PROJECT TRAINING WORKSHOP

Node JS

RAAFI

Builds You Best

Node JS

- **What is Node JS?**
  - server-side runtime environment for executing JavaScript code (Runtime + JavaScript)

- **What are the roles of Node JS?**

JavaScript Runtime	Server-Side Programming	Event Driven and Non-Blocking I/O	Scalability
NPM (Node Package Manager)	Single Language (Client and Server)	Rich Community and Ecosystem	Streaming and Data Processing
Cross-Platform	Open Source		

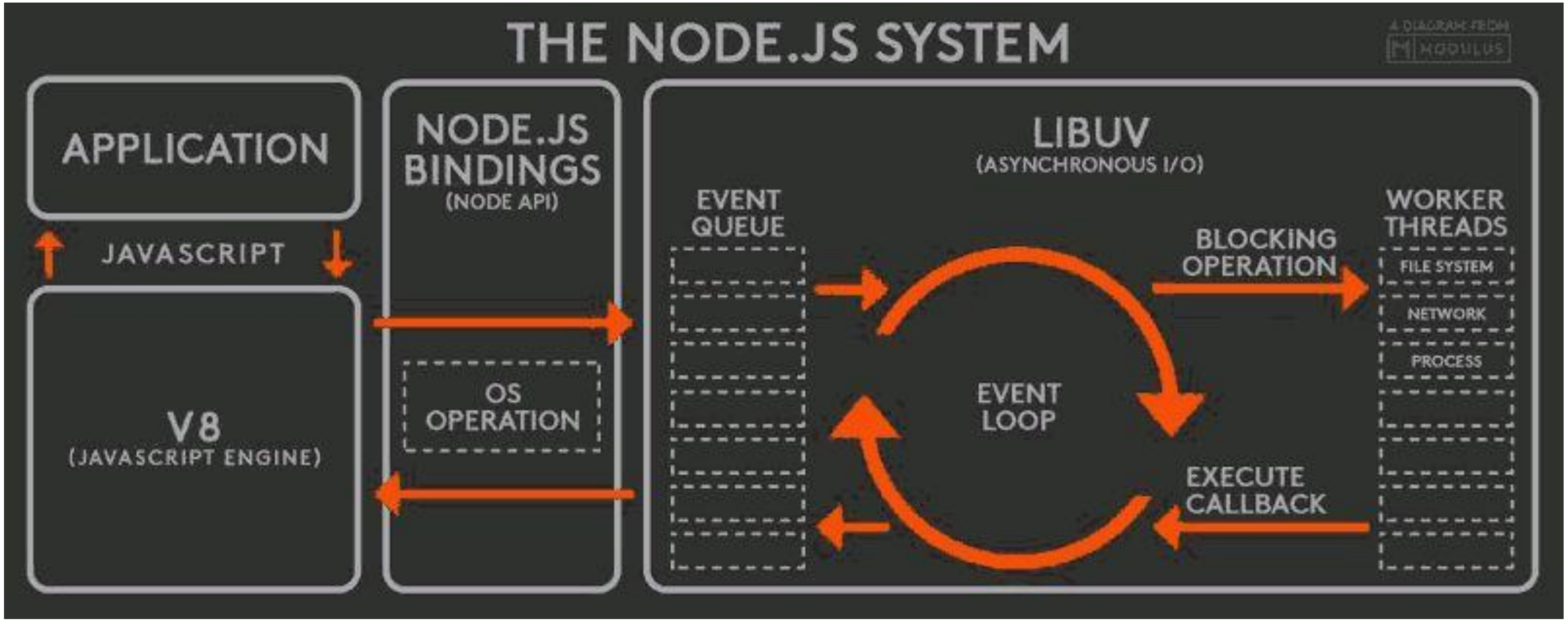
- **What are the roles of Node JS?**
  - **Single Page Applications**
  - **Data Streaming Applications**
  - **Real Time Application (Data Intensive)**
  - **JSON APIs based Applications**
  - **I/O bound Applications**

Node JS

- **What are the key concepts of Node JS?**

V8 JavaScript Engine (google)	Event Loop	Modules ('require')	Node Package Manager (open-source collection)
Asynchronous Programming	Event Emitters (custom event-driven modules)	Streams (files, http requests, data processing)	Built-in HTTP Module (RESTful APIs)
Buffers (Binary Data)	File System Modules (fs)	Promises and Async/Await	Child Processes (parallel code execution)
Global Objects (process, console, require)	Error Handling (try/catch)	Single Threaded	

Node JS ARCHITECTURE



RAAFI
Builds You Best

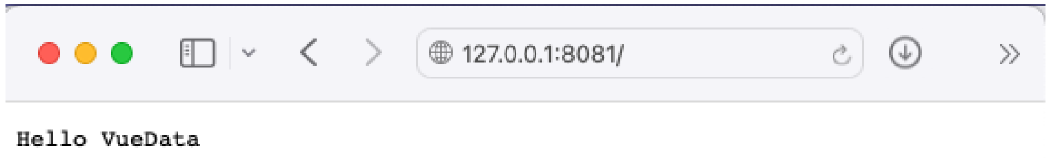
Node JS – Building Blocks of Application

- Entry Point (app.js or index.js)
- Import required modules
- Routing (incoming requests handling)
- Middleware (Authentication and Authorization, Logging or Validations)
- Database Connectivity
- Templates Engine (Optional)
- Error Handling
- Security (input sanitization, XSS and SQL Injection protection
- Caching
- Internationalization and Localization
- Session Management

RAAFI
Builds You Best

Node JS – Running Server

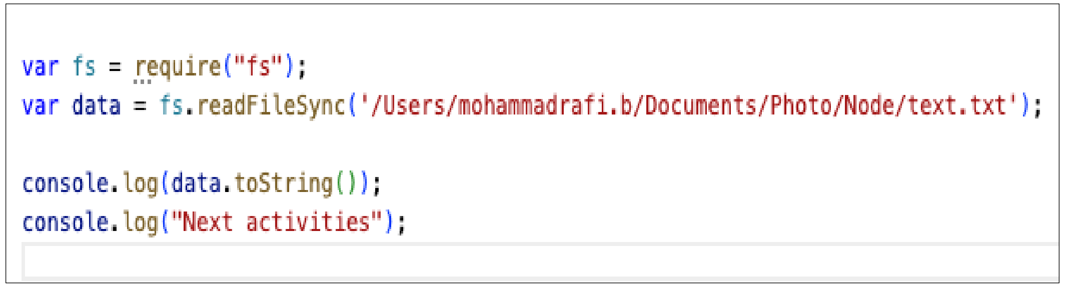
```
var http = require("http");

http.createServer(function (request, response) {
   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body as "Hello World"
   response.end('Hello VueData\n');
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```



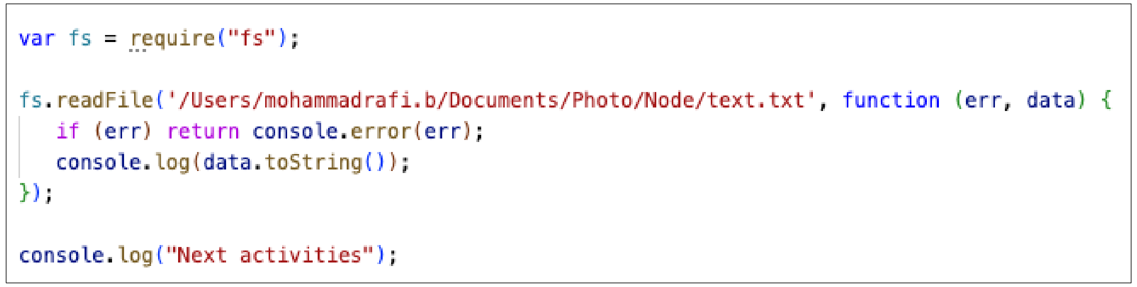
```
Hello VueData
```

RAAFI
Builds You Best

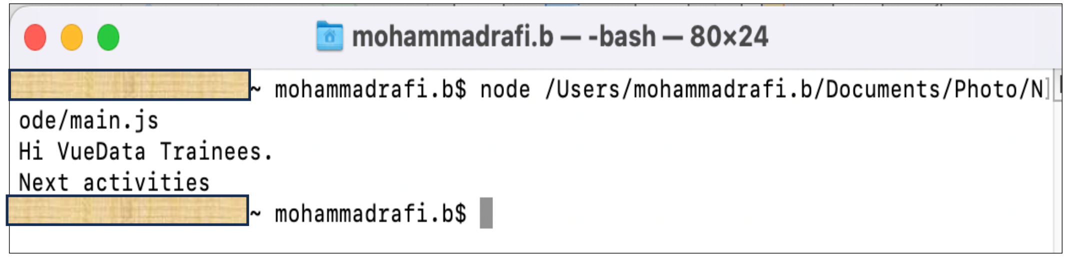
Node JS – Blocking and Non-Blocking



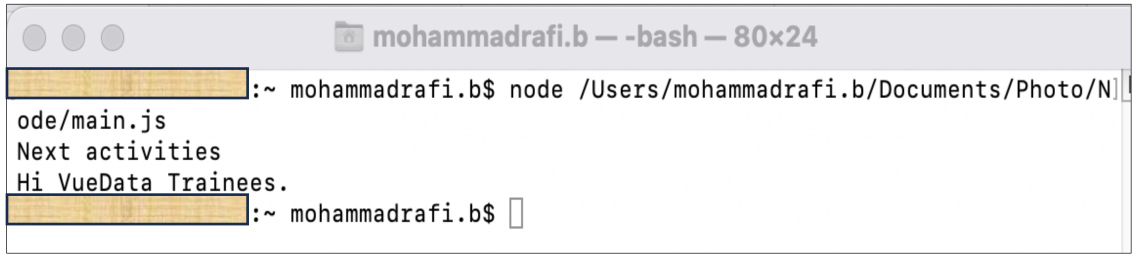
```
var fs = require("fs");
var data = fs.readFileSync('/Users/mohammadrafi.b/Documents/Photo/Node/text.txt');

console.log(data.toString());
console.log("Next activities");
```



```
var fs = require("fs");

fs.readFile('/Users/mohammadrafi.b/Documents/Photo/Node/text.txt', function (err, data) {
   if (err) return console.error(err);
   console.log(data.toString());
});

console.log("Next activities");
```



```
[redacted]~ mohammadrafi.b$ node /Users/mohammadrafi.b/Documents/Photo/N
ode/main.js
Hi VueData Trainees.
Next activities
[redacted]~ mohammadrafi.b$
```



```
[redacted]:~ mohammadrafi.b$ node /Users/mohammadrafi.b/Documents/Photo/N
ode/main.js
Next activities
Hi VueData Trainees.
[redacted]:~ mohammadrafi.b$
```

BLOCKING

NON - BLOCKING

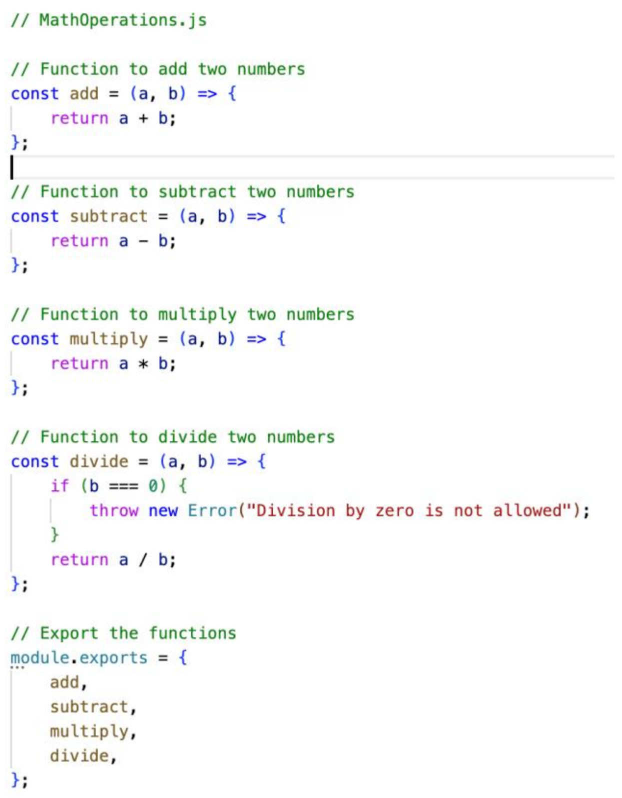
RAAFI
Builds You Best

Node JS – Modules

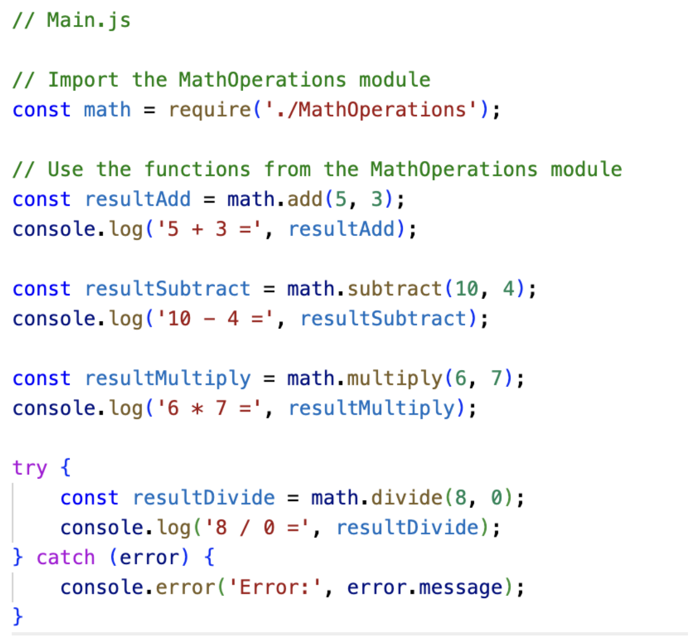
- **What are Modules?**
  - Organized reusable code written for specific purpose.

- **What are the types of Modules?**
  - **Core Modules**
    - **fs (File Stream)**
      - const fs = require('fs');
    - **http**
      - const http = require('http');
    - **https**
      - const https = require('https');
    - **path**
      - const path = require('path');
    - **os (Operating System)**
      - const os = require('os');
    - **Events**
      - const EventEmitter = require('events');
    - **Util**
      - const util = require('util');
    - **Querystring**
      - const querystring = require('querystring');
    - **url**
      - const url = require('url');

User- Defined Modules



```
// MathOperations.js

// Function to add two numbers
const add = (a, b) => {
    return a + b;
};

// Function to subtract two numbers
const subtract = (a, b) => {
    return a - b;
};

// Function to multiply two numbers
const multiply = (a, b) => {
    return a * b;
};

// Function to divide two numbers
const divide = (a, b) => {
    if (b === 0) {
        throw new Error("Division by zero is not allowed");
    }
    return a / b;
};

// Export the functions
module.exports = {
    add,
    subtract,
    multiply,
    divide,
};
```

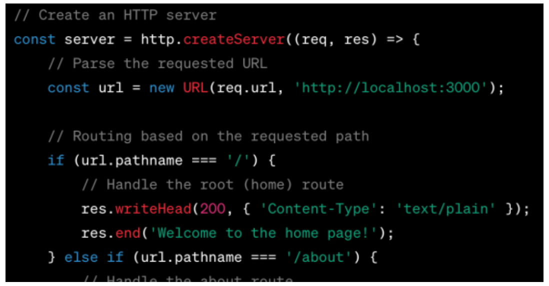


```
// Main.js

// Import the MathOperations module
const math = require('./MathOperations');

// Use the functions from the MathOperations module
const resultAdd = math.add(5, 3);
console.log('5 + 3 =', resultAdd);

const resultSubtract = math.subtract(10, 4);
console.log('10 - 4 =', resultSubtract);

const resultMultiply = math.multiply(6, 7);
console.log('6 * 7 =', resultMultiply);

try {
    const resultDivide = math.divide(8, 0);
    console.log('8 / 0 =', resultDivide);
} catch (error) {
    console.error('Error:', error.message);
}
```

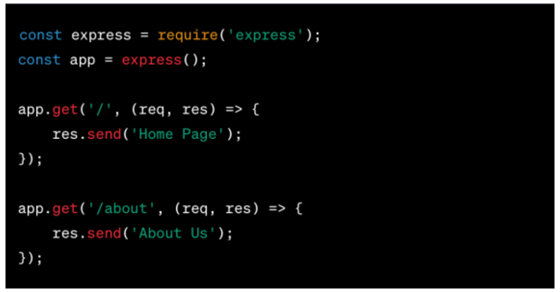
Node JS – Routing

- **What is Routing?**
  - process of directing incoming HTTP requests to specific handlers or controllers based on the requested URL and HTTP method

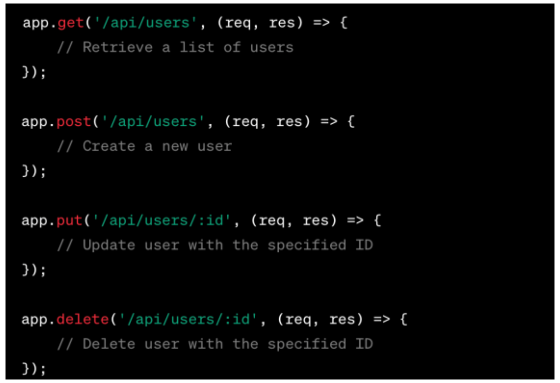
- **What are the types of Routing?**
  - **Basic (URL Parsing) Routing**
  - **Express JS Routing**
  - **RESTful Routing**
  - **Middleware Routing**
  - **Modular Routing**
  - **Controller-Based Routing**



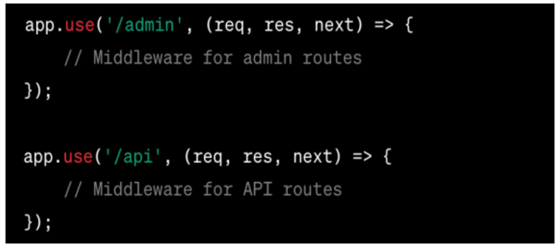
```
// Create an HTTP server
const server = http.createServer((req, res) => {
    // Parse the requested URL
    const url = new URL(req.url, 'http://localhost:3000');

    // Routing based on the requested path
    if (url.pathname === '/') {
        // Handle the root (home) route
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end('Welcome to the home page!');
    } else if (url.pathname === '/about') {
        // Handle the about route
```



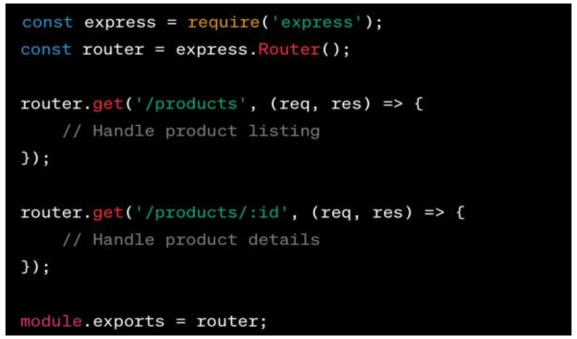
```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('Home Page');
});

app.get('/about', (req, res) => {
    res.send('About Us');
});
```



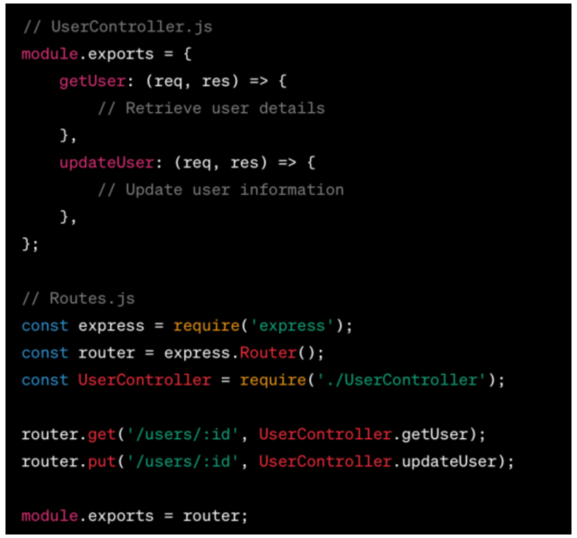
```
app.get('/api/users', (req, res) => {
    // Retrieve a list of users
});

app.post('/api/users', (req, res) => {
    // Create a new user
});

app.put('/api/users/:id', (req, res) => {
    // Update user with the specified ID
});

app.delete('/api/users/:id', (req, res) => {
    // Delete user with the specified ID
});
```



```
app.use('/admin', (req, res, next) => {
    // Middleware for admin routes
});

app.use('/api', (req, res, next) => {
    // Middleware for API routes
});
```



```
const express = require('express');
const router = express.Router();

router.get('/products', (req, res) => {
    // Handle product listing
});

router.get('/products/:id', (req, res) => {
    // Handle product details
});

module.exports = router;
```



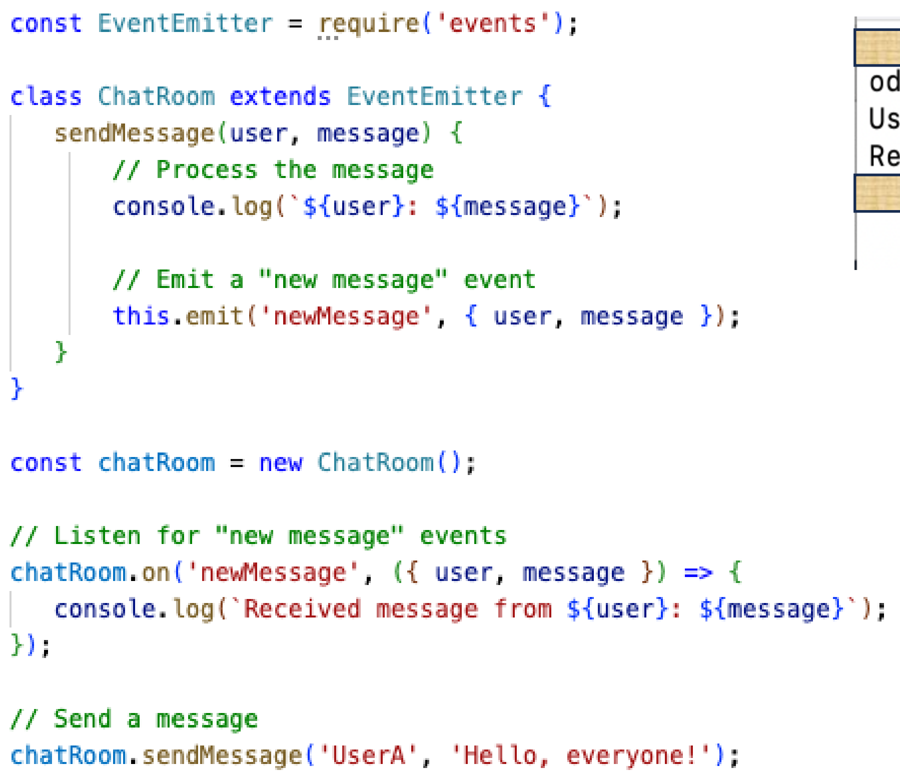
```
// UserController.js
module.exports = {
    getUser: (req, res) => {
        // Retrieve user details
    },
    updateUser: (req, res) => {
        // Update user information
    },
};

// Routes.js
const express = require('express');
const router = express.Router();
const UserController = require('./UserController');

router.get('/users/:id', UserController.getUser);
router.put('/users/:id', UserController.updateUser);

module.exports = router;
```

Node JS – Middleware

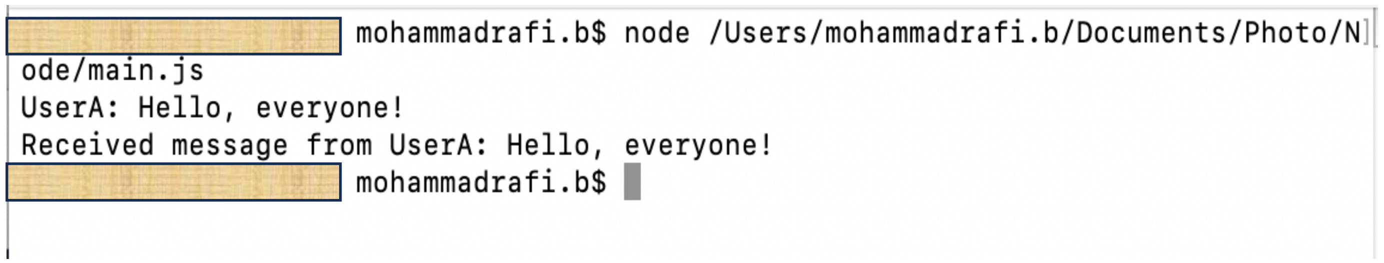
- **What is Middleware?**
  - **allows you to perform tasks and processing in the request-response cycle before it reaches the final route handler**

- **What are the types of Middleware?**
  - **Application-Level Middleware (**Logging middleware, body parsing middleware, authentication middleware.**)**
  - **Route-Specific Middleware (**Authentication middleware for specific routes, route-specific logging middleware**)**
  - **Error-Handling Middleware (**Error logging middleware, custom error-handling middleware**)**
  - **Third-Party Middleware (**Express.js middleware like express-session, passport for authentication**)**
  - **Built-In Middleware (**express.json() for parsing JSON data, express.static() for serving static files.**)**
  - **Custom Middleware (**Custom authentication middleware, middleware for request validation**)**
  - **Logging Middleware (**logs information about incoming requests, such as request method, URL, and timestamp**)**
  - **Authentication Middleware (**verifies the identity of users or clients before allowing access to certain routes or resources**)**
  - **Security Middleware (**adds security features to an application, such as preventing cross-site scripting (XSS) attacks or enforcing HTTPS**)**
  - **CORS Middleware (**handling Cross-Origin Resource Sharing (CORS) to control which domains are allowed to access resources on a web server**)**
  - **Session Middleware (**manages user sessions, often used in web applications to maintain user state between requests**)**
  - **Compression Middleware (**compressing server responses to reduce data transfer size and improve performance**)**
  - **Request Validation Middleware (**validates incoming request data to ensure it meets specific criteria or constraints**)**

Node JS – Event Emitter



```javascript
const EventEmitter = require('events');

class ChatRoom extends EventEmitter {
   sendMessage(user, message) {
       // Process the message
       console.log(`${user}: ${message}`);

       // Emit a "new message" event
       this.emit('newMessage', { user, message });
   }
}

const chatRoom = new ChatRoom();

// Listen for "new message" events
chatRoom.on('newMessage', ({ user, message }) => {
   console.log(`Received message from ${user}: ${message}`);
});

// Send a message
chatRoom.sendMessage('UserA', 'Hello, everyone!');
```



```
[redacted] mohammadrafi.b$ node /Users/mohammadrafi.b/Documents/Photo/N
ode/main.js
UserA: Hello, everyone!
Received message from UserA: Hello, everyone!
[redacted] mohammadrafi.b$ 
```

- addListener(event, listener)
- on(event, listener)
- once(event, listener)
- removeListener(event, listener)
- removeAllListeners([event])
- setMaxListeners(n)
- listeners(event)
- emit(event, [arg1], [arg2], [...])

Node JS – Streams and Pipes

- **What are Streams?**
  - **An object that lets us to read/write/transform data from source to destination.**

- **What are the types of Streams?**

Readable	Writeable	Duplex	Tansform

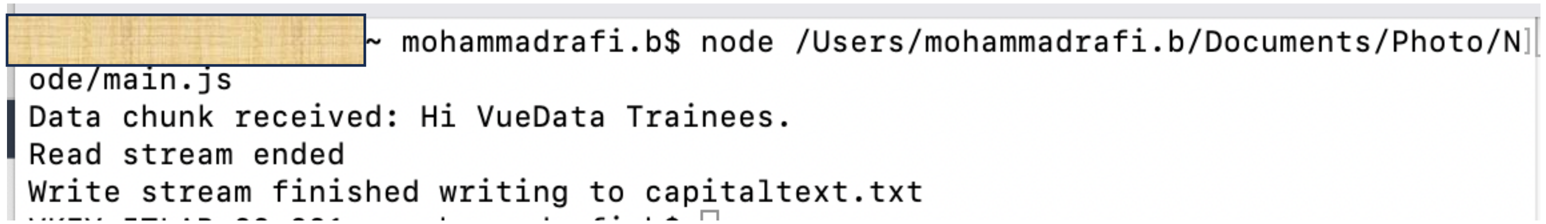
- **What are some common events for Streams?**

Data	End	Error	Finish

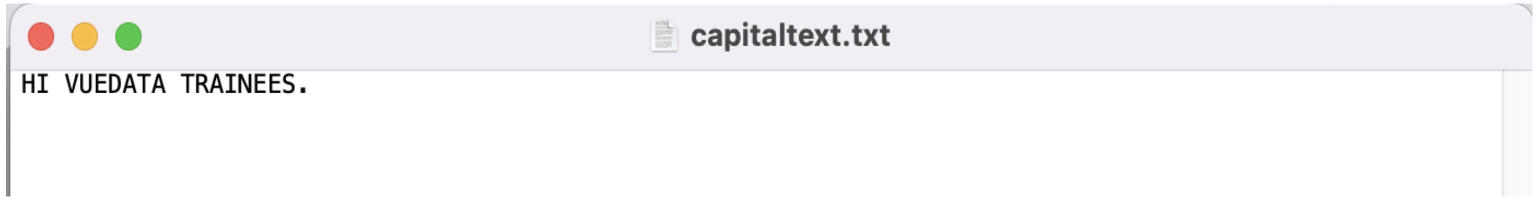
- **What is Pipe?**
  - a mechanism where we provide the output of one stream as the input to another stream
- **What is Pipe Chaining?**
  - a mechanism to connect the output of one stream to another stream and create a chain of multiple stream operations.

Node JS – Streams and Pipes

```javascript
const fs = require('fs');
const { Transform } = require('stream');
// Create a readable stream from the input file
const readStream = fs.createReadStream('/Users/mohammadrafi.b/Documents/Photo/Node/text.txt', 'utf8');
// Create a writable stream to the output file
const writeStream = fs.createWriteStream('/Users/mohammadrafi.b/Documents/Photo/Node/capitaltext.txt', 'utf8');
// Create a transform stream to capitalize each line
const capitalizeStream = new Transform({
   transform(chunk, encoding, callback) {
      const capitalizedChunk = chunk.toString().toUpperCase();
      this.push(capitalizedChunk);
      callback();
   }
});
// Pipe the data through the streams
readStream.pipe(capitalizeStream).pipe(writeStream);

// Event handling
readStream.on('data', (chunk) => {
   console.log('Data chunk received:', chunk);
});
readStream.on('end', () => {
   console.log('Read stream ended');
});
readStream.on('error', (err) => {
   console.error('Read stream error:', err);
});
writeStream.on('finish', () => {
   console.log('Write stream finished writing to capitaltext.txt');
});
writeStream.on('error', (err) => {
   console.error('Write stream error:', err);
});
```



```
[redacted]~ mohammadrafi.b$ node /Users/mohammadrafi.b/Documents/Photo/N
ode/main.js
Data chunk received: Hi VueData Trainees.
Read stream ended
Write stream finished writing to capitaltext.txt
```

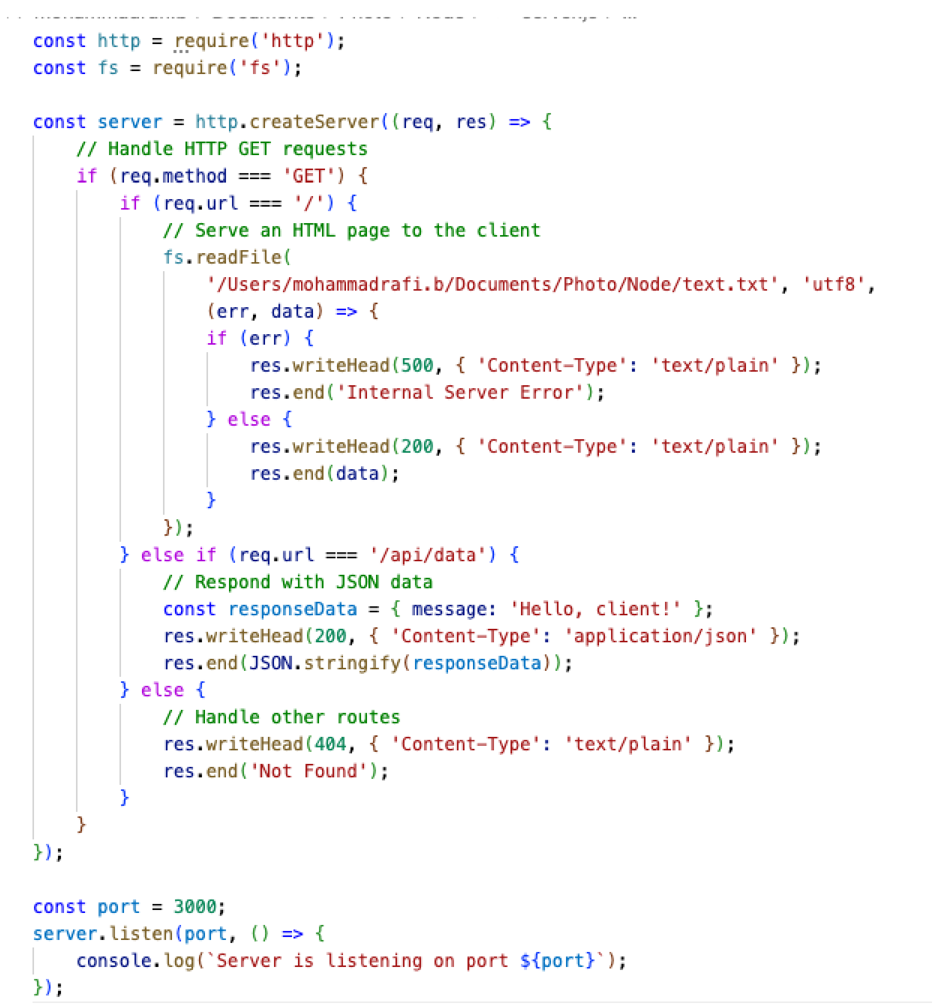


capitaltext.txt

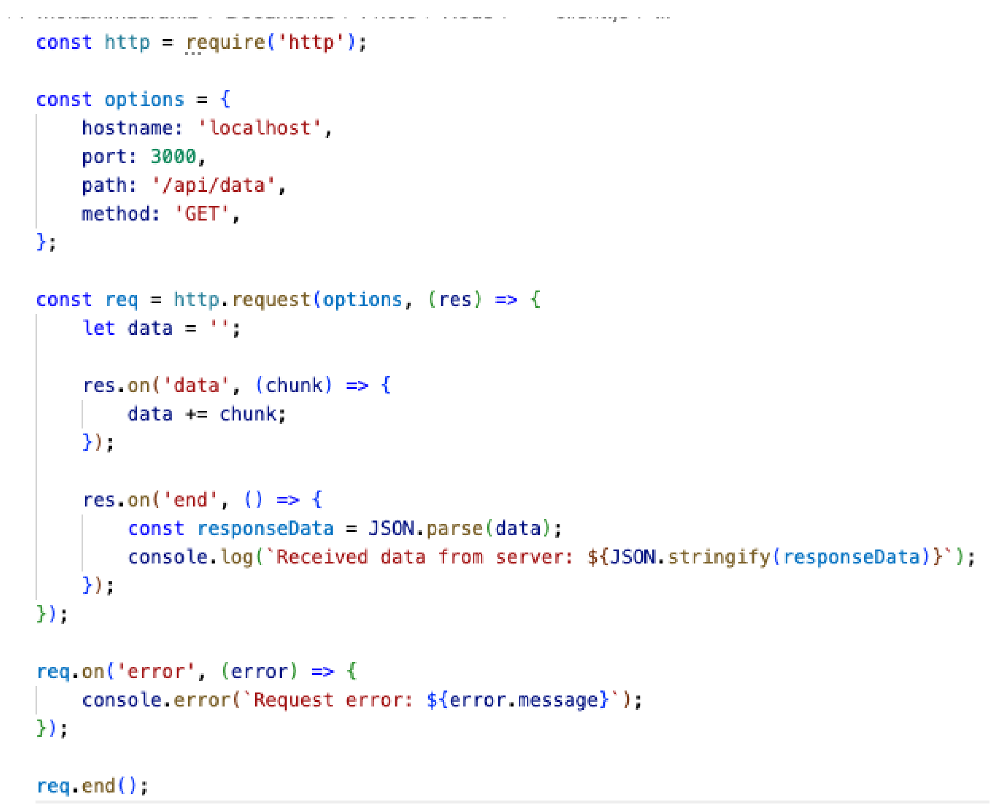
```
HI VUEDATA TRAINEES.
```

RAAFI
Builds You Best

Node JS – Web Server and Web Client



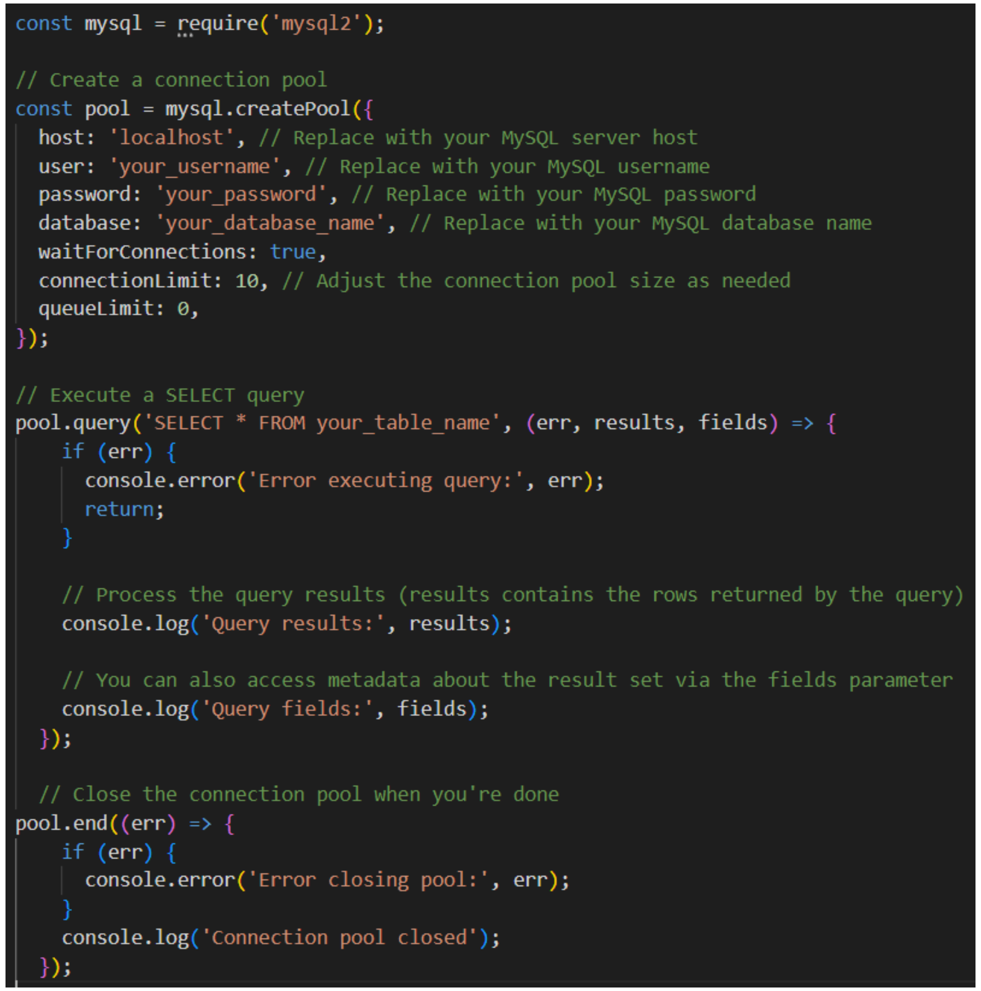
```javascript
const http = require('http');
const fs = require('fs');

const server = http.createServer((req, res) => {
    // Handle HTTP GET requests
    if (req.method === 'GET') {
        if (req.url === '/') {
            // Serve an HTML page to the client
            fs.readFile(
                '/Users/mohammadrafi.b/Documents/Photo/Node/text.txt', 'utf8',
                (err, data) => {
                if (err) {
                    res.writeHead(500, { 'Content-Type': 'text/plain' });
                    res.end('Internal Server Error');
                } else {
                    res.writeHead(200, { 'Content-Type': 'text/plain' });
                    res.end(data);
                }
            });
        } else if (req.url === '/api/data') {
            // Respond with JSON data
            const responseData = { message: 'Hello, client!' };
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(responseData));
        } else {
            // Handle other routes
            res.writeHead(404, { 'Content-Type': 'text/plain' });
            res.end('Not Found');
        }
    }
});

const port = 3000;
server.listen(port, () => {
    console.log(`Server is listening on port ${port}`);
});
```



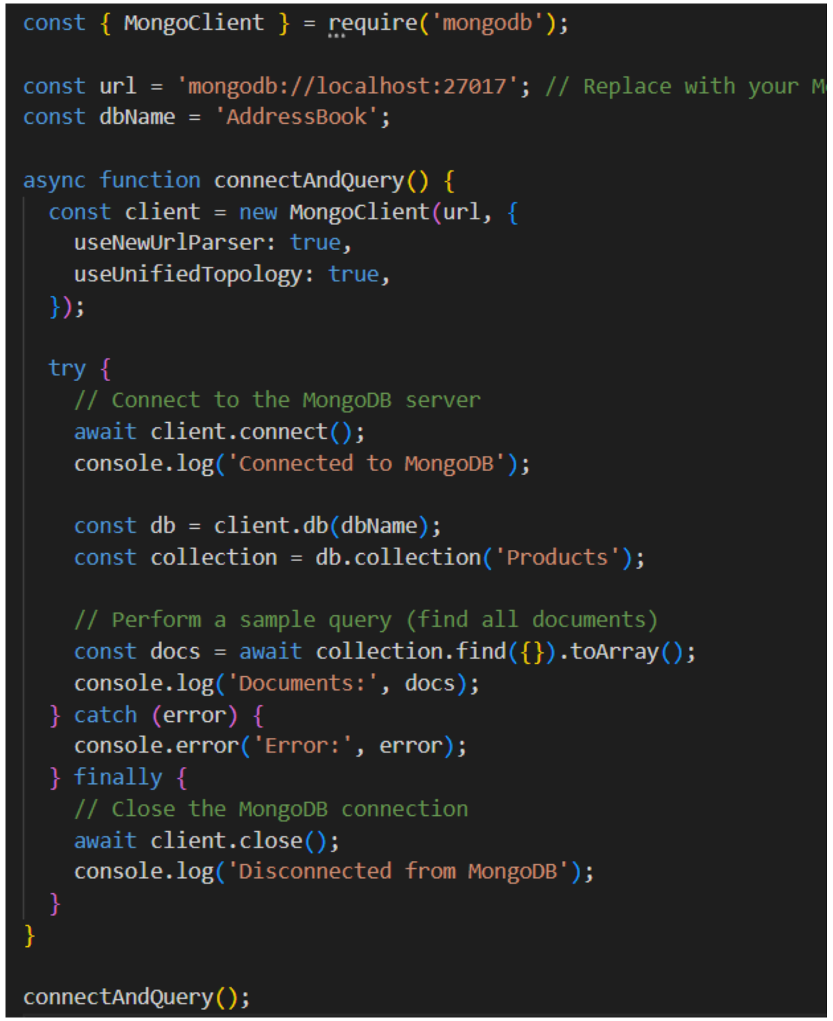
```javascript
const http = require('http');

const options = {
    hostname: 'localhost',
    port: 3000,
    path: '/api/data',
    method: 'GET',
};

const req = http.request(options, (res) => {
    let data = '';

    res.on('data', (chunk) => {
        data += chunk;
    });

    res.on('end', () => {
        const responseData = JSON.parse(data);
        console.log(`Received data from server: ${JSON.stringify(responseData)}`);
    });
});

req.on('error', (error) => {
    console.error(`Request error: ${error.message}`);
});

req.end();
```

RAAFI
Builds You Best

Node JS – Database Connectivity



```
const mysql = require('mysql2');

// Create a connection pool
const pool = mysql.createPool({
  host: 'localhost', // Replace with your MySQL server host
  user: 'your_username', // Replace with your MySQL username
  password: 'your_password', // Replace with your MySQL password
  database: 'your_database_name', // Replace with your MySQL database name
  waitForConnections: true,
  connectionLimit: 10, // Adjust the connection pool size as needed
  queueLimit: 0,
});

// Execute a SELECT query
pool.query('SELECT * FROM your_table_name', (err, results, fields) => {
    if (err) {
      console.error('Error executing query:', err);
      return;
    }

    // Process the query results (results contains the rows returned by the query)
    console.log('Query results:', results);

    // You can also access metadata about the result set via the fields parameter
    console.log('Query fields:', fields);
  });

  // Close the connection pool when you're done
pool.end((err) => {
    if (err) {
      console.error('Error closing pool:', err);
    }
    console.log('Connection pool closed');
  });
```



```
const { MongoClient } = require('mongodb');

const url = 'mongodb://localhost:27017'; // Replace with your M
const dbName = 'AddressBook';

async function connectAndQuery() {
  const client = new MongoClient(url, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });

  try {
    // Connect to the MongoDB server
    await client.connect();
    console.log('Connected to MongoDB');

    const db = client.db(dbName);
    const collection = db.collection('Products');

    // Perform a sample query (find all documents)
    const docs = await collection.find({}).toArray();
    console.log('Documents:', docs);
  } catch (error) {
    console.error('Error:', error);
  } finally {
    // Close the MongoDB connection
    await client.close();
    console.log('Disconnected from MongoDB');
  }
}

connectAndQuery();
```

RAAFI
Builds You Best