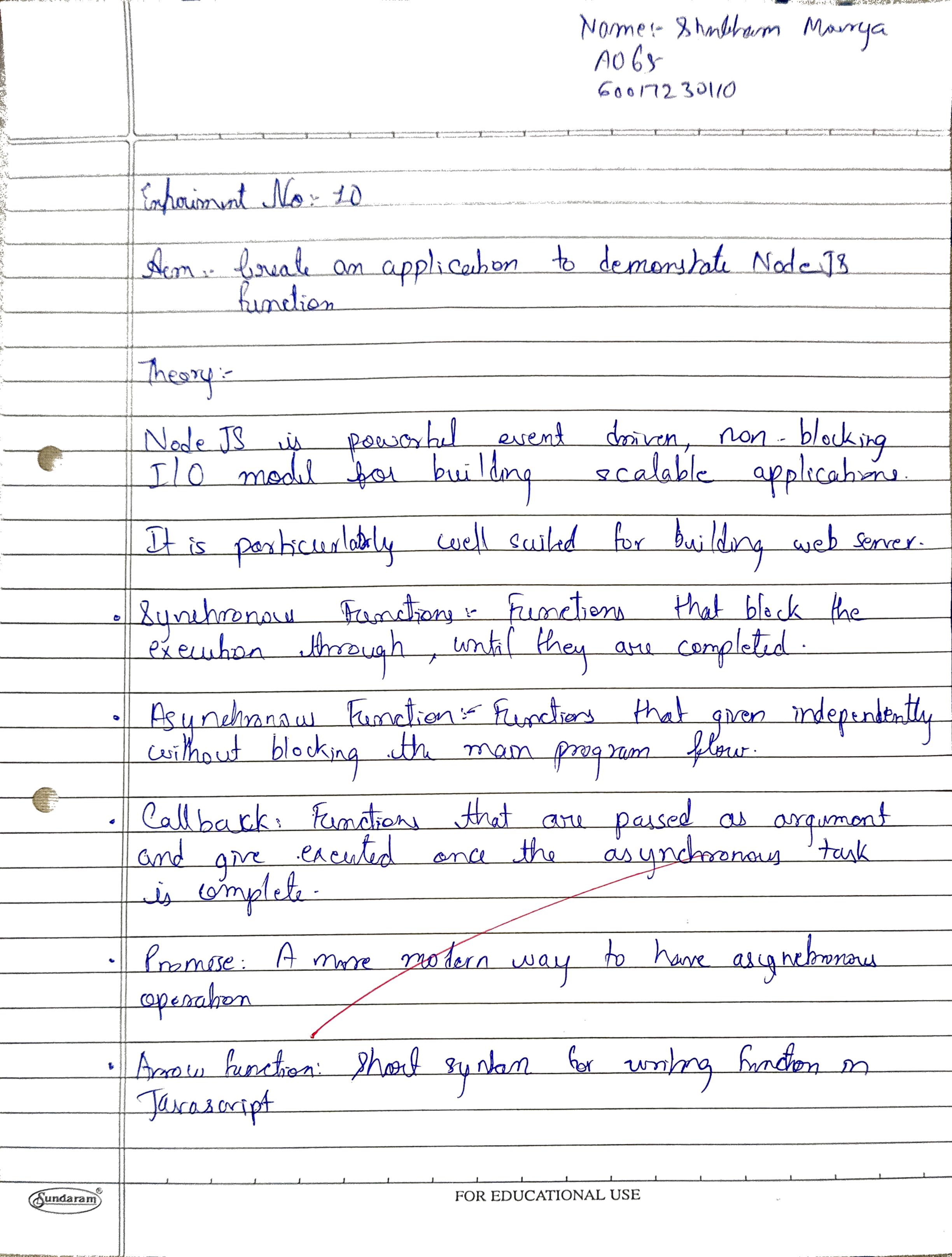
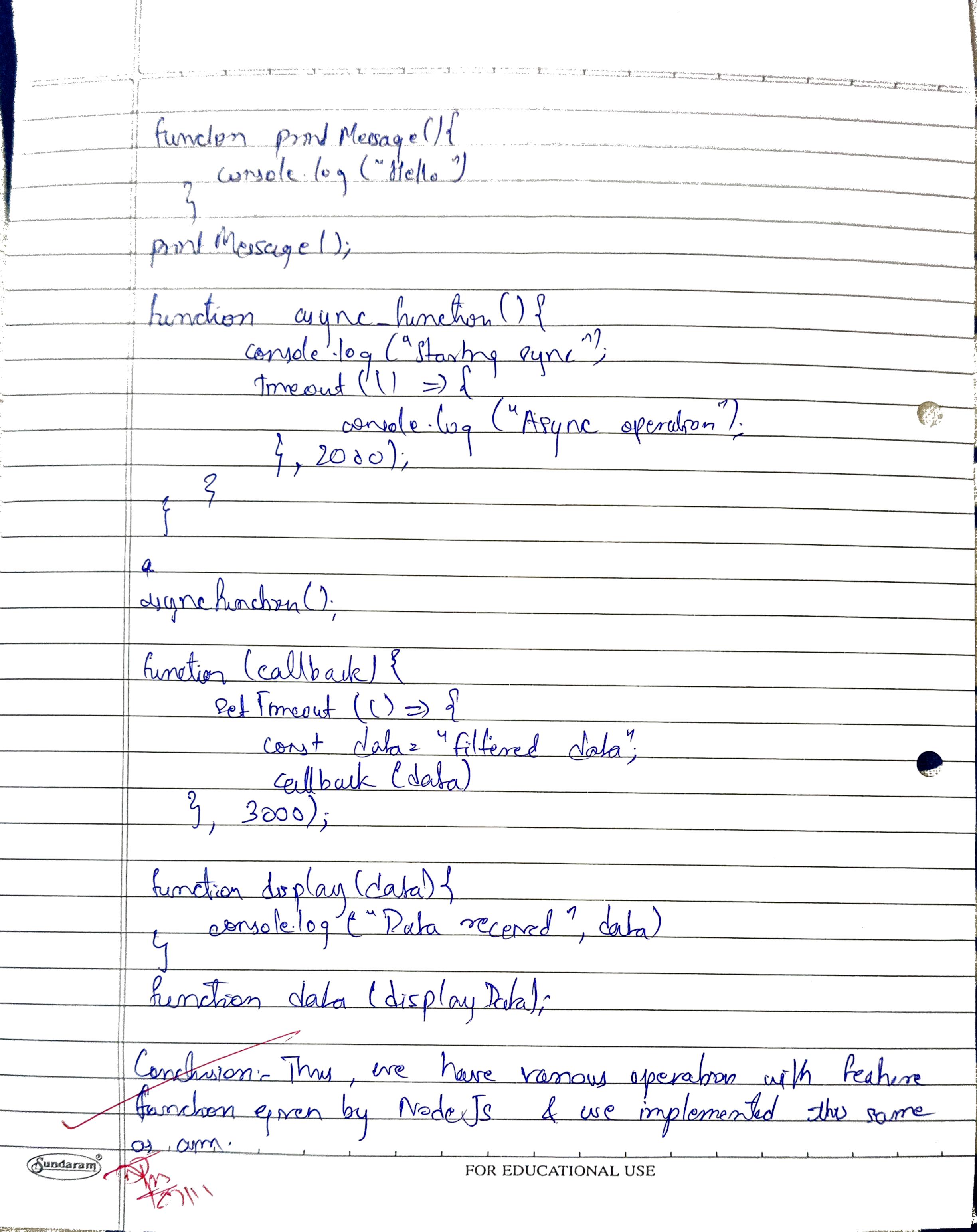
**Experiment 10**

**Name: Shubham Mourya SAP ID: 60017230110**

****

****

|  |  |
| --- | --- |
| Aim | **Create an application to demonstrate Node.js Functions** |
| Software | VS Code |
| Pre-requisite | Active internet connection |
| Code | **server.js**  const express = require('express');  const path = require('path');  const cors = require('cors'); // Import cors  const app = express();  app.use(cors()); // Enable CORS for all routes  app.use(express.json()); // Middleware to parse JSON requests  // Serve the HTML file on the root route  app.get('/', (req, res) => {      res.sendFile(path.join(\_\_dirname, 'index.html'));  });  // Middleware Functions  // Request Logging Middleware (Regular Function)  function logRequest(req, res, next) {      console.log(`${req.method} request to ${req.url}`);      next();  }  app.use(logRequest);  // Error Handling Middleware (Arrow Function)  const errorHandler = (err, req, res, next) => {      console.error('Error:', err.message);      res.status(500).json({ error: 'An error occurred', message: err.message });  };  // Utility Functions  // Formatting Response (Regular Function)  function formatResponse(data) {      return { success: true, timestamp: new Date(), data };  }  // Generating Random Data (Arrow Function)  const generateRandomId = () => Math.floor(Math.random() \* 1000);  // Route Handling Functions  // Sample "database" (array)  let items = [];  // Create Item (POST) - Regular Function  app.post('/items', (req, res) => {      const newItem = { id: generateRandomId(), ...req.body };      items.push(newItem);      res.json(formatResponse(newItem));  });  // Read All Items (GET) - Anonymous Function  app.get('/items', function (req, res) {      res.json(formatResponse(items));  });  // Read Single Item (GET by ID) - Arrow Function  app.get('/items/:id', (req, res) => {      const item = items.find(i => i.id == req.params.id);      if (item) {          res.json(formatResponse(item));      } else {          res.status(404).json({ error: 'Item not found' });      }  });  // Update Item (PUT) - Asynchronous Function with Try-Catch  app.put('/items/:id', async (req, res) => {      try {          const index = items.findIndex(i => i.id == req.params.id);          if (index === -1) throw new Error('Item not found');            items[index] = { ...items[index], ...req.body };          res.json(formatResponse(items[index]));      } catch (error) {          res.status(404).json({ error: error.message });      }  });  // Delete Item (DELETE) - Callback with Promise  app.delete('/items/:id', (req, res) => {      return new Promise((resolve, reject) => {          const index = items.findIndex(i => i.id == req.params.id);          if (index !== -1) {              const deletedItem = items.splice(index, 1);              resolve(formatResponse(deletedItem[0]));          } else {              reject(new Error('Item not found'));          }      })      .then(response => res.json(response))      .catch(error => res.status(404).json({ error: error.message }));  });  // Simulating Asynchronous Operations (Database call simulation)  app.get('/async-data', async (req, res, next) => {      try {          const data = await simulateAsyncDataFetch(); // Mock async function          res.json(formatResponse(data));      } catch (error) {          next(error);      }  });  // Simulate async data fetch (returns data after 2 seconds)  function simulateAsyncDataFetch() {      return new Promise((resolve) => {          setTimeout(() => {              resolve({ message: 'Simulated database data', timestamp: new Date() });          }, 2000);      });  }  // Use error handling middleware  app.use(errorHandler);  // Start the server  const PORT = 3000;  app.listen(PORT, () => console.log(`Server running on port ${PORT}`));  **index.html**  <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Node.js API UI</title>  </head>  <body>      <h1>Node.js API Interaction UI</h1>      <!-- Section for Creating a New Item -->      <h2>Create Item</h2>      <input type="text" id="createName" placeholder="Enter item name">      <button onclick="createItem()">Create Item</button>      <div id="createResult"></div>      <!-- Section for Reading All Items -->      <h2>Get All Items</h2>      <button onclick="getAllItems()">Fetch All Items</button>      <div id="allItems"></div>      <!-- Section for Updating an Item -->      <h2>Update Item</h2>      <input type="number" id="updateId" placeholder="Enter item ID">      <input type="text" id="updateName" placeholder="Enter new item name">      <button onclick="updateItem()">Update Item</button>      <div id="updateResult"></div>      <!-- Section for Deleting an Item -->      <h2>Delete Item</h2>      <input type="number" id="deleteId" placeholder="Enter item ID">      <button onclick="deleteItem()">Delete Item</button>      <div id="deleteResult"></div>      <!-- JavaScript for Making API Calls -->      <script>          const apiBase = 'http://localhost:3000';          // Function to Create an Item          async function createItem() {              const name = document.getElementById('createName').value;              const response = await fetch(`${apiBase}/items`, {                  method: 'POST',                  headers: { 'Content-Type': 'application/json' },                  body: JSON.stringify({ name })              });              const result = await response.json();              document.getElementById('createResult').innerText = JSON.stringify(result);          }          // Function to Get All Items          async function getAllItems() {              const response = await fetch(`${apiBase}/items`);              const items = await response.json();              document.getElementById('allItems').innerText = JSON.stringify(items, null, 2);          }          // Function to Update an Item          async function updateItem() {              const id = document.getElementById('updateId').value;              const name = document.getElementById('updateName').value;              const response = await fetch(`${apiBase}/items/${id}`, {                  method: 'PUT',                  headers: { 'Content-Type': 'application/json' },                  body: JSON.stringify({ name })              });              const result = await response.json();              document.getElementById('updateResult').innerText = JSON.stringify(result);          }          // Function to Delete an Item          async function deleteItem() {              const id = document.getElementById('deleteId').value;              const response = await fetch(`${apiBase}/items/${id}`, {                  method: 'DELETE'              });              const result = await response.json();              document.getElementById('deleteResult').innerText = JSON.stringify(result);          }      </script>  </body>  </html> |
| Result |  |
| Conclusion | Thus, we have successfully performed conditional rendering in a React application by demonstrating how server-side operations can be integrated with client-side interactions. This experiment covered CRUD operations and showcased the use of various function types in Node.js, including regular functions, arrow functions, and async functions. By building a simple UI in HTML, we enabled interaction with a Node.js API to create, read, update, and delete items, effectively simulating a full-stack web application without the need for a database. This practical experience enhances understanding of handling HTTP requests and responses in a web application. |