**B.E. / B.Tech. PRACTICAL END SEMESTER EXAMINATIONS, OCT 2024**

**Third Semester**

# CS3303 – WEB DEVELOPMENT FRAMEWORKS

**(Regulations – 2022R)**

**Answer any one Question**

**Time: 3 Hours** **Max. Marks 100**

| Aim and Procedure | Program | Output & Result | Viva-Voce | Record | Total |
|---|---|---|---|---|---|
| 10 | 40 | 30 | 10 | 10 | 100 |

1. i) Write a node.js program for the following
   a) Display a message in the browser
   b) Display the current date and time in the browser
   c) Pass a query string in URL
   ii) Write a mongoDB command to insert multiple documents into the students collection.

2. i) Write a node.js program for the following
   a) Splits up a web address into readable parts
   b) Display customized error message
   ii) Write mongoDB commands for the following
   a) How do you find a document where the name is "John" in the users collection?
   b) How would you find all documents in the products collection where the price is greater than 100?

3. i) Write a node.js program for the following
   a) Fileupload operation
   b) Event handling
   ii) Write mongoDB commands for the following
   a) Update the age of a student named "Alice" to 25 in the students collection
   b) update the status field to "active" for all documents in the employees collection

4. i) Write an Express.js program for the following
   a) Implementation of 2 middlewares
   b) Filtering paths using URL prefix
   ii) Write mongoDB commands for the following
   a) How do you delete a single document where the email is "jane@example.com" in the contacts collection?
   b) Write a MongoDB command to delete all documents in the orders collection where status is "canceled".

5. i) Write an Express.js program for the following
   a) Setting the status code
   b) Parsing data from request
   ii) Write mongoDB commands for the following
   a) Write a query to find all documents in the students collection where age is between 18 and 25.
   b) Write a MongoDB query to find all documents in the books collection that contain the word "MongoDB" in the title field.

6. i) Write an Express.js program for the following
    a) Setting the status code
    b) Parsing data from request
  ii) Write mongoDB commands for the following
    a) Update the age of a student named "Alice" to 25 in the students collection
    b) update the status field to "active" for all documents in the employees collection

7. i) Write an Express.js program for the following
    a) Implementation of 2 middlewares
    b) Filtering paths using URL prefix
  ii) Write mongoDB commands for the following
    a) Update the age of a student named "Alice" to 25 in the students collection
    b) update the status field to "active" for all documents in the employees collection

8. i) Write a node.js program for the following
    a) Fileupload operation
    b) Event handling
  ii) Write mongoDB commands for the following
    a) Write a query to find all documents in the students collection where age is between 18 and 25.
    b) Write a MongoDB query to find all documents in the books collection that contain the word "MongoDB" in the title field.

9. i) Write a node.js program for the following
    a) Splits up a web address into readable parts
    b) Display customized error message
  ii) Write mongoDB commands for the following
    a) Write a MongoDB command to find the first 5 students sorted by age in descending order.
    b) Write a MongoDB query to find all documents in the books collection that contain the word "MongoDB" in the title field.

10. i) Write a node.js program for the following
    a) Display a message in the browser
    b) Display the current date and time in the browser
    c) Pass a query string in URL
  ii) Write mongoDB commands for the following
    a) How do you update the email of a specific user in the customers collection where the customerID is 12345?
    b) Write a MongoDB command to add a status field with the value "inactive" to all documents in the users collection.?

11. i) Write an Express.js program for the following
    a) Setting the status code
    b) Parsing data from request
  ii) Write mongoDB commands for the following
    a) Write a command to insert a new employee record into the employees collection with the following fields: name "John Doe", age 28, and department "HR".
    b) Write a MongoDB command to insert multiple documents into the products collection, each having fields productName, price, and category.

12. i) Write an Express.js program for the following
    a) Implementation of 2 middlewares
    b) Filtering paths using URL prefix
  ii) Write mongoDB commands for the following
    a) How do you find all documents in the users collection where the age is greater than 25?
    b) Write a command to find a document in the customers collection where the customerID is 1001 and only return the name and email fields.

13. i) Write a node.js program for the following
    a) Fileupload operation
    b) Event handling
  ii) Write mongoDB commands for the following
    a) Write a MongoDB query to retrieve all products in the inventory collection where quantity is between 10 and 100, inclusive.
    b) Write a command to update the price of a product where the productName is "Laptop" to 1200 in the products collection.

14. i) Write a node.js program for the following
    a) Splits up a web address into readable parts
    b) Display customized error message
  ii) Write mongoDB commands for the following
    a) How do you update the status of all orders in the orders collection where the status is "pending" to "shipped"?
    b) Write a command to increment the quantitySold field by 5 for all documents in the sales collection where productID is 2001.

.

15. i) Write a node.js program for the following
    a) Display a message in the browser
    b) Display the current date and time in the browser
    c) Pass a query string in URL
  ii) Write mongoDB commands for the following
    a) Write a command to delete a single document from the employees collection where the employeeID is 101.
    b) Write a MongoDB command to delete all documents from the products collection where the category is "obsolete".

16. i) Write a node.js program for the following
    a) Display a message in the browser
    b) Display the current date and time in the browser
    c) Pass a query string in URL
  ii) Write mongoDB commands for the following
    a) How do you delete all documents from the logs collection?
    b) Write a command to remove a document from the orders collection where the orderID is 555, if it exists.

17. i) Write a node.js program for the following
    a) Splits up a web address into readable parts
    b) Display customized error message
  ii) Write mongoDB commands for the following
    a) Write a query to retrieve the first 5 documents in the students collection, sorted by age in descending order.

b) Update the status field to "closed" and add a closingDate field for all documents in the projects collection where dueDate is less than the current date.

18. i) Write a node.js program for the following
    a) Fileupload operation
    b) Event handling
    ii) Write mongoDB commands for the following
    a) Write a MongoDB query to update a product's quantity by 10, and then delete the product if its quantity becomes greater than 1000.

19. i) Write an Express.js program for the following
    a) Implementation of 2 middlewares
    b) Filtering paths using URL prefix
    ii) Write mongoDB commands for the following
    a) Insert a document into the students collection with the fields name, age, grade, and subjects where subjects is an array containing "Math" and "Science".
    b) Insert multiple documents into the orders collection with fields orderID, product, quantity, and status.

20. i) Write an Express.js program for the following
    a) Setting the status code
    b) Parsing data from request
    ii) Write mongoDB commands for the following
    a) Find all documents in the employees collection where department is "IT" and age is less than 40.
    b) Write a query to return only the name and salary of employees in the employees collection where the salary is greater than 5000.

# Answers:

**1)**

**i) Node.js Program**

**a) Display a message in the browser:**

1. Install Node.js if you haven't.
2. Create a file called server.js.
3. Add the following code to display a message in the browser.

javascript
Copy code

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

// Create a server object
http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' }); // Set content type to HTML
  res.write('Hello! Welcome to my website.'); // Write a response to the client
  res.end(); // End the response
}).listen(8080); // The server object listens on port 8080

console.log('Server running at http://localhost:8080/');
```

- Run the server with: node server.js
- Visit http://localhost:8080/ in your browser, and you'll see the message "Hello! Welcome to my website."

**b) Display the current date and time in the browser:**

1. Modify the same server.js to include the current date and time.

javascript
Copy code

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

// Create a server object
http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' }); // Set content type to HTML
  const currentDateTime = new Date();
  res.write(`Current Date and Time: ${currentDateTime}`); // Display current date and time
  res.end(); // End the response
}).listen(8080); // The server object listens on port 8080

console.log('Server running at http://localhost:8080/');
```

- Visit http://localhost:8080/ in your browser, and you will see the current date and time displayed.

**c) Pass a query string in the URL:**

To handle query strings in Node.js, you can use the url module. Here's how to modify the program:

javascript
Copy code

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

// Create a server object
http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' }); // Set content type to HTML
  const queryObject = url.parse(req.url, true).query; // Get the query string as an object

  if (queryObject.name) {
    res.write(`Hello, ${queryObject.name}!`); // Respond with name from query string
  } else {
    res.write('Please provide a name in the query string. Example:
http://localhost:8080/?name=John');
  }

  res.end(); // End the response
}).listen(8080); // The server object listens on port 8080

console.log('Server running at http://localhost:8080/');
```

- Run the server and visit http://localhost:8080/?name=John. You should see "Hello, John!" displayed in the browser.

**ii) MongoDB Command to Insert Multiple Documents into the Students Collection**

To insert multiple documents into the students collection in MongoDB, you use the insertMany() function.

Here's the MongoDB command:

javascript
Copy code

```
db.students.insertMany([
  { name: "Alice", age: 21, major: "Computer Science" },
  { name: "Bob", age: 22, major: "Mechanical Engineering" },
  { name: "Carol", age: 23, major: "Physics" },
  { name: "David", age: 24, major: "Mathematics" }
])
```

This command will insert four documents into the students collection with fields like name, age, and major.

**2)**

**i) Node.js Program**

### *a) Splitting up a web address into readable parts*

To parse and split a URL, you can use Node.js's built-in url module. Here's how you can split a web address into readable parts:

javascript
Copy code

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

// Create a server object
http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });

  // Parse the request URL
  const parsedUrl = url.parse(req.url, true);

  // Split the URL into readable parts
  const urlParts = `
    Hostname: ${parsedUrl.hostname}<br>
    Pathname: ${parsedUrl.pathname}<br>
    Search: ${parsedUrl.search}<br>
    Query Object: ${JSON.stringify(parsedUrl.query)}<br>
  `;

  res.write(urlParts); // Display the split-up URL parts
  res.end();
}).listen(8080);

console.log('Server running at http://localhost:8080/');
```

- Run the server and visit something like http://localhost:8080/path?name=John&age=30. You'll see the hostname, pathname, query string, and query object broken down into readable parts.

### *b) Display customized error messages*

To display customized error messages in Node.js, you can modify the response based on the status code or conditions. Here's an example:

javascript
Copy code

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

// Create a server object
```

```
http.createServer((req, res) => {
  res.writeHead(404, { 'Content-Type': 'text/html' }); // Simulate a 404 Not Found error
  res.write('Oops! The page you are looking for does not exist.'); // Custom error message
  res.end();
}).listen(8080);

console.log('Server running at http://localhost:8080/');
```

- When you visit http://localhost:8080/, it will display a custom 404 error message saying, "Oops! The page you are looking for does not exist."

**ii) MongoDB Commands**

***a) Find a document where the name is "John" in the users collection***

To find a specific document where the name is "John", you can use the findOne() function:

javascript
Copy code
```
db.users.findOne({ name: "John" })
```

This command will return the first document in the users collection where the name field is "John."

***b) Find all documents in the products collection where the price is greater than 100***

To find documents with a price greater than 100, you can use the find() function with a condition:

javascript
Copy code
```
db.products.find({ price: { $gt: 100 } })
```

This command will return all documents in the products collection where the price field is greater than 100 ($gt stands for "greater than").

**3)**

**i) Node.js Program**

***a) File Upload Operation***

To handle file uploads in Node.js, you can use the formidable package, which simplifies the process of handling form data.

First, you need to install the formidable package by running:

bash
Copy code

npm install formidable

Here is a Node.js program for file upload:

javascript
Copy code

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

// Create a server object
http.createServer((req, res) => {
  if (req.url === '/fileupload' && req.method.toLowerCase() === 'post') {
    // Parse the incoming form with formidable
    const form = new formidable.IncomingForm();
    form.parse(req, (err, fields, files) => {
      if (err) {
        res.writeHead(500, { 'Content-Type': 'text/html' });
        res.write('Error uploading file');
        return res.end();
      }

      // File upload path (temporary)
      const oldPath = files.fileToUpload.path;
      const newPath = __dirname + '/' + files.fileToUpload.name; // Save file in the server
directory

      // Move the file to the desired location
      fs.rename(oldPath, newPath, (err) => {
        if (err) throw err;
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('File uploaded and moved!');
        res.end();
      });
    });
  } else {
    // Display a file upload form
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<form action="/fileupload" method="post" enctype="multipart/form-data">');
    res.write('<input type="file" name="fileToUpload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);

console.log('Server running at http://localhost:8080/');
```

- Run the server with node server.js.
- Navigate to http://localhost:8080/ in your browser. You'll see a file upload form. Select a file, and it will upload and save to the same directory.

### *b) Event Handling*

Node.js has a built-in events module, which allows you to create, fire, and listen for custom events.

Here is an example of basic event handling:

javascript
Copy code
```
const events = require('events');
const eventEmitter = new events.EventEmitter();

// Event handler
const eventHandler = () => {
  console.log('An event was handled!');
};

// Bind the event handler to the 'myevent' event
eventEmitter.on('myevent', eventHandler);

// Fire the 'myevent' event
eventEmitter.emit('myevent');
```

In this example, an event myevent is emitted, and the eventHandler is executed when the event is triggered.

- To run, save it as event.js and run node event.js in your terminal.

**ii) MongoDB Commands**

### *a) Update the age of a student named "Alice" to 25 in the students collection*

You can use the updateOne() command to update a single document in MongoDB:

javascript
Copy code
```
db.students.updateOne(
  { name: "Alice" },       // Filter: find the document where the name is "Alice"
  { $set: { age: 25 } }    // Update: set the age to 25
)
```

This command finds the document where name is "Alice" and updates the age field to 25.

### *b) Update the status field to "active" for all documents in the employees collection*

To update multiple documents in MongoDB, you can use the updateMany() function:

javascript
Copy code
```
db.employees.updateMany(
  {},                      // No filter means it will apply to all documents
```

```
  { $set: { status: "active" } }  // Set the status field to "active" for all documents
)
```

This command will update the status field to "active" for every document in the employees collection.

**4)**

**i) Express.js Program**

***a) Implementation of 2 Middlewares***

Middleware functions in Express.js are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.

Here's an Express.js program implementing two middlewares:

1. **Middleware 1**: Logs the request method and URL.
2. **Middleware 2**: Adds a timestamp to the request.

javascript
Copy code

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

// Middleware 1: Logs request method and URL
const logRequest = (req, res, next) => {
  console.log(`Method: ${req.method}, URL: ${req.url}`);
  next(); // Call the next middleware or route handler
};

// Middleware 2: Adds a timestamp to the request
const addTimestamp = (req, res, next) => {
  req.requestTime = new Date().toISOString();
  next();
};

// Apply the middlewares to all routes
app.use(logRequest);
app.use(addTimestamp);

// Route to handle requests
app.get('/', (req, res) => {
  res.send(`Hello, World! Request received at: ${req.requestTime}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

- To run the program, save it as app.js, install Express (npm install express), and run node app.js.
- Navigate to http://localhost:3000/, and you will see the timestamp printed along with the "Hello, World!" message. Check your terminal for the logged request method and URL.

### *b) Filtering Paths Using URL Prefix*

Express allows you to define routes that use URL prefixes. This is helpful when you want to filter or group routes based on the prefix.

Here's an example where routes are filtered based on a URL prefix /api:

javascript
Copy code

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

// Middleware to log the API path
app.use('/api', (req, res, next) => {
  console.log(`Accessing API path: ${req.url}`);
  next();
});

// Route for /api/users
app.get('/api/users', (req, res) => {
  res.send('User list');
});

// Route for /api/products
app.get('/api/products', (req, res) => {
  res.send('Product list');
});

// Route outside the /api prefix
app.get('/', (req, res) => {
  res.send('Welcome to the main page');
});

// Start the server
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

- When you visit http://localhost:3000/api/users or http://localhost:3000/api/products, you will see a message, and the API path will be logged in the console.
- Visiting http://localhost:3000/ will show a different message as it is outside the /api prefix.

**ii) MongoDB Commands**

### *a) Delete a Single Document Where the Email Is "jane@example.com" in the contacts Collection*

To delete a single document based on a condition, use the deleteOne() command:

javascript
Copy code
```
db.contacts.deleteOne({ email: "jane@example.com" })
```

This command will find the first document in the contacts collection where the email is "jane@example.com" and delete it.

### *b) Delete All Documents in the orders Collection Where the Status Is "canceled"*

To delete multiple documents based on a condition, use the deleteMany() command:

javascript
Copy code
```
db.orders.deleteMany({ status: "canceled" })
```

This command will delete all documents in the orders collection where the status field is "canceled".

**5)**

**i) Express.js Program**

### *a) Setting the Status Code*

In Express.js, you can set the HTTP status code using the res.status() method. Here's an example of how to set the status code:

javascript
Copy code
```
const express = require('express');
const app = express();

// Route with custom status code
app.get('/success', (req, res) => {
  res.status(200).send('Request was successful!'); // 200 OK
});

app.get('/error', (req, res) => {
  res.status(404).send('Resource not found!'); // 404 Not Found
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- Visiting /success will return a 200 OK response with the message "Request was successful!"
- Visiting /error will return a 404 Not Found response with the message "Resource not found!"

### *b) Parsing Data from Request*

To parse data from a request in Express.js, you can use the built-in middleware express.json() and express.urlencoded(). This allows you to parse JSON data and URL-encoded form data.

Here's an example program that parses both JSON and URL-encoded data:

javascript
Copy code

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

// Middleware to parse JSON and URL-encoded data
app.use(express.json()); // for parsing application/json
app.use(express.urlencoded({ extended: true })); // for parsing application/x-www-form-urlencoded

// Route to handle POST requests with JSON data
app.post('/json-data', (req, res) => {
  const data = req.body; // Parse JSON data
  res.status(200).send(`Received JSON data: ${JSON.stringify(data)}`);
});

// Route to handle POST requests with URL-encoded form data
app.post('/form-data', (req, res) => {
  const data = req.body; // Parse URL-encoded data
  res.status(200).send(`Received form data: ${JSON.stringify(data)}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- Use a tool like Postman or a form to send JSON data or URL-encoded form data to the respective endpoints (/json-data and /form-data).
- Example of a JSON payload:

  json
  Copy code

  ```
  {
    "name": "John",
    "age": 25
  }
  ```

- Example of URL-encoded data:

makefile
Copy code
name=John&age=25

**ii) MongoDB Commands**

### *a) Query to Find All Documents in the students Collection Where Age Is Between 18 and 25*

To find all documents where the age field is between 18 and 25 (inclusive), you can use the $gte (greater than or equal to) and $lte (less than or equal to) operators:

javascript
Copy code
db.students.find({ age: { $gte: 18, $lte: 25 } })

This command will return all documents in the students collection where the age is between 18 and 25.

### *b) Query to Find All Documents in the books Collection Containing the Word "MongoDB" in the Title Field*

To find all documents where the title field contains the word "MongoDB", you can use a regular expression:

javascript
Copy code
db.books.find({ title: { $regex: "MongoDB", $options: "i" } })

- The $regex operator allows you to search for a pattern within a string.
- The $options: "i" makes the search case-insensitive.

This command will return all documents in the books collection where the title contains the word "MongoDB" (case-insensitive).

**6)**

**i) Express.js Program**

### *a) Setting the Status Code*

To set the HTTP status code in Express.js, you use the res.status() method. Here's an example:

javascript
Copy code
```
const express = require('express');
const app = express();

// Route that returns a success response
app.get('/success', (req, res) => {
  res.status(200).send('Request was successful!');
});
```

```
// Route that returns an error response
app.get('/error', (req, res) => {
  res.status(404).send('Resource not found!');
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- When you visit /success, you'll get a 200 OK status with a message.
- When you visit /error, you'll get a 404 Not Found status with an error message.

### *b) Parsing Data from Request*

To parse data from a request in Express.js, you need to use express.json() for JSON data and express.urlencoded() for URL-encoded form data.

```
javascript
Copy code
const express = require('express');
const app = express();

// Middleware to parse JSON and URL-encoded data
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Route to handle POST requests with JSON data
app.post('/json-data', (req, res) => {
  const data = req.body;
  res.status(200).send(`Received JSON data: ${JSON.stringify(data)}`);
});

// Route to handle POST requests with form data
app.post('/form-data', (req, res) => {
  const data = req.body;
  res.status(200).send(`Received form data: ${JSON.stringify(data)}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- Send a JSON request to /json-data, e.g., { "name": "Alice", "age": 25 }, and the server will parse and return the data.
- Send a URL-encoded form request to /form-data, e.g., name=Alice&age=25, and it will do the same.

**ii) MongoDB Commands**

### *a) Update the Age of a Student Named "Alice" to 25 in the students Collection*

To update a specific document in the students collection, you can use the updateOne() function:

```
javascript
Copy code
db.students.updateOne(
  { name: "Alice" },       // Filter to find the document where the name is "Alice"
  { $set: { age: 25 } }    // Update the age to 25
)
```

This will find the document where the name is "Alice" and set the age to 25.

### *b) Update the Status Field to "Active" for All Documents in the employees Collection*

To update multiple documents, you can use the updateMany() function:

```
javascript
Copy code
db.employees.updateMany(
  {},                      // No filter, so it applies to all documents
  { $set: { status: "active" } }  // Set the status field to "active" for all documents
)
```

This command will update the status field to "active" for all documents in the employees collection.

**7)**

**i) Express.js Program**

### *a) Implementation of 2 Middlewares*

In Express.js, middleware functions can be used for processing requests before they reach the route handler. Here's an example program where we implement two middlewares:

1. **Middleware 1**: Logs the request method and URL.
2. **Middleware 2**: Adds a timestamp to the request.

```
javascript
Copy code
const express = require('express');
const app = express();

// Middleware 1: Logs request method and URL
const logRequest = (req, res, next) => {
  console.log(`Method: ${req.method}, URL: ${req.url}`);
  next(); // Pass control to the next middleware
};

// Middleware 2: Adds a timestamp to the request
```

```
const addTimestamp = (req, res, next) => {
  req.requestTime = new Date().toISOString();
  next(); // Pass control to the next middleware
};

// Use the middlewares for all routes
app.use(logRequest);
app.use(addTimestamp);

// Route handler
app.get('/', (req, res) => {
  res.send(`Hello! Request received at: ${req.requestTime}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server is running at http://localhost:3000/');
});
```

- This program uses two middlewares for all routes. When you access http://localhost:3000/, it will show the time when the request was received and log the method and URL to the console.

### *b) Filtering Paths Using URL Prefix*

You can filter routes based on URL prefixes in Express.js by using the app.use() method. Here's an example of routing using the /api prefix:

```
javascript
Copy code
const express = require('express');
const app = express();

// Middleware that applies to all routes with the /api prefix
app.use('/api', (req, res, next) => {
  console.log('API route accessed');
  next(); // Proceed to the next middleware or route handler
});

// Route handler for /api/users
app.get('/api/users', (req, res) => {
  res.send('User list');
});

// Route handler for /api/products
app.get('/api/products', (req, res) => {
  res.send('Product list');
});

// Non-API route
app.get('/', (req, res) => {
```

```
  res.send('Main page');
});

// Start the server
app.listen(3000, () => {
  console.log('Server is running at http://localhost:3000/');
});
```

- When you visit any URL starting with /api (e.g., /api/users, /api/products), the middleware logs the message, and the appropriate route is handled. The root / path is unaffected by the middleware.

**ii) MongoDB Commands**

***a) Update the Age of a Student Named "Alice" to 25 in the students Collection***

To update a single document where the name is "Alice", you can use the following MongoDB command:

```
javascript
Copy code
db.students.updateOne(
  { name: "Alice" },       // Filter to find the document where the name is "Alice"
  { $set: { age: 25 } }    // Update the age to 25
)
```

This command will find the document in the students collection where the name is "Alice" and update the age field to 25.

***b) Update the Status Field to "Active" for All Documents in the employees Collection***

To update the status field to "active" for all documents in the employees collection, you can use the updateMany() command:

```
javascript
Copy code
db.employees.updateMany(
  {},                      // Empty filter means apply to all documents
  { $set: { status: "active" } }  // Set the status field to "active"
)
```

This command will update the status field to "active" for all documents in the employees collection.

**8)**

**i) Node.js Program**

***a) File Upload Operation***

To handle file uploads in Node.js, you can use the formidable package, which makes it easy to manage file uploads.

First, install the formidable package by running:

bash
Copy code
npm install formidable

Here is an example program for file upload:

javascript
Copy code

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

// Create an HTTP server
http.createServer((req, res) => {
  if (req.url === '/fileupload' && req.method.toLowerCase() === 'post') {
    // Create an instance of formidable to parse the incoming form
    const form = new formidable.IncomingForm();

    form.parse(req, (err, fields, files) => {
      if (err) {
        res.writeHead(500, { 'Content-Type': 'text/plain' });
        res.write('Error uploading file');
        return res.end();
      }
      // Get the temporary file path and save it to a new location
      const oldPath = files.fileToUpload.filepath;
      const newPath = __dirname + '/' + files.fileToUpload.originalFilename;

      fs.rename(oldPath, newPath, (err) => {
        if (err) throw err;
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('File uploaded and moved!');
        res.end();
      });
    });
  } else {
    // Display a simple file upload form
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<form action="/fileupload" method="post" enctype="multipart/form-data">');
    res.write('<input type="file" name="fileToUpload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

```
console.log('Server running at http://localhost:8080/');
```

- Start the server using node server.js.
- Navigate to http://localhost:8080/ in your browser, where you can upload a file, and the server will save it to the specified path.

### *b) Event Handling*

In Node.js, the events module allows you to handle events. Here's an example that shows basic event handling:

javascript
Copy code

```
const events = require('events');
const eventEmitter = new events.EventEmitter();

// Create an event handler
const eventHandler = () => {
  console.log('Event triggered!');
};

// Assign the event handler to an event called 'myEvent'
eventEmitter.on('myEvent', eventHandler);

// Fire the 'myEvent' event
eventEmitter.emit('myEvent');
```

This program demonstrates how to bind an event and trigger it. It will output "Event triggered!" when run.

**ii) MongoDB Commands**

### *a) Find All Documents in the students Collection Where Age Is Between 18 and 25*

To find all documents where the age field is between 18 and 25 (inclusive), you can use the following query:

javascript
Copy code

```
db.students.find({ age: { $gte: 18, $lte: 25 } })
```

This query will return all documents in the students collection where the age field is between 18 and 25.

### *b) Find All Documents in the books Collection Containing the Word "MongoDB" in the title Field*

To find documents where the title field contains the word "MongoDB", you can use a regular expression search:

javascript

Copy code
```
db.books.find({ title: { $regex: "MongoDB", $options: "i" } })
```

- The $regex operator searches for the string "MongoDB" within the title field.
- The $options: "i" makes the search case-insensitive, so it matches both "MongoDB" and "mongodb".

**9)**

**i) Node.js Program**

***a) Split Up a Web Address Into Readable Parts***

To split a web address into readable parts in Node.js, you can use the built-in url module.

Here's an example:

javascript
Copy code
```
const url = require('url');

// Example web address
const address = 'http://localhost:8080/default.htm?year=2023&month=October';

// Parse the web address into parts
const parsedUrl = url.parse(address, true);

console.log('Host: ' + parsedUrl.host);          // localhost:8080
console.log('Pathname: ' + parsedUrl.pathname);  // /default.htm
console.log('Search Params: ' + parsedUrl.search); // ?year=2023&month=October

// Query parameters as an object
const queryData = parsedUrl.query;
console.log('Year: ' + queryData.year);    // 2023
console.log('Month: ' + queryData.month);  // October
```

- The url.parse() method breaks down the URL into its parts (host, pathname, query parameters).
- This will output the host, pathname, and the query parameters like "year" and "month."

***b) Display Customized Error Message***

To display customized error messages in Node.js, you can use a try-catch block or custom error handling.

Here's an example with custom error handling:

javascript
Copy code
```
const express = require('express');
const app = express();
```

```
// Route handler with an error
app.get('/', (req, res) => {
  throw new Error('Something went wrong!');
});

// Custom error handling middleware
app.use((err, req, res, next) => {
  console.error(err.message); // Log the error message
  res.status(500).send('Custom Error: ' + err.message); // Send a custom error message
});

// Start the server
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000/');
});
```

- In this example, an error is intentionally thrown, and the custom error handler sends a personalized error message to the client.

**ii) MongoDB Commands**

***a) Find the First 5 Students Sorted by Age in Descending Order***

To find the first 5 students sorted by age in descending order, you can use the following MongoDB command:

```
javascript
Copy code
db.students.find().sort({ age: -1 }).limit(5)
```

- .sort({ age: -1 }) sorts the documents by the age field in descending order.
- .limit(5) limits the result to the first 5 documents.

***b) Find All Documents in the books Collection Containing the Word "MongoDB" in the Title Field***

To find all documents where the title field contains the word "MongoDB", you can use a regular expression search:

```
javascript
Copy code
db.books.find({ title: { $regex: "MongoDB", $options: "i" } })
```

- The $regex operator searches for the string "MongoDB" within the title field.
- The $options: "i" ensures the search is case-insensitive.

**10)**

**i) Node.js Program**

### *a) Display a Message in the Browser*

You can create a simple HTTP server in Node.js that sends a message to the browser.

javascript
Copy code

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

// Create an HTTP server
http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.write('Hello, welcome to the Node.js server!');
  res.end();
}).listen(8080);

console.log('Server running at http://localhost:8080/');
```

When you access http://localhost:8080/, the browser will display "Hello, welcome to the Node.js server!".

### *b) Display the Current Date and Time in the Browser*

To display the current date and time, you can modify the previous example:

javascript
Copy code

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

// Create an HTTP server
http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  const date = new Date();
  res.write(`Current Date and Time: ${date}`);
  res.end();
}).listen(8080);

console.log('Server running at http://localhost:8080/');
```

This will display the current date and time in the browser when you visit http://localhost:8080/.

### *c) Pass a Query String in the URL*

To handle query strings, you can use Node.js's url module. Here's how you can extract and display query parameters:

javascript
Copy code

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

// Create an HTTP server
```

```
http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });

  // Parse the query string
  const queryObject = url.parse(req.url, true).query;

  // Display the query parameters
  res.write('Query Parameters:<br>');
  res.write(`Name: ${queryObject.name}<br>`);
  res.write(`Age: ${queryObject.age}<br>`);
  res.end();
}).listen(8080);

console.log('Server running at http://localhost:8080/');

// Example: http://localhost:8080/?name=John&age=25
```

- When you visit http://localhost:8080/?name=John&age=25, it will display the query string parameters (name and age) in the browser.

**ii) MongoDB Commands**

***a) Update the Email of a Specific User in the customers Collection Where the customerID Is 12345***

To update the email of a specific user where the customerID is 12345, you can use the following MongoDB command:

```
javascript
Copy code
db.customers.updateOne(
  { customerID: 12345 },       // Filter to find the document with customerID 12345
  { $set: { email: "newemail@example.com" } }  // Update the email
)
```

This will find the customer with customerID: 12345 and update their email.

***b) Add a status Field with the Value "Inactive" to All Documents in the users Collection***

To add a new status field with the value "inactive" to all documents in the users collection, you can use the updateMany() function:

```
javascript
Copy code
db.users.updateMany(
  {},                          // No filter, so it applies to all documents
  { $set: { status: "inactive" } }  // Add the status field with value "inactive"
)
```

This command will add a status field with the value "inactive" to every document in the users collection.

**11)**

**i) Express.js Program**

### *a) Setting the Status Code*

In Express.js, you can set the HTTP status code using the res.status() method before sending the response.

Here's a simple example:

javascript
Copy code

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

// Middleware to parse JSON data
app.use(express.json());

// Route to demonstrate setting the status code
app.post('/status', (req, res) => {
  const { success } = req.body;

  if (success) {
    // Set status code to 200 for success
    res.status(200).send('Success: Operation completed successfully.');
  } else {
    // Set status code to 400 for failure
    res.status(400).send('Error: Operation failed.');
  }
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- You can send a POST request to http://localhost:3000/status with a JSON body like {"success": true} or {"success": false} to see the respective status codes.

### *b) Parsing Data from Request*

To parse data from the request, you can use middleware like express.json() for JSON data or express.urlencoded() for URL-encoded data. Here's an example that handles both:

javascript
Copy code

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

// Middleware to parse JSON data
```

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Route to demonstrate parsing data from a request
app.post('/submit', (req, res) => {
  const { name, age } = req.body; // Destructure data from the request body
  res.send(`Received Name: ${name}, Age: ${age}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- You can send a POST request to http://localhost:3000/submit with a JSON body or form data containing name and age.

**ii) MongoDB Commands**

### *a) Insert a New Employee Record into the employees Collection*

To insert a new employee record with the fields name, age, and department, you can use the following MongoDB command:

```
javascript
Copy code
db.employees.insertOne({
  name: "John Doe",
  age: 28,
  department: "HR"
})
```

- This command will add a new document to the employees collection with the specified fields.

### *b) Insert Multiple Documents into the products Collection*

To insert multiple documents into the products collection, each having fields productName, price, and category, you can use the following MongoDB command:

```
javascript
Copy code
db.products.insertMany([
  { productName: "Laptop", price: 999.99, category: "Electronics" },
  { productName: "Coffee Maker", price: 49.99, category: "Home Appliances" },
  { productName: "Headphones", price: 89.99, category: "Electronics" },
  { productName: "Office Chair", price: 199.99, category: "Furniture" },
  { productName: "Smartphone", price: 799.99, category: "Electronics" }
])
```

- This command will insert five new documents into the products collection with the specified fields and values.

**12)**

**i) Express.js Program**

### *a) Implementation of 2 Middlewares*

In Express.js, middleware functions can perform operations on the request object, response object, or even end the request-response cycle. Below is an example that demonstrates two middleware functions.

javascript
Copy code

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

// Middleware 1: Logging the request method and URL
app.use((req, res, next) => {
  console.log(`Request Method: ${req.method}, Request URL: ${req.url}`);
  next(); // Call the next middleware
});

// Middleware 2: Check if a custom header is present
app.use((req, res, next) => {
  if (req.headers['x-custom-header']) {
    console.log('Custom Header Found:', req.headers['x-custom-header']);
  } else {
    console.log('No Custom Header Found');
  }
  next(); // Call the next middleware
});

// Route to respond to GET requests
app.get('/', (req, res) => {
  res.send('Welcome to the Express.js server!');
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- The first middleware logs the request method and URL.
- The second middleware checks for the presence of a custom header in the request.

### *b) Filtering Paths Using URL Prefix*

To filter paths using URL prefixes, you can use the app.use() method to define a base path for a group of routes.

javascript
Copy code
```
const express = require('express');
const app = express();

// Middleware to parse JSON data
app.use(express.json());

// Base path for user-related routes
app.use('/users', (req, res, next) => {
  console.log('User-related request received');
  next(); // Call the next middleware or route handler
});

// Route to get all users
app.get('/users', (req, res) => {
  res.send('List of all users');
});

// Route to get a specific user by ID
app.get('/users/:id', (req, res) => {
  res.send(`User details for user ID: ${req.params.id}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- All routes prefixed with /users will use the middleware defined for user-related requests.

**ii) MongoDB Commands**

***a) Find All Documents in the users Collection Where the Age Is Greater Than 25***

To find all documents in the users collection where the age is greater than 25, you can use the following command:

javascript
Copy code
```
db.users.find({ age: { $gt: 25 } })
```

- The $gt operator specifies "greater than."

***b) Find a Document in the customers Collection Where the customerID Is 1001 and Only Return the Name and Email Fields***

To find a specific document in the customers collection where the customerID is 1001 and only return the name and email fields, you can use the following command:

javascript

Copy code
```
db.customers.find(
  { customerID: 1001 }, // Filter condition
  { name: 1, email: 1, _id: 0 } // Projection to include only name and email, exclude _id
)
```

- In the projection, 1 means to include the field, and 0 means to exclude it. In this case, we're excluding the _id field.

**13)**

**i) Node.js Program**

***a) File Upload Operation***

To handle file uploads in Node.js, you can use the multer middleware. Below is an example of how to set up a basic file upload operation.

1. First, install the required packages:

bash
Copy code
```
npm install express multer
```

2. Create a Node.js server with a file upload route:

javascript
Copy code
```
const express = require('express');
const multer = require('multer');
const path = require('path');

const app = express();
const upload = multer({ dest: 'uploads/' }); // Specify the upload directory

// Route to upload a file
app.post('/upload', upload.single('file'), (req, res) => {
  res.send(`File uploaded successfully: ${req.file.filename}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- In this example, when you send a POST request to /upload with a file under the form field file, the file will be saved in the uploads directory.

***b) Event Handling***

Node.js has an EventEmitter class that allows you to handle events. Below is an example demonstrating basic event handling:

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

// Create an instance of EventEmitter
const myEmitter = new EventEmitter();

// Define an event listener
myEmitter.on('greet', (name) => {
  console.log(`Hello, ${name}!`);
});

// Emit the event
myEmitter.emit('greet', 'Alice');

// Start the server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- This code creates an event called greet that listens for a name and logs a greeting message. You can emit this event with a name to see the greeting.

**ii) MongoDB Commands**

***a) Retrieve All Products in the inventory Collection Where Quantity Is Between 10 and 100, Inclusive***

To retrieve all products in the inventory collection where the quantity is between 10 and 100 (inclusive), you can use the following MongoDB query:

javascript
Copy code
```
db.inventory.find({ quantity: { $gte: 10, $lte: 100 } })
```

- The $gte operator means "greater than or equal to," and $lte means "less than or equal to."

***b) Update the Price of a Product Where the ProductName Is "Laptop" to 1200 in the products Collection***

To update the price of a product where the productName is "Laptop" to 1200, you can use the following command:

javascript
Copy code
```
db.products.updateOne(
  { productName: "Laptop" }, // Filter condition
```

```
  { $set: { price: 1200 } }   // Update operation
)
```

- The updateOne() method will find the first document that matches the filter condition and update its price to 1200.

**14)**

**i) Node.js Program**

***a) Splits Up a Web Address into Readable Parts***

You can use Node.js's built-in url module to parse a web address. Below is a simple program that splits a web address into its components:

javascript
Copy code

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

const server = http.createServer((req, res) => {
  // Parse the URL
  const parsedUrl = url.parse(req.url, true);

  // Display the readable parts of the URL
  res.writeHead(200, { 'Content-Type': 'application/json' });
  res.write(JSON.stringify(parsedUrl, null, 2)); // Format JSON for better readability
  res.end();
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- When you access http://localhost:3000/?name=John&age=25, it will display the URL's parsed components, including protocol, host, pathname, and query parameters.

***b) Display Customized Error Message***

To display a customized error message, you can define a route that handles errors. Here's an example:

javascript
Copy code

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

const server = http.createServer((req, res) => {
  if (req.url === '/error') {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end('Customized Error Message: The requested resource was not found.');
```

```
  } else {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Welcome to the Node.js server!');
  }
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- Accessing http://localhost:3000/error will display a customized error message, while accessing any other route will show a welcome message.

**ii) MongoDB Commands**

***a) Update the Status of All Orders in the orders Collection Where the Status Is "Pending" to "Shipped"***

To update the status of all orders in the orders collection where the status is "pending" to "shipped", you can use the following MongoDB command:

```
javascript
Copy code
db.orders.updateMany(
  { status: "pending" },       // Filter condition
  { $set: { status: "shipped" }} // Update operation
)
```

- The updateMany() method will update all matching documents in the collection.

***b) Increment the quantitySold Field by 5 for All Documents in the sales Collection Where productID Is 2001***

To increment the quantitySold field by 5 for all documents in the sales collection where productID is 2001, you can use the following command:

```
javascript
Copy code
db.sales.updateMany(
  { productID: 2001 }, // Filter condition
  { $inc: { quantitySold: 5 } } // Increment operation
)
```

- The $inc operator is used to increment the value of the specified field by the specified amount. In this case, it adds 5 to quantitySold for all matching documents.

**15)**

**i) Node.js Program**

### *a) Display a Message in the Browser*

To create a simple Node.js server that displays a message in the browser, you can use the built-in http module.

javascript
Copy code

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end('<h1>Hello, Welcome to the Node.js Server!</h1>');
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- When you visit http://localhost:3000, you will see the welcome message.

### *b) Display the Current Date and Time in the Browser*

You can modify the server to display the current date and time along with the welcome message.

javascript
Copy code

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });

  const currentDateTime = new Date().toLocaleString(); // Get current date and time
  res.end(`<h1>Hello, Welcome to the Node.js Server!</h1><p>Current Date and Time:
${currentDateTime}</p>`);
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- This will display the current date and time each time you refresh the browser.

### *c) Pass a Query String in URL*

You can enhance the server to display query string parameters passed in the URL.

javascript
Copy code

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

```
const url = require('url');

const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true); // Parse the URL
  const query = parsedUrl.query; // Get the query string

  res.writeHead(200, { 'Content-Type': 'text/html' });

  const currentDateTime = new Date().toLocaleString(); // Get current date and time
  res.end(`
    <h1>Hello, Welcome to the Node.js Server!</h1>
    <p>Current Date and Time: ${currentDateTime}</p>
    <p>Query Parameters: ${JSON.stringify(query)}</p>
  `);
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- If you visit http://localhost:3000/?name=John&age=25, it will display the query parameters in JSON format.

**ii) MongoDB Commands**

***a) Delete a Single Document from the employees Collection Where the employeeID Is 101***

To delete a single document from the employees collection where the employeeID is 101, you can use the following MongoDB command:

javascript
Copy code

```
db.employees.deleteOne({ employeeID: 101 })
```

- This command will remove the first document that matches the filter condition.

***b) Delete All Documents from the products Collection Where the category Is "Obsolete"***

To delete all documents from the products collection where the category is "obsolete", you can use the following command:

javascript
Copy code

```
db.products.deleteMany({ category: "obsolete" })
```

- This command will remove all documents that match the specified filter condition in the products collection.

**16)**

**i) Node.js Program**

### *a) Display a Message in the Browser*

To create a simple Node.js server that displays a message in the browser, you can use the built-in http module.

javascript
Copy code

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end('<h1>Hello, Welcome to the Node.js Server!</h1>');
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- When you visit http://localhost:3000, you will see the welcome message.

### *b) Display the Current Date and Time in the Browser*

You can modify the server to display the current date and time along with the welcome message.

javascript
Copy code

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });

  const currentDateTime = new Date().toLocaleString(); // Get current date and time
  res.end(`<h1>Hello, Welcome to the Node.js Server!</h1><p>Current Date and Time:
${currentDateTime}</p>`);
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- This will display the current date and time each time you refresh the browser.

### *c) Pass a Query String in URL*

You can enhance the server to display query string parameters passed in the URL.

javascript

Copy code
```
const http = require('http');
const url = require('url');

const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true); // Parse the URL
  const query = parsedUrl.query; // Get the query string

  res.writeHead(200, { 'Content-Type': 'text/html' });

  const currentDateTime = new Date().toLocaleString(); // Get current date and time
  res.end(`
    <h1>Hello, Welcome to the Node.js Server!</h1>
    <p>Current Date and Time: ${currentDateTime}</p>
    <p>Query Parameters: ${JSON.stringify(query)}</p>
  `);
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- If you visit http://localhost:3000/?name=John&age=25, it will display the query parameters in JSON format.

**ii) MongoDB Commands**

***a) Delete All Documents from the logs Collection***

To delete all documents from the logs collection, you can use the following MongoDB command:

javascript
Copy code
```
db.logs.deleteMany({})
```

- The deleteMany({}) command will remove all documents in the logs collection.

***b) Remove a Document from the orders Collection Where the orderID Is 555, If It Exists***

To remove a document from the orders collection where the orderID is 555, you can use the following command:

javascript
Copy code
```
db.orders.deleteOne({ orderID: 555 })
```

- The deleteOne() command will delete the first document that matches the filter condition (if it exists). If no document matches, no changes will be made.

**17)**

**i) Node.js Program**

### *a) Splits Up a Web Address into Readable Parts*

You can use Node.js's built-in url module to parse a web address and display its components. Here's a simple program that does this:

javascript
Copy code

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

const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true); // Parse the URL

  // Display the readable parts of the URL
  res.writeHead(200, { 'Content-Type': 'application/json' });
  res.end(JSON.stringify(parsedUrl, null, 2)); // Format JSON for better readability
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- When you access http://localhost:3000/?name=John&age=25, it will display the parsed components of the URL in a readable JSON format.

### *b) Display Customized Error Message*

To display a customized error message for a specific route, you can add a conditional statement that checks the requested URL.

javascript
Copy code

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

const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true); // Parse the URL

  // Check if the request URL is "/error" to display a customized error message
  if (parsedUrl.pathname === '/error') {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end('Customized Error Message: The requested resource was not found.');
  } else {
    // Display the readable parts of the URL for other routes
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify(parsedUrl, null, 2)); // Format JSON for better readability
```

```
  }
});

// Start the server
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- Accessing http://localhost:3000/error will show a customized error message, while any other URL will display its parsed components.

**ii) MongoDB Commands**

***a) Query to Retrieve the First 5 Documents in the students Collection, Sorted by Age in Descending Order***

To retrieve the first 5 documents in the students collection, sorted by age in descending order, you can use the following MongoDB command:

javascript
Copy code
```
db.students.find().sort({ age: -1 }).limit(5)
```

- In this command:
  - find() retrieves documents from the collection.
  - sort({ age: -1 }) sorts the results in descending order based on the age field (use 1 for ascending).
  - limit(5) restricts the output to the first 5 documents.

***b) Update the Status Field to "Closed" and Add a closingDate Field for All Documents in the projects Collection Where dueDate Is Less Than the Current Date***

To update the status field to "closed" and add a closingDate field for all documents in the projects collection where the dueDate is less than the current date, you can use the following MongoDB command:

javascript
Copy code
```
db.projects.updateMany(
  { dueDate: { $lt: new Date() } }, // Filter for documents with dueDate less than current date
  {
    $set: {
      status: "closed",
      closingDate: new Date() // Set the closingDate to the current date
    }
  }
)
```

- In this command:
  - updateMany() is used to update all matching documents.

- The filter { dueDate: { $lt: new Date() } } finds documents with a dueDate less than the current date.
- The $set operator updates the status to "closed" and adds a closingDate with the current date.

**18)**

**i) Node.js Program**

To implement a file upload operation and event handling in Node.js, you can use the express framework along with the multer middleware for handling file uploads.

### *a) File Upload Operation*

First, make sure to install the required packages by running:

bash
Copy code
npm install express multer

Here is a simple Node.js program for handling file uploads:

javascript
Copy code

```
const express = require('express');
const multer = require('multer');
const path = require('path');

const app = express();
const port = 3000;

// Set up storage using multer
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/'); // Directory to save uploaded files
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname)); // Rename the file to avoid conflicts
  }
});

const upload = multer({ storage: storage });

// Serve the HTML form for file upload
app.get('/', (req, res) => {
  res.send(`
    <h2>Upload a File</h2>
    <form action="/upload" method="post" enctype="multipart/form-data">
      <input type="file" name="file" required />
      <button type="submit">Upload</button>
    </form>
```

```
  `);
});

// Handle file upload
app.post('/upload', upload.single('file'), (req, res) => {
  res.send('File uploaded successfully: ' + req.file.filename);
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

- In this code, an HTML form is served to allow users to upload a file.
- The uploaded file is saved in the uploads/ directory with a unique name.

### *b) Event Handling*

You can also include event handling to handle file upload success or errors. Below is an extension of the above code that includes basic event handling using the EventEmitter class.

javascript
Copy code

```
const EventEmitter = require('events');
const eventEmitter = new EventEmitter();

// Event listener for successful uploads
eventEmitter.on('fileUploaded', (filename) => {
  console.log(`File uploaded: ${filename}`);
});

// Handle file upload with event emission
app.post('/upload', upload.single('file'), (req, res) => {
  eventEmitter.emit('fileUploaded', req.file.filename); // Emit the event
  res.send('File uploaded successfully: ' + req.file.filename);
});
```

- The event emitter listens for the fileUploaded event and logs the filename to the console whenever a file is successfully uploaded.

**ii) MongoDB Commands**

### *a) Update a Product's Quantity by 10, and Then Delete the Product if Its Quantity Becomes Greater Than 1000*

To accomplish this, you can use the following MongoDB commands:

1. **Update the Quantity**: First, update the product's quantity by 10.

javascript
Copy code

```
db.products.updateOne(
  { productID: <your_product_id> }, // Filter by productID or any unique identifier
  { $inc: { quantity: 10 } } // Increment quantity by 10
)
```

2. **Check the Quantity and Delete if Greater Than 1000**: You can use the following command in a JavaScript function to handle the condition:

```
javascript
Copy code
const product = db.products.findOne({ productID: <your_product_id> }); // Retrieve the product

if (product.quantity + 10 > 1000) {
  db.products.deleteOne({ productID: <your_product_id> }); // Delete the product if the
condition is met
}
```

3. **Combined Command**: If you want to handle this in one function (like in an application context), you could do it like this:

```
javascript
Copy code
const productID = <your_product_id>; // Replace with the actual product ID

// Update the product's quantity by 10
db.products.updateOne(
  { productID: productID },
  { $inc: { quantity: 10 } },
  (err, result) => {
    if (err) throw err;

    // Check the new quantity
    db.products.findOne({ productID: productID }, (err, product) => {
      if (err) throw err;

      // If quantity exceeds 1000, delete the product
      if (product.quantity > 1000) {
        db.products.deleteOne({ productID: productID }, (err) => {
          if (err) throw err;
          console.log(`Product with ID ${productID} deleted as quantity exceeded 1000.`);
        });
      }
    });
  }
);
```

- This code snippet updates the product's quantity by 10 and checks if it exceeds 1000, subsequently deleting it if necessary. Make sure to replace <your_product_id> with the actual ID of the product you want to update.

**19)**

**i) Express.js Program**

To implement an Express.js application with two middlewares and filtering paths using URL prefixes, follow the example below:

### *a) Implementation of 2 Middlewares*

1. **Logging Middleware**: Logs each request to the console.
2. **Authorization Middleware**: Checks for a custom header to simulate an authorization check.

javascript
Copy code

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

const app = express();
const port = 3000;

// Middleware 1: Logging Middleware
app.use((req, res, next) => {
  console.log(`${req.method} request for '${req.url}'`);
  next(); // Call the next middleware or route handler
});

// Middleware 2: Authorization Middleware
app.use((req, res, next) => {
  if (req.headers['authorization'] !== 'Bearer my-secret-token') {
    return res.status(403).send('Forbidden: Invalid authorization token');
  }
  next(); // Call the next middleware or route handler
});

// Routes
app.get('/api/data', (req, res) => {
  res.send('Protected data accessed!');
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

### *b) Filtering Paths Using URL Prefix*

You can use a URL prefix to filter requests to specific routes. For example, if you want to create routes that respond to URLs starting with /api, you can do it like this:

javascript
Copy code

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

```
const app = express();
const port = 3000;

// Middleware for logging
app.use((req, res, next) => {
  console.log(`${req.method} request for '${req.url}'`);
  next();
});

// Middleware for authorization
app.use((req, res, next) => {
  if (req.headers['authorization'] !== 'Bearer my-secret-token') {
    return res.status(403).send('Forbidden: Invalid authorization token');
  }
  next();
});

// Filtering paths using URL prefix
const apiRouter = express.Router();

apiRouter.get('/data', (req, res) => {
  res.send('Protected data accessed!');
});

apiRouter.get('/info', (req, res) => {
  res.send('Some protected info!');
});

// Use the apiRouter for any route that starts with /api
app.use('/api', apiRouter);

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

- In this code, the apiRouter handles any requests that start with /api, making it easy to organize routes under a common prefix.

**ii) MongoDB Commands**

***a) Insert a Document into the students Collection***

To insert a document into the students collection with the fields name, age, grade, and subjects (where subjects is an array containing "Math" and "Science"), you can use the following MongoDB command:

javascript
Copy code

```
db.students.insertOne({
  name: "John Doe",
```

```
  age: 20,
  grade: "A",
  subjects: ["Math", "Science"]
});
```

- This command creates a new document in the students collection with the specified fields.

### *b) Insert Multiple Documents into the orders Collection*

To insert multiple documents into the orders collection with the fields orderID, product, quantity, and status, you can use the following MongoDB command:

```
javascript
Copy code
db.orders.insertMany([
  {
    orderID: 101,
    product: "Laptop",
    quantity: 2,
    status: "Shipped"
  },
  {
    orderID: 102,
    product: "Smartphone",
    quantity: 5,
    status: "Pending"
  },
  {
    orderID: 103,
    product: "Tablet",
    quantity: 1,
    status: "Delivered"
  }
]);
```

- This command inserts multiple documents into the orders collection in one operation, allowing you to specify different values for each order.

**20)**

**i) Express.js Program**

To create an Express.js program that sets a status code and parses data from a request, you can follow the example below:

### *a) Setting the Status Code*

This example demonstrates how to set different HTTP status codes based on the outcome of a request.

### *b) Parsing Data from Request*

You can use express.json() middleware to parse JSON data from incoming requests.

Here's a complete Express.js program that illustrates these concepts:

javascript
Copy code

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware to parse JSON bodies
app.use(express.json());

// Route to handle POST request
app.post('/api/employees', (req, res) => {
  const { name, age, department, salary } = req.body;

  // Check if the required fields are present
  if (!name || !age || !department || !salary) {
    return res.status(400).json({ error: 'All fields are required' }); // Bad Request
  }

  // Simulating adding an employee (you would usually save to a database)
  const newEmployee = { name, age, department, salary };
  console.log('New Employee Added:', newEmployee);

  // Respond with a 201 Created status
  res.status(201).json({ message: 'Employee added successfully', employee: newEmployee });
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

- In this program:
  - The JSON body of incoming requests is parsed using express.json().
  - The endpoint /api/employees accepts POST requests, checks for the presence of required fields, and responds with an appropriate status code.

**ii) MongoDB Commands**

### *a) Find All Documents in the employees Collection Where Department is "IT" and Age is Less Than 40*

To find documents in the employees collection with the specified criteria, use the following MongoDB query:

javascript

Copy code
```
db.employees.find({
  department: "IT",
  age: { $lt: 40 }
});
```

- This query retrieves all employees in the "IT" department whose age is less than 40.

### *b) Write a Query to Return Only the Name and Salary of Employees in the employees Collection Where the Salary is Greater Than 5000*

To return specific fields from documents in the employees collection, you can use the following MongoDB query:

javascript
Copy code
```
db.employees.find(
  { salary: { $gt: 5000 } }, // Condition to filter employees with salary greater than 5000
  { name: 1, salary: 1, _id: 0 } // Projection to return only name and salary, exclude _id
);
```

- In this command:
    - The first argument of find() specifies the filtering criteria (employees with salary greater than 5000).
    - The second argument specifies the fields to include in the results (name and salary) and excludes the default _id field by setting it to 0.