

Server Side Scalable Application

Unit-1

1.1 File system module in node.js

The **File System (fs) module** in Node.js allows you to interact with the file system, such as creating, reading, updating, and deleting files.

1.1.1 Inputs from Users

Notes:

- In Node.js, you can take **user input** from the command line using the process.argv array.
- process.argv stores all command-line arguments passed when running a Node.js script.

process.argv is an array:

- process.argv[0] → Node.js executable path.
- process.argv[1] → File path of the script.
- process.argv[2] and beyond → User inputs.

Example 1:

```
// file: userInput.js
const name = process.argv[2]; // First input after filename
console.log(`Hello, ${name}! Welcome to Node.js.`);
```

Run:

```
node userInput.js Chirag
```

Output:

```
Hello, Chirag! Welcome to Node.js.
```

Example 2: Sum of Two Numbers

```
// file: sumInput.js
const num1 = parseInt(process.argv[2]);
const num2 = parseInt(process.argv[3]);
const sum = num1 + num2;

console.log(`Sum of ${num1} and ${num2} is: ${sum}`);
```

Run:

```
node sumInput.js 10 20
```

Output:

Sum of 10 and 20 is: 30

1.1.2 Pass Multiple Arguments with Yargs

Notes:

- **Yargs** is a powerful npm package used to parse command-line arguments in a more structured and readable way.
- It makes handling multiple arguments easier.
- You can set default values.
- You can validate argument types.
- You can create command-based CLI apps.

Installation:

```
npm install yargs
```

Example:

```
// file: yargsExample.js
```

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

```
const args = yargs
```

```
  .option('name', { describe: 'Your Name', demandOption: true, type: 'string' })
```

```
  .option('age', { describe: 'Your Age', demandOption: true, type: 'number' })
```

```
  .argv;
```

```
console.log(`Hello, ${args.name}. You are ${args.age} years old.`);
```

Run:

```
node yargsExample.js --name=Chirag --age=30
```

Output:

Hello, Chirag. You are 30 years old.

List of Important Arguments

Argument	Purpose	Example
describe	Description shown in help	describe: 'Your name'
type	Expected data type: string, number, boolean	type: 'string'

Argument	Purpose	Example
demandOption	Makes the argument required	demandOption: true
default	Provides a default value	default: 'Guest'
alias	Shortcut/alternative key	alias: 'n'
boolean	Marks the argument as a true/false flag	boolean: true
number	Specifies argument should be a number	number: true
string	Specifies argument should be a string	string: true

Explanation with Examples:

1. describe

- Shown in the --help menu.
- ```
.option('name', { describe: 'Your full name' })
```
- 

#### 2. type

- Defines expected data type: 'string', 'number', 'boolean'.
- ```
.option('age', { type: 'number' })
```
-

3. demandOption

- Makes the argument **mandatory**.
- ```
.option('city', { demandOption: true })
```
- 

#### 4. default

- If not provided by the user, the **default value is used**.
- ```
.option('role', { default: 'guest' })
```
-

5. alias

- Shortcut for the option.
- ```
.option('name', { alias: 'n' })
```

✓ Usage:

node app.js --name=Chirag

# or

node app.js -n Chirag

---

---

## 6. boolean

- Flag option, no value needed.

```
.option('isAdmin', { boolean: true })
```

✓ Usage:

```
node app.js --isAdmin
```

---

## 7. number / string (alternative)

- You can directly specify:

```
.option('price', { number: true })
```

```
.option('product', { string: true })
```

---

### Sample Example Using All:

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

```
const args = yargs
```

```
 .option('name', { describe: 'Your name', alias: 'n', type: 'string', demandOption: true })
```

```
 .option('age', { describe: 'Your age', type: 'number', default: 18 })
```

```
 .option('isAdmin', { describe: 'Admin access', boolean: true })
```

```
 .argv;
```

```
console.log(args);
```

---

### 1.1.3 File System Module

- Node.js provides the **fs (File System) module** to handle file operations.

- Common methods:

- fs.writeFileSync → Write or create a file (synchronous)

- fs.readFileSync → Read file contents (synchronous)

- fs.appendFileSync → Append data to a file

- fs.unlinkSync → Delete a file

#### Example: Basic File Operations

```
// file: fileSystemExample.js
```

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

```
// Writing to a file
```

```
fs.writeFileSync('example.txt', 'This is the initial content.');
```

---

```
// Reading from the file
const data = fs.readFileSync('example.txt', 'utf8');
console.log('File Content:', data);

// Appending data
fs.appendFileSync('example.txt', '\nThis is the appended content.');
```

```
// Reading again after appending
const updatedData = fs.readFileSync('example.txt', 'utf8');
console.log('Updated File Content:', updatedData);
```

**Output:**

File Content: This is the initial content.  
Updated File Content: This is the initial content.  
This is the appended content.

---

#### 1.1.4 Operations Associated with File System Module

**Notes:**

Here are the key operations:

| Operation     | Method         | Description                        |
|---------------|----------------|------------------------------------|
| Create        | writeFileSync  | Creates a new file and writes data |
| Read          | readFileSync   | Reads content from a file          |
| Update        | appendFileSync | Adds data to an existing file      |
| Delete        | unlinkSync     | Removes a file from the system     |
| Exists        | existsSync     | Check if file exists.              |
| Create folder | mkdirSync      | Create a folder.                   |

---

**Example: Full CRUD Operations**

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

// 1. Create File
fs.writeFileSync('data.txt', 'Initial file content');
```

```

// 2. Read File
let fileData = fs.readFileSync('data.txt', 'utf8');
console.log('File Content:', fileData);

// 3. Update File
fs.appendFileSync('data.txt', '\nAdded new content');

// Read Updated File
fileData = fs.readFileSync('data.txt', 'utf8');
console.log('Updated File Content:', fileData);

// 4. Delete File
fs.unlinkSync('data.txt');
console.log('File deleted successfully.');
```

```

// Create folder
if (!fs.existsSync('myFolder')) {
 fs.mkdirSync('myFolder');
 console.log('Folder created successfully.');
```

```

}

// Create file inside folder
fs.writeFileSync('myFolder/info.txt', 'Folder file content');
```

```

// Read file from folder
const content = fs.readFileSync('myFolder/info.txt', 'utf8');
console.log('File content:', content);
```

## 1.2 JSON Data, HTTP Server and Client

### JSON Data

- **JSON (JavaScript Object Notation)** is a lightweight data-interchange format.
- It is easy for humans to read and write and easy for machines to parse and generate.

#### Example JSON:

```

{
 "name": "John",
```

```
"age": 30,
"city": "New York"
}
```

### Reading JSON in Node.js:

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

const data = fs.readFileSync('data.json');
const jsonData = JSON.parse(data);
console.log(jsonData);
```

---

## HTTP Server and Client

### HTTP Server Example (Node.js Built-in Module)

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

const server = http.createServer((req, res) => {
 if (req.url === '/home') {
 res.writeHead(200, {'Content-Type': 'application/json'});
 res.end(JSON.stringify({message: "Welcome to Home Page"}));
 } else {
 res.writeHead(404);
 res.end('Page Not Found');
 }
});

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

---

### 1.2.1 Sending and Receiving Events with EventEmitters

#### What is EventEmitter?

- Node.js has a built-in module called events.
- The EventEmitter class allows us to **create and handle custom events**.

#### Example:

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

```
// Create an event handler
const greet = () => {
 console.log('Hello! Event Triggered Successfully.');
```

```
}

// Assign the handler to an event
eventEmitter.on('sayHello', greet);
```

```
// Trigger the event
eventEmitter.emit('sayHello');
```

You can pass data with events:

```
eventEmitter.on('greetUser', (name) => {
 console.log(`Hello ${name}! Welcome.`);
});
```

```
eventEmitter.emit('greetUser', 'Chirag');
```

---

### 1.2.2 Express Framework – Run a Web Server Using Express Framework

#### Introduction to Express:

- Express is a minimal and flexible Node.js web application framework.
- Makes it easier to build web servers and APIs.

Express is a **minimal, fast, and flexible web framework** for Node.js that helps developers:

- Build web servers.
- Create APIs.
- Handle HTTP requests and responses.
- Manage routing.
- Use middleware for processing requests.

#### Why Use Express?

- Simple and quick to set up.
- Provides powerful routing.
- Supports middleware for request handling.
- Easily integrates with databases.
- Good for creating REST APIs.



### Install Express:

```
npm install express
```

### Basic Express Server Example:

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

app.get('/', (req, res) => {
 res.send('Hello World from Express!');
});

app.get('/about', (req, res) => {
 res.json({ message: 'About Page' });
});

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

---

### 1.2.3 Routes in Express

#### What are Routes?

- Routes are **paths** that the server listens to for incoming requests.
- Example: /home, /about

#### Example of Multiple Routes:

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

```
app.post('/submit', (req, res) => {
 res.send('Form Submitted');
});
```

```
app.put('/update', (req, res) => {
 res.send('Data Updated');
});
```

```
app.delete('/delete', (req, res) => {
```

---

```
res.send('Data Deleted');
});
```

## Handling Request & Response in Express

### Request Object (req)

- req.query – Get query parameters: /search?name=John
- req.params – Get route parameters: /user/:id
- req.body – Get POST request data

### Response Object (res)

- res.send() – Send text or HTML response
- res.json() – Send JSON response
- res.status() – Set HTTP status code

### Example:

```
app.get('/example', (req, res) => {
 res.status(200).json({ message: 'Everything is OK!' });
});
```

## CRUD API using Express.js

We will:

- **Create** a new item (POST)
- **Read** all items (GET)
- **Update** an item (PUT)
- **Delete** an item (DELETE)

We will use a **local in-memory array** as a temporary database.

---

### Project Setup

#### 1. Initialize Project

```
mkdir express-crud-api
cd express-crud-api
npm init -y
npm install express
```

---

### CRUD API Code

```
Create a file: app.js
const express = require('express');
const app = express();
const PORT = 3000;
```

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

// Sample in-memory data store
let users = [
 { id: 1, name: 'John Doe', email: 'john@example.com' },
 { id: 2, name: 'Jane Smith', email: 'jane@example.com' }
];

// READ: Get all users
app.get('/users', (req, res) => {
 res.json(users);
});

// READ: Get a single user by ID
app.get('/users/:id', (req, res) => {
 const user = users.find(u => u.id === parseInt(req.params.id));
 if (!user) return res.status(404).send('User not found');
 res.json(user);
});

// CREATE: Add a new user
app.post('/users', (req, res) => {
 const { name, email } = req.body;
 const newUser = {
 id: users.length + 1,
 name,
 email
 };
 users.push(newUser);
 res.status(201).json(newUser);
});

// UPDATE: Modify existing user
app.put('/users/:id', (req, res) => {
 const user = users.find(u => u.id === parseInt(req.params.id));
 if (!user) return res.status(404).send('User not found');
```

```

 const { name, email } = req.body;
 user.name = name || user.name;
 user.email = email || user.email;

 res.json(user);
 });

// DELETE: Remove user
app.delete('/users/:id', (req, res) => {
 const userIndex = users.findIndex(u => u.id === parseInt(req.params.id));
 if (userIndex === -1) return res.status(404).send('User not found');

 const deletedUser = users.splice(userIndex, 1);
 res.json(deletedUser[0]);
});

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

```



### How to Run the Project

node app.js

Test the API using **Postman** or **cURL**.



### API Endpoints Summary

| Method | Endpoint   | Description         |
|--------|------------|---------------------|
| GET    | /users     | Get all users       |
| GET    | /users/:id | Get a specific user |
| POST   | /users     | Create a new user   |
| PUT    | /users/:id | Update a user       |
| DELETE | /users/:id | Delete a user       |

---

### 💡 Example Postman Requests

#### 1. GET All Users

- GET `http://localhost:3000/users`

#### 2. POST Create User

- POST `http://localhost:3000/users`

```
{
 "name": "Alice",
 "email": "alice@example.com"
}
```

#### 3. PUT Update User

- PUT `http://localhost:3000/users/1`

```
{
 "name": "John Updated",
 "email": "johnupdated@example.com"
}
```

#### 4. DELETE User

- DELETE `http://localhost:3000/users/1`

---

### 1.2.4 Deploy Application Using PM2 and Nginx

#### What is PM2?

- PM2 is a **process manager** for Node.js applications.
- Keeps your app running forever.

It allows you to:

- Keep your app **always running**
- Automatically **restart** it on failure or crash
- Run it in the **background** (daemon)
- Easily **monitor, log, and manage** multiple apps

#### Why Use PM2?

| Feature            | Benefit                       |
|--------------------|-------------------------------|
| Background Process | No need to keep terminal open |
| Auto Restart       | On crash or file change       |

| Feature            | Benefit                            |
|--------------------|------------------------------------|
| Logs Management    | View output and errors easily      |
| Clustering         | Use multi-core systems efficiently |
| Process Monitoring | View RAM, CPU usage                |
| Startup Script     | App starts after server reboot     |

#### **Install PM2:**

```
npm install -g pm2
```

#### **Start App with PM2:**

```
pm2 start app.js
```

#### **Other Useful PM2 Commands:**

```
pm2 list # Shows running apps
pm2 restart app # Restart your app
pm2 stop app # Stop your app
pm2 delete app # Remove app from PM2
pm2 startup # Auto-start on system reboot
pm2 save # Save the PM2 process list
pm2 logs # Show logs
```

---

### **Nginx Configuration for Node.js Reverse Proxy**

Nginx (pronounced "engine-X") is a powerful, open-source web server and reverse proxy. It's known for its high performance, stability, and efficiency in handling a large number of concurrent connections. Beyond serving as a web server, Nginx is widely used for reverse proxying, load balancing, caching, and even as a mail proxy.

#### **Key Features and Functions:**

- **Web Server:**

Nginx can serve static content (like HTML, CSS, JavaScript, and images) directly to users.

- **Reverse Proxy:**

It acts as an intermediary between clients and servers, forwarding client requests to appropriate backend servers. This helps improve security, performance, and scalability.

- **Load Balancer:**

Nginx can distribute traffic across multiple servers, preventing any single server from becoming overloaded and ensuring high availability.

- **HTTP Cache:**

It can cache frequently accessed content, reducing the load on backend servers and improving response times.

- **Mail Proxy:**

Nginx can also act as a proxy server for email protocols like IMAP, POP3, and SMTP.

- **Event-Driven Architecture:**

Nginx uses an event-driven, asynchronous architecture, which allows it to handle many connections simultaneously with minimal resources. This is a key factor in its high performance and scalability.

- **High Performance and Stability:**

Nginx is designed for maximum performance and stability, making it a popular choice for high-traffic websites and applications.

- **Open Source:**

Nginx is open-source software, meaning it's free to use and modify.

In essence, Nginx is a versatile tool that can be used for a wide range of tasks related to web serving and network traffic management.

## **1. Install Nginx:**

```
sudo apt update
```

```
sudo apt install nginx
```

## **2. Basic Nginx Config:**

Open Nginx configuration:

```
sudo nano /etc/nginx/sites-available/default
```

Add the following:

```
server {
 listen 80;

 server_name your_domain_or_ip;

 location / {
 proxy_pass http://localhost:3000;
 proxy_http_version 1.1;
 proxy_set_header Upgrade $http_upgrade;
```

```
 proxy_set_header Connection 'upgrade';
 proxy_set_header Host $host;
 proxy_cache_bypass $http_upgrade;
 }
}
```

### **3. Restart Nginx:**

```
sudo systemctl restart nginx
```

### **4. Test Deployment:**

Visit your server IP or domain in the browser → App should be live!