1. Introduction to Node.js

What is Node.js?

- Node.js is a runtime environment that allows JavaScript to run on the server side.
- Built on Google Chrome's V8 JavaScript engine.
- Event-driven and non-blocking I/O model makes it lightweight and efficient.

Features of Node.js

- **Asynchronous and Event-Driven**: Handles multiple requests concurrently.
- **Fast Execution**: Uses the V8 engine for fast JavaScript execution.
- **Single-Threaded**: Designed for scalability using a single-threaded model.
- **Cross-Platform**: Runs on Windows, macOS, and Linux.

Installing Node.js

1. Download from Node.js Official Website.
2. Install and verify:

```
bash
Copy code
node -v
npm -v
```

First Program

```
javascript
Copy code
console.log("Hello, Node.js!");
```

2. Node.js Architecture

Event Loop

- Core of Node.js; handles all asynchronous operations.
- Phases:
  - Timers
  - I/O callbacks
  - Idle, prepare
  - Poll
  - Check
  - Close callbacks

Callback, Promises, and Async/Await

- **Callback**:

```
javascript
const fs = require('fs');
fs.readFile('file.txt', (err, data) => {
    if (err) throw err;
    console.log(data.toString());
});
```

- **Promises**:

```
javascript
Copy code
const readFile = require('fs').promises;
readFile('file.txt')
    .then(data => console.log(data.toString()))
    .catch(err => console.error(err));
```

- **Async/Await**:

```
javascript
Copy code
const readFile = require('fs').promises;
async function read() {
    try {
        const data = await readFile('file.txt');
        console.log(data.toString());
    } catch (err) {
        console.error(err);
    }
}
read();
```

3. Core Modules

File System (fs)

- Read and write files:

```
javascript
const fs = require('fs');
fs.writeFileSync('test.txt', 'Hello, Node.js!');
const data = fs.readFileSync('test.txt', 'utf8');
console.log(data);
```

HTTP

- Create a basic server:

```
javascript
const http = require('http');
const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, World!');
});
server.listen(3000, () => console.log('Server running on port 3000'));
```

Path

- Handle file paths:

```
javascript
```

```
const path = require('path');
console.log(path.basename('/folder/file.txt')); // Output: file.txt
```

4. Package Management with NPM

Using NPM

- Initialize a project:

```
bash
npm init -y
```

- Install a package:

```
bash
npm install express
```

Creating a Custom Package

1. Add code in `index.js`:

```
javascript
module.exports = () => console.log("Custom package");
```

2. Publish using:

```
bash
npm publish
```

5. Express.js Framework

Setting up Express.js

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

```
app.get('/', (req, res) => res.send('Hello, Express!'));
app.listen(3000, () => console.log('Server running on port 3000'));
```

Middleware

- Example middleware:

```
javascript
Copy code
app.use((req, res, next) => {
    console.log(`Request received: ${req.method} ${req.url}`);
    next();
});
```

6. Database Integration

MongoDB

- Connect to MongoDB:

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/test', { useNewUrlParser:
true, useUnifiedTopology: true });
```

- Define and use a model:

```
const User = mongoose.model('User', { name: String });
const user = new User({ name: 'John' });
user.save().then(() => console.log('User saved'));
```

7. RESTful APIs

Example API

```
javascript
const express = require('express');
const app = express();
app.use(express.json());

let users = [{ id: 1, name: 'John Doe' }];

app.get('/users', (req, res) => res.json(users));
app.post('/users', (req, res) => {
    users.push(req.body);
    res.status(201).json(req.body);
});
app.listen(3000, () => console.log('API running on port 3000'));
```

8. Real-Time Applications

Using WebSockets with Socket.io

```
javascript
const http = require('http');
const socketIo = require('socket.io');

const server = http.createServer();
const io = socketIo(server);

io.on('connection', (socket) => {
    console.log('User connected');
    socket.on('message', (msg) => console.log(msg));
});

server.listen(3000, () => console.log('WebSocket server running on
port 3000'));
```

9. Authentication

JSON Web Tokens (JWT)

```
javascript
const jwt = require('jsonwebtoken');
const token = jwt.sign({ userId: 123 }, 'secretKey', { expiresIn:
'1h' });
console.log(token);
```

10. Deployment

Hosting on Heroku

1. Create `Procfile`:

```
makefile
web: node index.js
```

2. Deploy:

```
bash
Copy code
git init
heroku create
git push heroku main
```

Exercises and Projects

- **Exercises**:
  - Build a to-do list API with Express.js.
  - Integrate a MongoDB database for storing data.

- **Projects**:
    - Chat application using WebSockets.
    - E-commerce backend with authentication.