

# Class Practice Problems

Name: Gaurav Deshmukh

USN: 72232433C

Q. 1. Display File Statistics Using fs and path Modules – 5 Marks Objective: Use Node.js fs and path modules to read a file and display its statistics.

The screenshot shows a code editor interface with a dark theme. At the top, there's a tab bar with 'JS getFileStats.js X' and 'test\_file.txt'. Below the tabs is a status bar with 'JS getFileStats.js > ...'. The main area contains the following code:

```
JS getFileStats.js > ...
11
12 // Import modules
13 const fs = require("fs");
14 const path = require("path");
15
16 const filePath = path.join(__dirname, "sample.txt"); // file path
17 try {
18     const stats = fs.statSync(filePath);
19
20     console.log("File Statistics");
21     console.log("File Size:", stats.size, "bytes"); // file size
22     console.log("Creation Date:", stats.birthtime); // Creation Date
23     console.log("Last Modified:", stats.mtime); // Last Modified
24     console.log("Directory Name:", path.dirname(filePath)); // Directory Name
25
26 } catch (err) {
27     console.error("Error reading file:", err.message);
28 }
29
30
```

Below the code editor is a terminal window. The 'OUTPUT' tab is selected. The 'TERMINAL' section shows the execution of the script and its output:

```
File Statistics for: test_file.txt
Directory Name: .
File Size (bytes): 0
Creation Date: 22/11/2025, 6:16:42 pm
>Last Modified Date: 22/11/2025, 6:16:42 pm
PS C:\Users\gaura\OneDrive\Desktop\FINACLE TRAINING\Node.js and V8JS\Day 20 22-11-2025>
```

Q. 2. Function to run the transformation pipeline : Uppercase -> Reverse -> Append Suffix

```
JS getFileStats.js > ...
52
53     function toUpperCase(str) {
54         return str.toUpperCase(); // Uppercase conversion
55     }
56
57     function reverseString(str) {
58         return str.split('').reverse().join(''); // Split, reverse, join
59     }
60
61     function appendSuffix(str) {
62         return str + "_DONE"; // Append suffix
63     }
64
65     // Pipeline
66
67     function runPipeline(inputStr) {
68         // Apply transformations sequentially
69         let result = toUpperCase(inputStr);
70         result = reverseString(result);
71         result = appendSuffix(result);
72         return result;
73     }
74

55     // Input Handling
56
57     const inputString = process.argv[2]; // Get argument
58
59     if (!inputString) {
60         console.error("Usage: node stringPipeline.js \"your string here\"");
61         process.exit(1);
62     }
63
64     try {
65         const finalResult = runPipeline(inputString);
66         console.log("Original:", inputString);
67         console.log("Transformed:", finalResult);
68     } catch (error) {
69         console.error("Error:", error.message); // Handle errors
70     }
71

OUTPUT DEBUG CONSOLE PORTS
▼ TERMINAL
PS C:\Users\gaura\OneDrive\Desktop\FINACLE TRAINING\Node.js and V8JS\Day 20 22-11-2025> node
Original: GAURAV
↳ Transformed: VARUAG_DONE
PS C:\Users\gaura\OneDrive\Desktop\FINACLE TRAINING\Node.js and V8JS\Day 20 22-11-2025> []
```

### Q. 3. Student Management System (CRUD) .

studentModel.js

```
JS studentModel.js X JS studentRoutes.js JS server.js
JS studentModel.js > ...
1 const mongoose = require('mongoose');
2
3 const studentSchema = new mongoose.Schema({
4   name: { type: String, required: true },
5   age: { type: Number, required: true, min: 16 },
6   grade: { type: String, required: true }
7 });
8
9 module.exports = mongoose.model('Student', studentSchema);
```

studentRoutes.js

```
JS studentModel.js JS studentRoutes.js X JS server.js
JS studentRoutes.js > ...
1 const express = require('express');
2 const router = express.Router();
3 const Student = require('../studentModel');
4
5 // GET all students
6 router.get('/', async (req, res) => {
7   try {
8     const students = await Student.find();
9     res.json(students);
10  } catch (err) {
11    res.status(500).json({ message: err.message });
12  }
13});
14
15 // GET by ID
16 router.get('/:id', async (req, res) => {
17  try {
18    const student = await Student.findById(req.params.id);
19    res.json(student);
20  } catch (err) {
21    res.status(404).json({ message: 'Not found' });
22  }
23});
```

```
24
25 // POST
26 router.post('/', async (req, res) => {
27   const student = new Student(req.body);
28   try {
29     const newStudent = await student.save();
30     res.status(201).json(newStudent);
31   } catch (err) {
32     res.status(400).json({ message: err.message });
33   }
34 });
35
36 // PUT
37 router.put('/:id', async (req, res) => {
38   try {
39     const updatedStudent = await Student.findByIdAndUpdate(
40       req.params.id,
41       req.body,
42       { new: true, runValidators: true }
43     );
44     res.json(updatedStudent);
45   } catch (err) {
46     res.status(400).json({ message: err.message });
47   }
48 });
```

```
● 50 // DELETE
51 router.delete('/:id', async (req, res) => {
52   try {
53     await Student.findByIdAndDelete(req.params.id);
54     res.json({ message: 'Deleted student' });
55   } catch (err) {
56     res.status(500).json({ message: err.message });
57   }
58 });
59
60 module.exports = router;
61
```

## Server.js

```
serverjs > m
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const studentRoutes = require('./studentRoutes');
4
5  const app = express();
6  const PORT = 3000;
7  const DB_URI = 'mongodb://localhost:27017/studentDB';
8
9  app.use(express.json());
10
11 // Connect to MongoDB
12 mongoose.connect(DB_URI)
13   .then(() => console.log('DB Connected'))
14   .catch(err => console.error('DB Error:', err));
15
16 app.use('/api/students', studentRoutes);
17
18 // Start the server
19 app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
20
```

POSTMAN:

(1) GET all students

The screenshot shows the Postman interface for a GET request to `http://localhost:3000/api/students`. The 'Params' tab is selected. The response body is a JSON array containing two student objects:

```
1 [  
2 {  
3   "_id": "6921bcc730ffaf214c63b112",  
4   "name": "Gaurav Sharma",  
5   "age": 23,  
6   "grade": "A",  
7   "enrollmentYear": 2021  
8 },  
9 {  
10  "_id": "6921bcd030ffaf214c63b113",  
11  "name": "Ram Kumar",  
12  "age": 19,  
13  "grade": "B+",  
14  "enrollmentYear": 2024  
15 }  
16 ]
```

(2) GET student by ID

The screenshot shows the Postman interface for a GET request to `http://localhost:3000/api/students/6921bcc730ffaf214c63b112`. The 'Params' tab is selected. The response body is a single student object:

```
1 {  
2   "_id": "6921bcc730ffaf214c63b112",  
3   "name": "Gaurav Sharma",  
4   "age": 23,  
5   "grade": "A",  
6   "enrollmentYear": 2021  
7 }
```

### (3) POST new student

The screenshot shows a Postman request for a POST operation to `http://localhost:3000/api/students`. The request body is a JSON object with fields `name`, `age`, and `grade`. The response status is `201 Created` with a response time of 79 ms and a size of 319 B. The response body contains the created student's details, including an `_id` and a `__v` field.

```
1 {  
2   "name": "Vedant",  
3   "age": 22,  
4   "grade": "A"  
5 }
```

```
1 {  
2   "name": "Vedant",  
3   "age": 22,  
4   "grade": "A",  
5   "_id": "6921c5f24e95896dfd32b858",  
6   "__v": 0  
7 }
```

### (4) PUT – update student

The screenshot shows a Postman request for a PUT operation to `http://localhost:3000/api/students/6921c5f24e95896dfd32b858`. The request body is a JSON object with fields `age` and `grade`. The response status is `200 OK` with a response time of 12 ms and a size of 314 B. The response body shows the updated student details, including the original `_id` and the updated `__v` value.

```
1 {  
2   "age": 22,  
3   "grade": "B"  
4 }
```

```
1 {  
2   "_id": "6921c5f24e95896dfd32b858",  
3   "name": "Vedant",  
4   "age": 22,  
5   "grade": "B",  
6   "__v": 0  
7 }
```

## (5) DELETE student by ID

The screenshot shows a REST API testing interface. At the top, a 'DELETE' button is highlighted, and the URL is set to `http://localhost:3000/api/students/6921c5f24e95896dfd32b858`. Below the URL, there are tabs for 'Docs', 'Params', 'Authorization', 'Headers (8)', 'Body', 'Scripts', 'Tests', and 'Settings'. The 'Params' tab is selected. Under 'Query Params', there is a table with columns 'Key', 'Value', and 'Description'. The 'Body' tab is selected. In the JSON preview area, the response body is shown as:

```
1 {  
2   "message": "Deleted student"  
3 }
```

The status bar at the bottom right indicates a '200 OK' response with a duration of '14 ms' and a size of '264 B'.

## Q . 4. Book Management System (CRUD).

bookModel.js

The screenshot shows a code editor with three tabs: 'bookModel.js X', 'bookRoutes.js', and 'server.js'. The 'bookModel.js' tab is active and contains the following code:

```
BookManagement > JS bookModel.js >  <unknown>  
1 const mongoose = require('mongoose');  
2  
3 const bookSchema = new mongoose.Schema({  
4   title: {  
5     type: String,  
6     required: true,  
7     trim: true  
8   },  
9   author: {  
10    type: String,  
11    required: true  
12  },  
13   price: {  
14    type: Number,  
15    required: true,  
16    min: 0  
17  },  
18   publishedDate: {  
19    type: Date,  
20    required: true  
21  }  
22 }, { timestamps: true });  
23  
24 module.exports = mongoose.model('Book', bookSchema);
```

## bookRoutes.js

```
BookManagement > JS bookRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const Book = require('./bookModel');
4
5  // 1. READ ALL (GET /api/books)
6  router.get('/', async (req, res) => {
7    try {
8      const books = await Book.find();
9      res.status(200).json(books);
10     } catch (err) {
11       res.status(500).json({ message: 'Error retrieving books: ' + err.message });
12     }
13   });
14
15 // 2. READ BY ID (GET /api/books/:id)
16 router.get('/:id', async (req, res) => {
17   try {
18     const book = await Book.findById(req.params.id);
19     if (!book) return res.status(404).json({ message: 'Book not found' });
20     res.status(200).json(book);
21   } catch (err) {
22     res.status(500).json({ message: err.message });
23   }
24 });
25
26 // 3. CREATE (POST /api/books)
27 router.post('/', async (req, res) => {
28   const newBook = new Book(req.body);
29   try {
30     const savedBook = await newBook.save();
31     res.status(201).json(savedBook);
32   } catch (err) {
33     res.status(400).json({ message: 'Validation failed: ' + err.message });
34 });
35
36 // 4. UPDATE (PUT /api/books/:id)
37 router.put('/:id', async (req, res) => {
38   try {
39     const updatedBook = await Book.findByIdAndUpdate(
40       req.params.id,
41       req.body,
42       { new: true, runValidators: true }
43     );
44     if (!updatedBook) return res.status(404).json({ message: 'Book not found' });
45     res.status(200).json(updatedBook);
46   } catch (err) {
47     res.status(400).json({ message: 'Update failed: ' + err.message });
48   }
49 });
50
51 // 5. DELETE (DELETE /api/books/:id)
52 router.delete('/:id', async (req, res) => {
53   try {
54     const deletedBook = await Book.findByIdAndDelete(req.params.id);
55     if (!deletedBook) return res.status(404).json({ message: 'Book not found' });
56     res.status(200).json({ message: 'Book deleted successfully' });
57   } catch (err) {
58     res.status(500).json({ message: err.message });
59   }
60 });
61
62 module.exports = router;
```

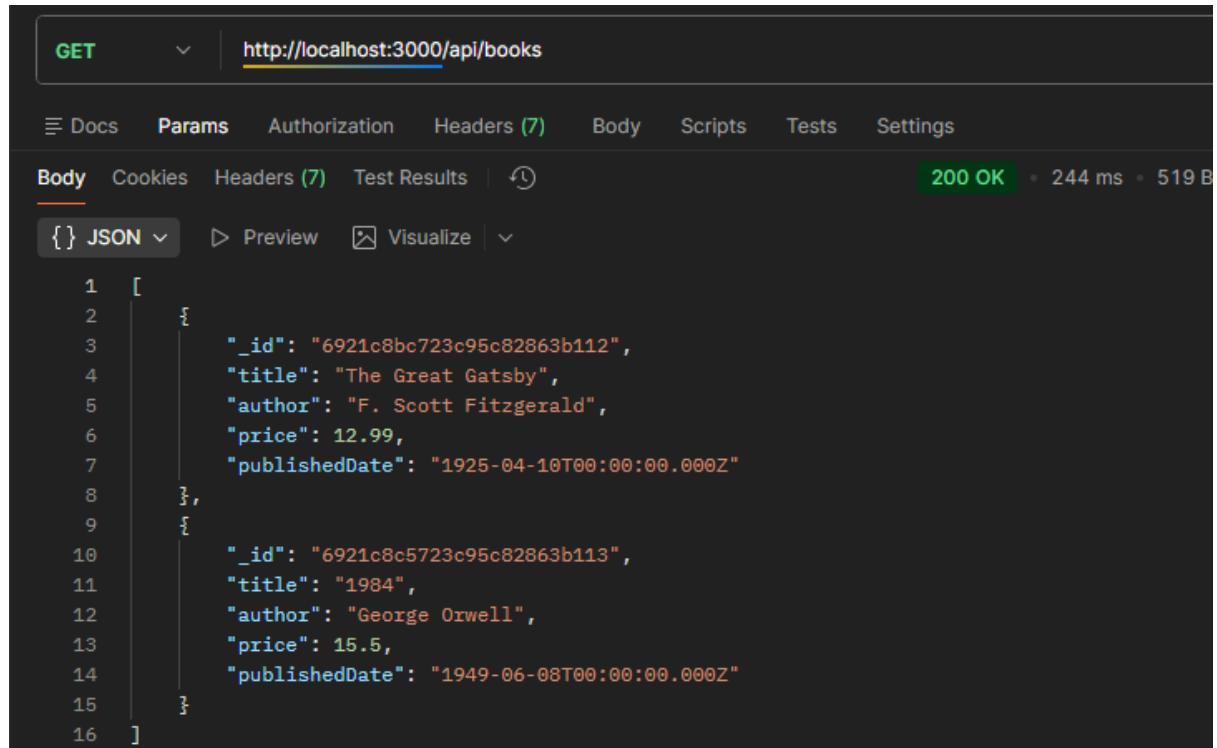
```
37 // 4. UPDATE (PUT /api/books/:id)
38 router.put('/:id', async (req, res) => {
39   try {
40     const updatedBook = await Book.findByIdAndUpdate(
41       req.params.id,
42       req.body,
43       { new: true, runValidators: true }
44     );
45     if (!updatedBook) return res.status(404).json({ message: 'Book not found' });
46     res.status(200).json(updatedBook);
47   } catch (err) {
48     res.status(400).json({ message: 'Update failed: ' + err.message });
49   }
50 });
51
52 // 5. DELETE (DELETE /api/books/:id)
53 router.delete('/:id', async (req, res) => {
54   try {
55     const deletedBook = await Book.findByIdAndDelete(req.params.id);
56     if (!deletedBook) return res.status(404).json({ message: 'Book not found' });
57     res.status(200).json({ message: 'Book deleted successfully' });
58   } catch (err) {
59     res.status(500).json({ message: err.message });
60   }
61 });
62
63 module.exports = router;
```

## Server.js

```
BookManagement > JS server.js > ...
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const bookRoutes = require('./bookRoutes');
4
5  const app = express();
6  const PORT = 3000;
7  const DB_URI = 'mongodb://localhost:27017/libraryDB';
8
9  app.use(express.json());
10
11 // Connection handling
12 mongoose.connect(DB_URI)
13   .then(() => console.log('MongoDB: Connected to libraryDB'))
14   .catch(err => console.error('MongoDB: Connection failed:', err));
15
16 // Route mounting
17 app.use('/api/books', bookRoutes);
18
19 // Start the server
20 app.listen(PORT, () => {
21   console.log(`Server running on http://localhost:\${PORT}`);
22 })
```

POSTMAN:

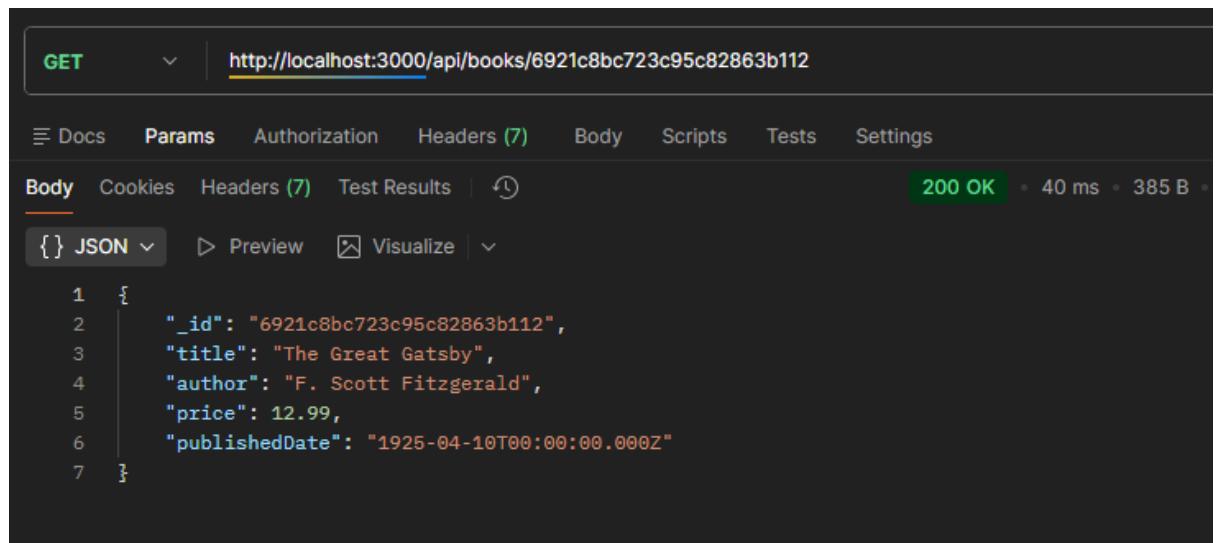
(1) GET all books



The screenshot shows the Postman interface with a successful GET request to `http://localhost:3000/api/books`. The response status is `200 OK` with a response time of `244 ms` and a size of `519 B`. The response body is displayed in JSON format, showing two book objects. The first book has the following details:

```
1 [  
2 {  
3   "_id": "6921c8bc723c95c82863b112",  
4   "title": "The Great Gatsby",  
5   "author": "F. Scott Fitzgerald",  
6   "price": 12.99,  
7   "publishedDate": "1925-04-10T00:00:00.000Z"  
8 },  
9 {  
10  "_id": "6921c8c5723c95c82863b113",  
11  "title": "1984",  
12  "author": "George Orwell",  
13  "price": 15.5,  
14  "publishedDate": "1949-06-08T00:00:00.000Z"  
15 }  
16 ]
```

(2) GET book by ID



The screenshot shows the Postman interface with a successful GET request to `http://localhost:3000/api/books/6921c8bc723c95c82863b112`. The response status is `200 OK` with a response time of `40 ms` and a size of `385 B`. The response body is displayed in JSON format, showing a single book object with the following details:

```
1 {  
2   "_id": "6921c8bc723c95c82863b112",  
3   "title": "The Great Gatsby",  
4   "author": "F. Scott Fitzgerald",  
5   "price": 12.99,  
6   "publishedDate": "1925-04-10T00:00:00.000Z"  
7 }
```

(3) POST new book

The screenshot shows the Postman application interface. At the top, it says "POST" and the URL "http://localhost:3000/api/books". Below the URL, there are tabs for "Docs", "Params", "Authorization", "Headers (9)", "Body" (which is selected), "Scripts", "Tests", and "Settings". Under "Body", there are options for "none", "form-data", "x-www-form-urlencoded", and "raw" (which is selected). To the right of "raw" are buttons for "binary", "GraphQL", and "JSON" (which is selected). The "Body" section contains a JSON payload:

```
1  {
2    "title": "The Hitchhiker's Guide to the Galaxy",
3    "author": "Douglas Adams",
4    "price": 9.99,
5    "publishedDate": "1979-10-12"
6 }
```

Below the body, there are tabs for "Body", "Cookies", "Headers (7)", and "Test Results". The "Test Results" tab shows a green box indicating "201 Created" and "124 ms". At the bottom, there are buttons for "JSON" (selected), "Preview", and "Visualize". The "Visualize" tab shows the full response object:

```
1  {
2    "title": "The Hitchhiker's Guide to the Galaxy",
3    "author": "Douglas Adams",
4    "price": 9.99,
5    "publishedDate": "1979-10-12T00:00:00.000Z",
6    "_id": "6921e52f12ac72105fe407fc",
7    "createdAt": "2025-11-22T16:30:39.062Z",
8    "updatedAt": "2025-11-22T16:30:39.062Z",
9    "__v": 0
10 }
```

#### (4) PUT update book

The screenshot shows a POST request to `http://localhost:3000/api/books/6921e52f12ac72105fe407fc`. The request method is `PUT`. The body contains the following JSON:

```
1 {  
2   |   "price": 14.50  
3 }
```

The response status is `200 OK` with a response time of 31 ms and a size of 484 B. The response body is:

```
1 {  
2   |   "_id": "6921e52f12ac72105fe407fc",  
3   |   "title": "The Hitchhiker's Guide to the Galaxy",  
4   |   "author": "Douglas Adams",  
5   |   "price": 14.5,  
6   |   "publishedDate": "1979-10-12T00:00:00.000Z",  
7   |   "createdAt": "2025-11-22T16:30:39.062Z",  
8   |   "updatedAt": "2025-11-22T16:31:27.841Z",  
9   |   "__v": 0  
10 }
```

#### (5) DELETE book by ID

The screenshot shows a DELETE request to `http://localhost:3000/api/books/6921e52f12ac72105fe407fc`. The response status is `200 OK` with a response time of 18 ms and a size of 274 B. The response body is:

```
1 {  
2   |   "message": "Book deleted successfully"  
3 }
```