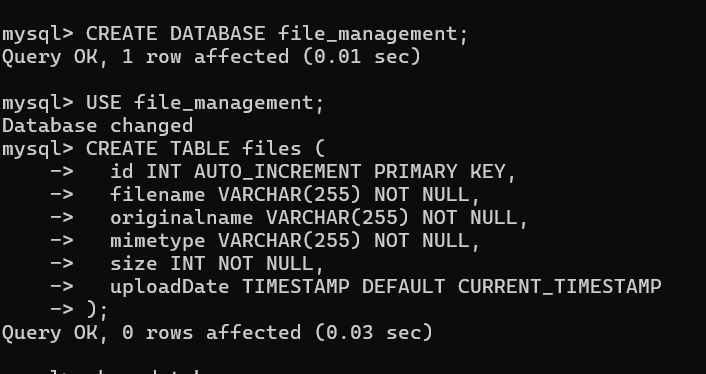
* **Node.js**: JavaScript runtime.
* **Express.js**: Framework for API routes.
* **Multer**: Middleware for handling multipart form data (file uploads).

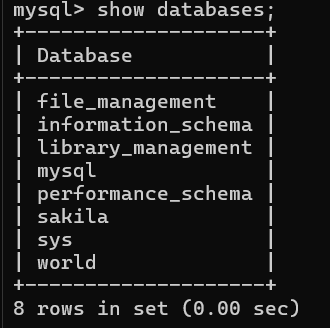
**Process**:



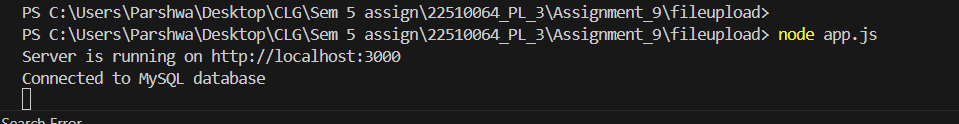
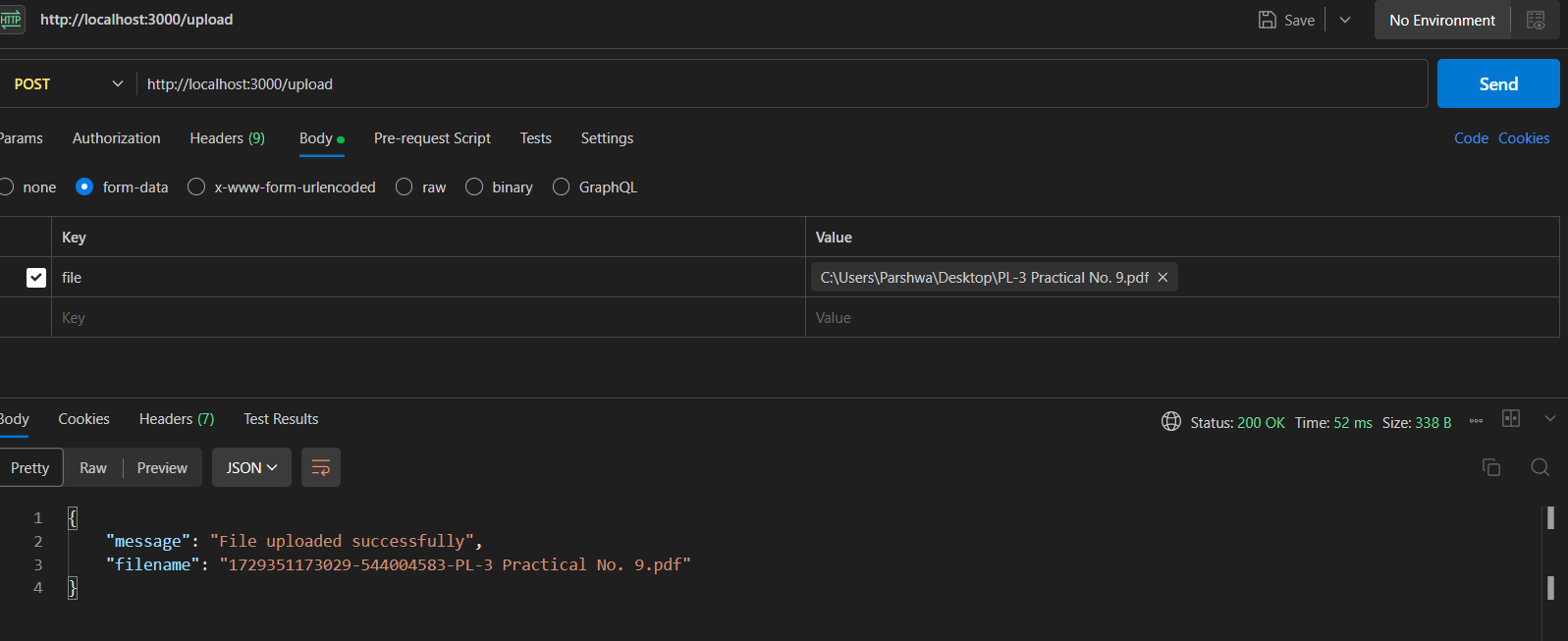


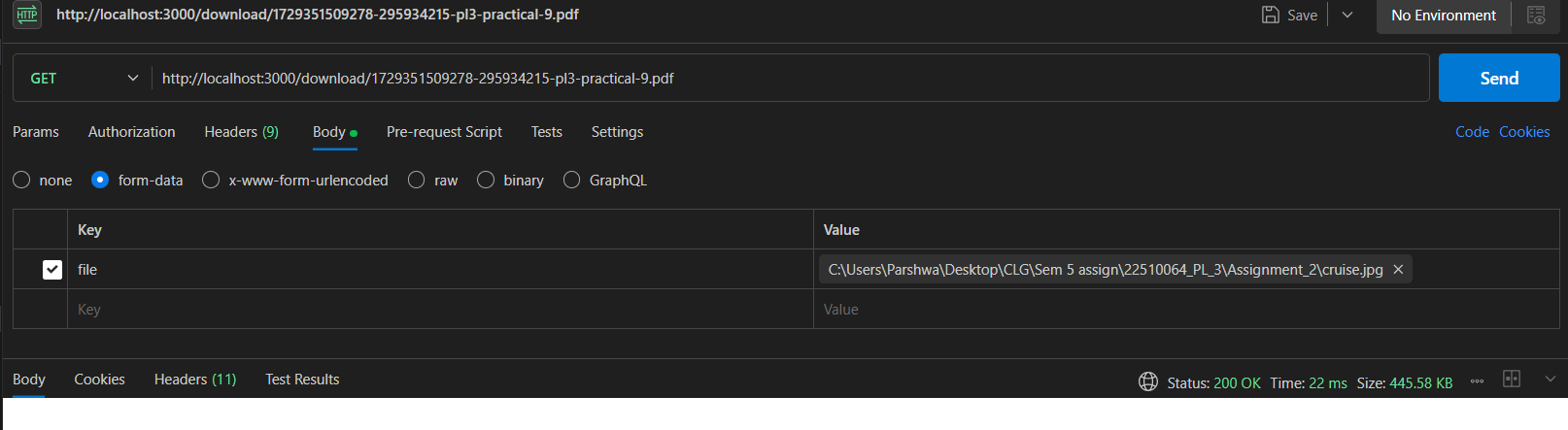
Similar way install mysql through npm packages.

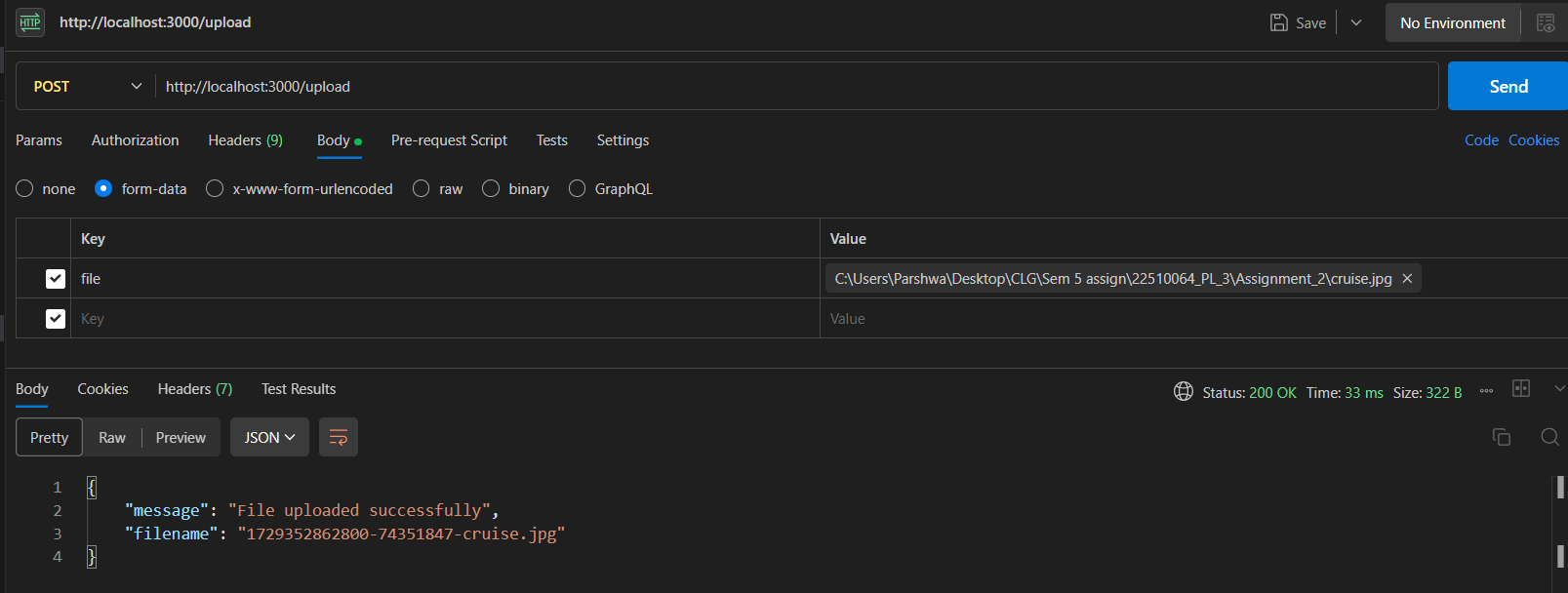
Create a file\_management database in mysql workbench:  




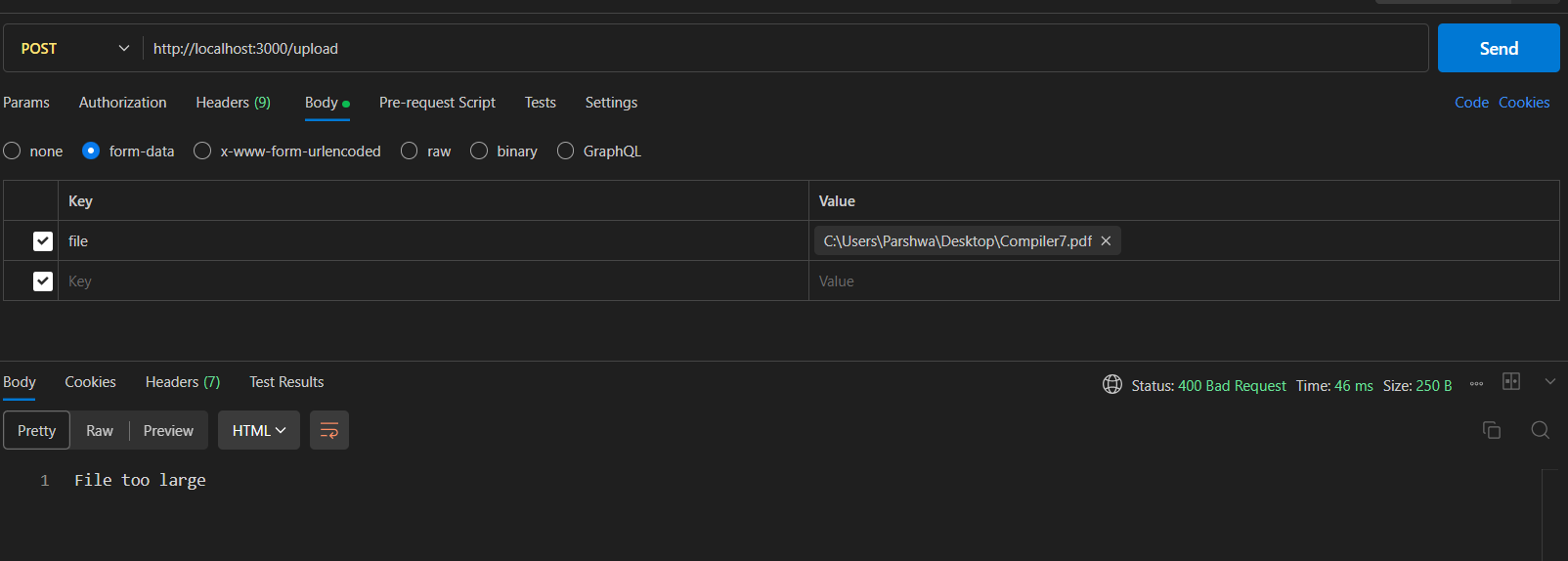
Connect to the database through app.js file :

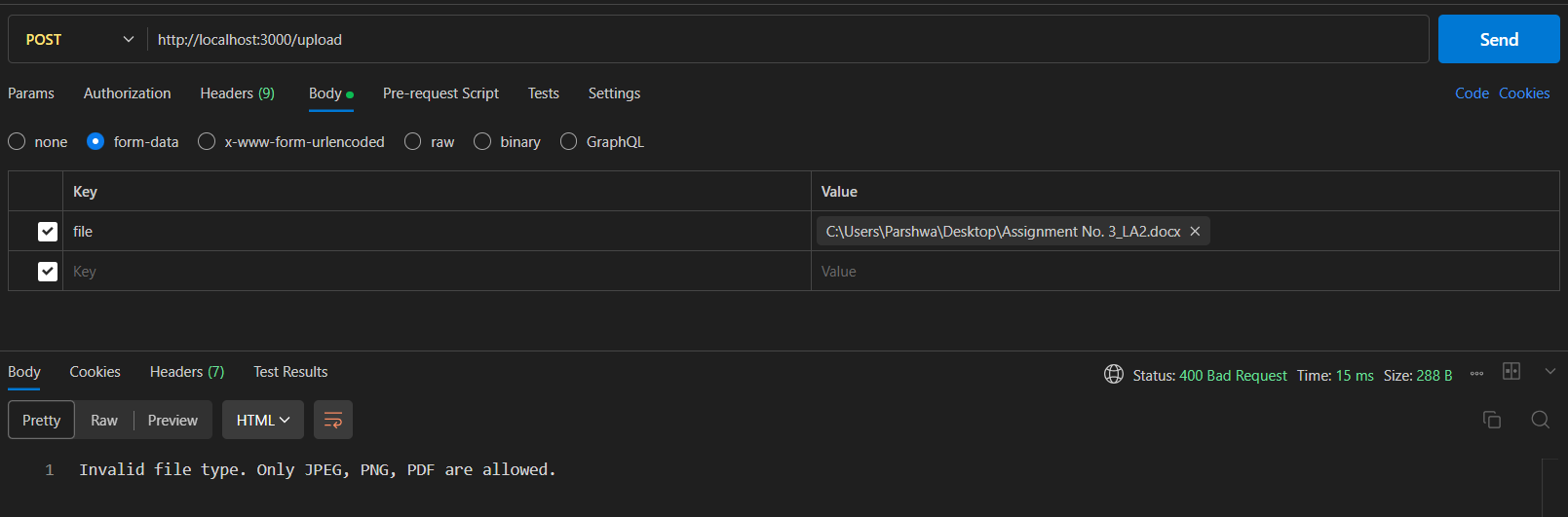
  
And then do a post request using postman:  


Now to download the file just put download in the url and the file name and paste it in browser to download the file:  


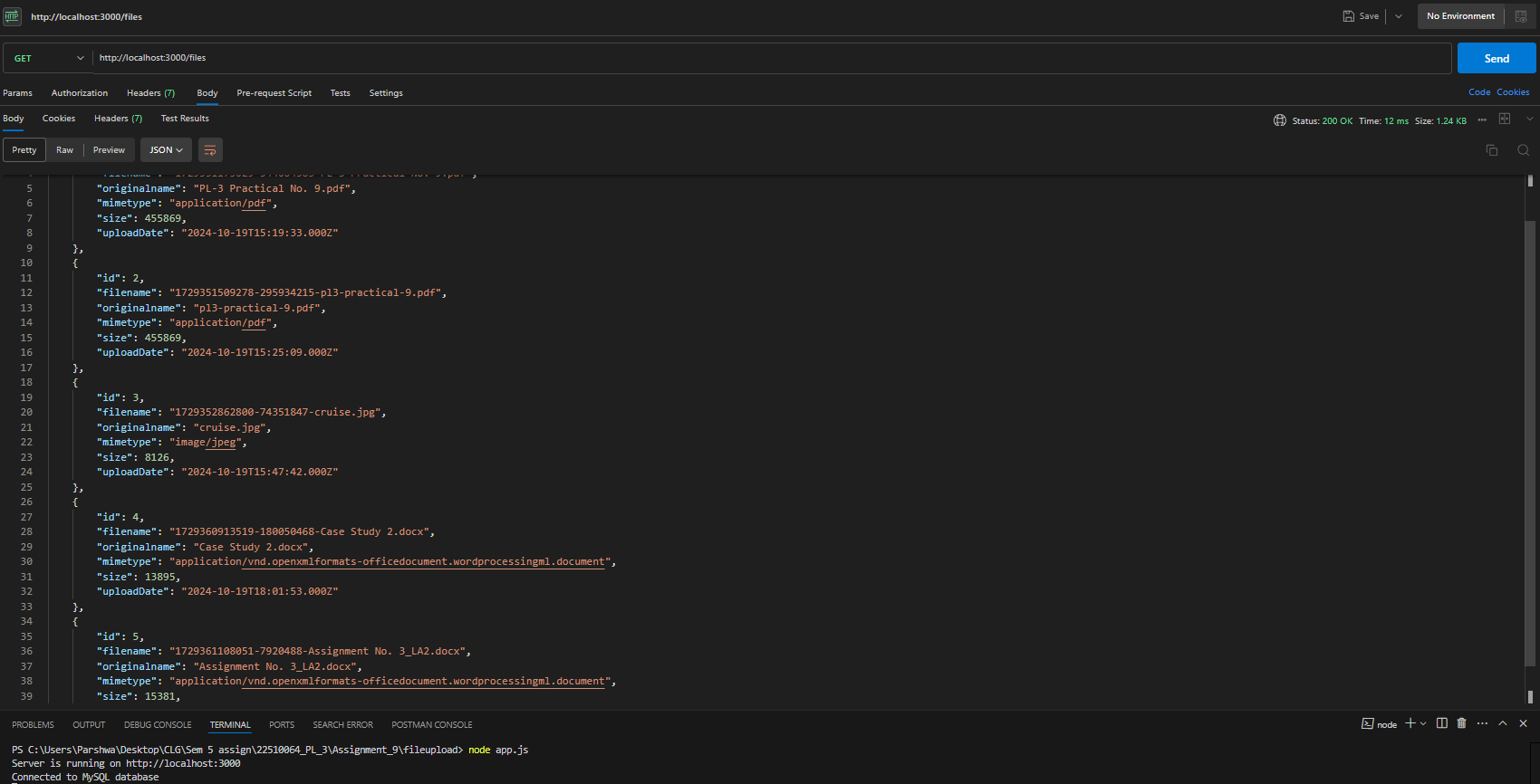


Errors :



When a docx file is not allowed and tried to upload it gives error:  


Getting all files in one view:



Problem Statement 5: Real-time Chat Application with Socket.io

Create a real-time chat application using Node.js, Express, and Socket.io that allows

multiple users to join and communicate in a chat room.

• Set up a Node.js server with Socket.io for real-time bi-directional

communication.

• Implement event listeners to handle user connections, disconnections, and

message broadcasting to all connected users

**Theory and Technologies Used**

In a real-time chat application, the goal is to provide seamless, instantaneous communication between multiple users without the need for continuous refreshing. To achieve this, the application utilizes the **Socket.io** library, which allows for **bi-directional real-time communication** between a server and clients.

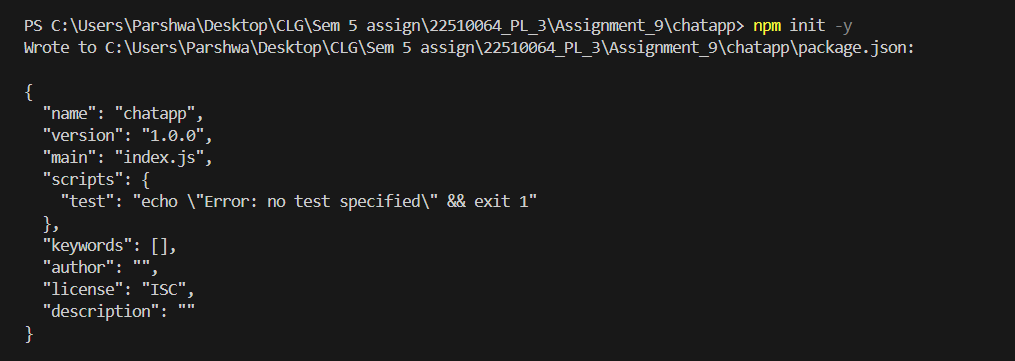
**Technologies:**

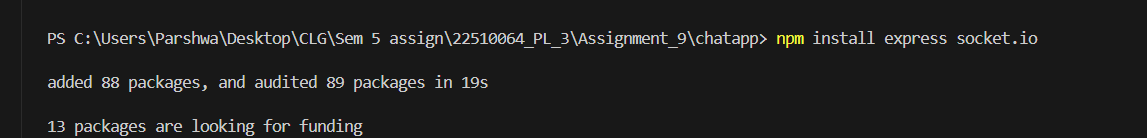
1. **Node.js**: A JavaScript runtime used for building scalable server-side applications.
2. **Express.js**: A lightweight web application framework for Node.js used to manage HTTP requests and responses.
3. **Socket.io**: A library that enables real-time, event-based communication between clients and servers using WebSockets under the hood but also provides fallbacks for older browsers.
4. **HTML/CSS/JavaScript**: Used for the frontend to create the user interface.
5. **WebSockets**: A communication protocol that provides full-duplex communication channels over a single TCP connection, enabling real-time updates.

**Flow of the Application**:

* **Server-Side (Node.js + Express + Socket.io)**: The server is responsible for listening to incoming connections and broadcasting messages to all users. It also manages user connections and disconnections.
* **Client-Side (HTML/JS + Socket.io-client)**: Each client connects to the server via a WebSocket and sends/receives real-time messages.

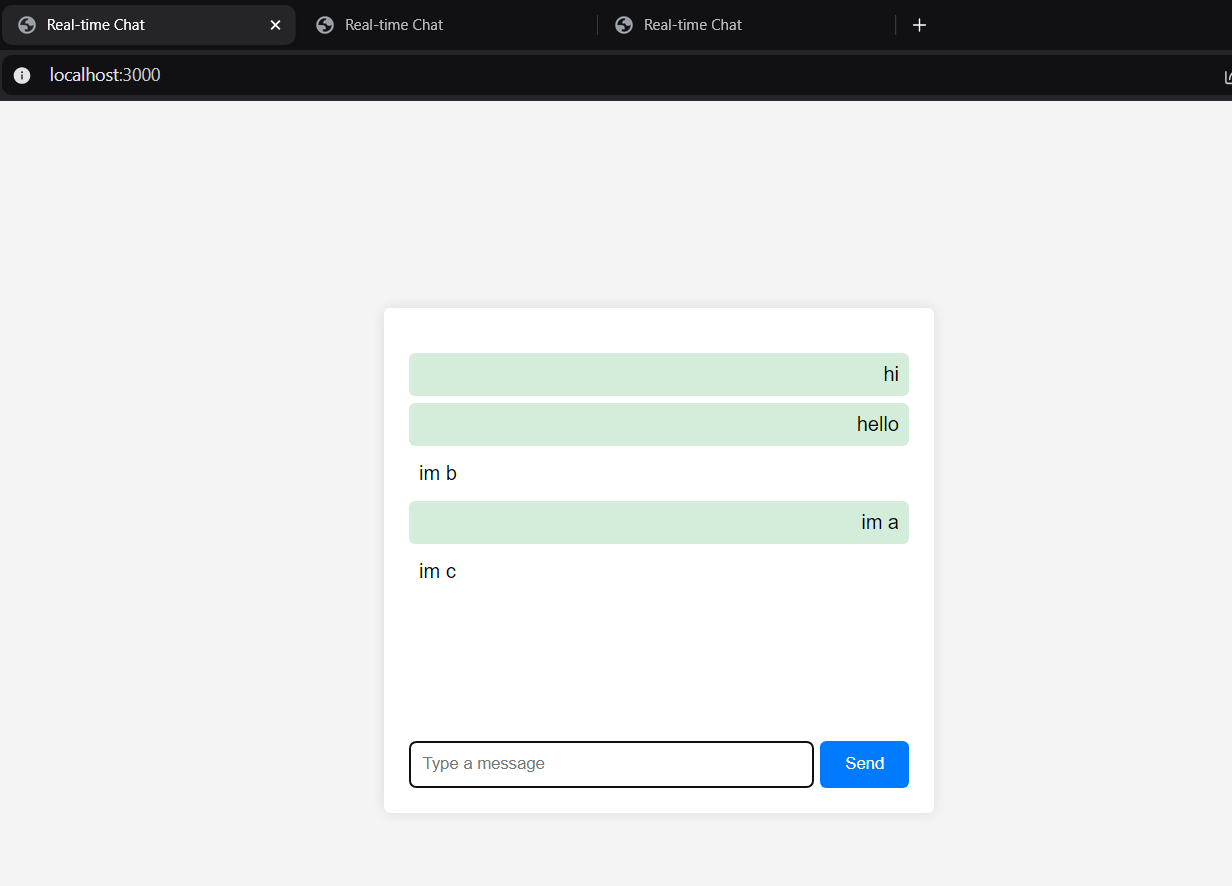
Process:

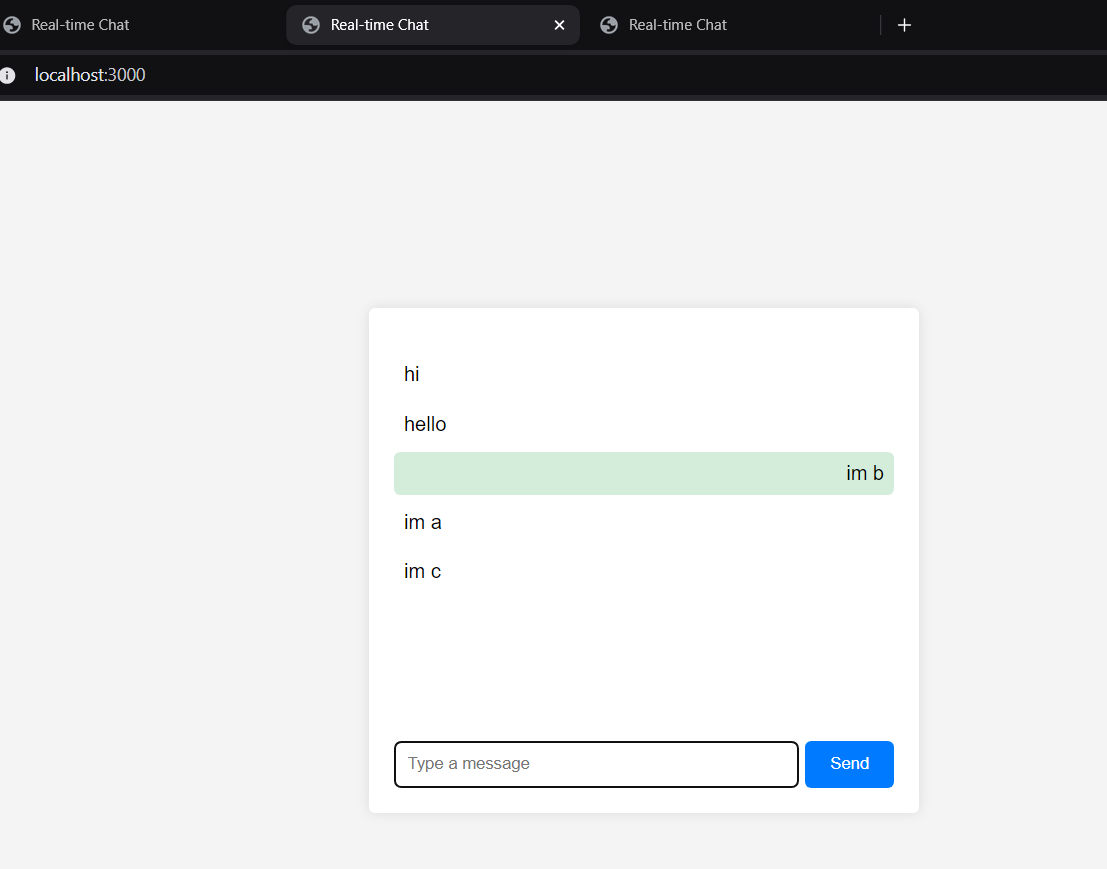


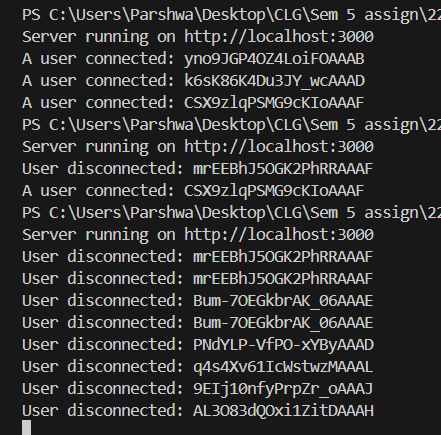


Create a server.js file:  
Then run the server.js file

After the code runs on the localhost  
Open different instances of it to start a grp chat kinda thing, where our sent msg is highlighted as green and from other person white.





  
  
Logs from terminal for different users.

