Node.js Lab: Building a File-Based Notes API

For this lab, students will create a simple Notes API server using pure Node.js (no Express framework yet) that stores notes in a JSON file.

Requirements

1. Create a server using Node.js's built-in `http` module
2. Implement the following API endpoints:
    - `GET /api/notes` - Get all notes
    - `GET /api/notes/:id` - Get a specific note by ID
    - `POST /api/notes` - Create a new note
    - `PUT /api/notes/:id` - Update a note
    - `DELETE /api/notes/:id` - Delete a note
3. Store notes in a JSON file (`notes.json`)
4. Serve a simple HTML page at the root URL (`/`) that displays the notes
5. Implement proper error handling for missing routes and server errors

Setup Instructions

1. Create a new folder for your project
2. Initialize a new Node.js project with `npm init -y`
3. Create the following files:
    - `server.js` - Main server file
    - `notes.json` - To store the notes (initially empty array)
    - `public/index.html` - Basic HTML for the front-end
    - `public/style.css` - Styling for the front-end
    - `public/script.js` - Frontend JavaScript file

Bonus Challenges (Pick One or More)

1. **Request Logger**: Create a module that logs all incoming requests to a file including timestamp, method, URL, and response status code.
2. **Validation**: Add data validation for the notes (title and content required, character limits, etc.) and return appropriate error messages.
3. **Query Parameters**: Implement query parameter support to filter notes:
    - `GET /api/notes?search=keyword` - Search notes by keyword
    - `GET /api/notes?limit=10&page=2` - Add pagination support

4. **Content-Type Handling**: Make your API accept both JSON and URL-encoded form data by checking the Content-Type header and parsing accordingly.

Hints

- Use `fs.readFile` and `fs.writeFile` to read and write the JSON file
- Parse URL parameters using the `url` module
- You'll need to use `req.on('data', ...)` to collect the request body for POST/PUT requests
- Return appropriate HTTP status codes for different situations
- Remember to set proper content-type headers for different responses (JSON vs HTML)
- To serve static files (similar to express.static), check if the requested URL starts with "/public" and use the `fs` module to read and serve the corresponding file from the public directory
- Use the `path` module to safely resolve file paths when serving static files
- Set the appropriate content-type headers based on file extensions (e.g., .html, .css, .js, .jpg)

Helper BoilerPlate:

Server.js

```
// Import required modules
const http = require('http');
const fs = require('fs');
const path = require('path');
const url = require('url');

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

  console.log(`${req.method} ${pathname}`);

  // TODO: Implement route handling for:
  // 1. Static files (HTML, CSS, JS)
  // 2. API endpoints for notes (GET, POST, PUT, DELETE)

  // Default response for unhandled routes
  res.writeHead(404, { 'Content-Type': 'text/plain' });
  res.end('404 Not Found');
});
```

```
// Start the server
const PORT = process.env.PORT || 3000;
server.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

---

public/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Notes App</title>
  <link rel="stylesheet" href="/public/style.css">
</head>
<body>
  <div class="container">
    <h1>Node.js Notes App</h1>

    <!-- Form to add/edit notes -->
    <div class="form-container">
      <h2>Add New Note</h2>
      <form id="noteForm">
        <!-- TODO: Add form fields for note title and content -->
      </form>
    </div>

    <!-- Container to display notes -->
    <div class="notes-container">
      <h2>My Notes</h2>
      <div id="notesList">
        <!-- Notes will be displayed here -->
      </div>
    </div>
  </div>

  <script src="/public/script.js"></script>
</body>
</html>
```

---

public/style.css

```
/* Basic styles for the Notes App */
body {
  font-family: Arial, sans-serif;
  line-height: 1.6;
```

```
  margin: 0;
  padding: 20px;
}

.container {
  max-width: 800px;
  margin: 0 auto;
}

/* TODO: Add additional styles as needed */
```

---

public/script.js

```
document.addEventListener('DOMContentLoaded', () => {
  // Get DOM elements
  const noteForm = document.getElementById('noteForm');
  const notesList = document.getElementById('notesList');

  // Function to fetch all notes
  const fetchNotes = async () => {
    // TODO: Implement fetching notes from the API
  };

  // Function to render notes to the DOM
  const renderNotes = (notes) => {
    // TODO: Implement rendering notes to the page
  };

  // Function to handle form submission
  const handleSubmit = async (event) => {
    // TODO: Implement form submission
  };

  // Add event listeners
  noteForm.addEventListener('submit', handleSubmit);

  // Initial fetch
  fetchNotes();
});
```

Optional: **logger.js**

```
// Bonus Challenge: Request Logger Module
const fs = require('fs');

// Create a logger module
const logger = {
  logToFile: function(message) {
```

```
    // TODO: Implement logging functionality
  }
};

module.exports = logger
```