SECTION 2 — Node.js and Project Initialization

1. What Is Node.js?

Node.js is a JavaScript runtime environment that allows JavaScript to run outside the browser.

Traditionally, JavaScript was only used for frontend development. Node.js enables developers to use JavaScript for backend development.

Node.js is built on Chrome’s V8 engine and is designed to handle:

- Server-side logic
- File system operations
- Network requests
- Database connections
- API handling

With Node.js, JavaScript can be used for full-stack development.

2. Why Use Node.js for Backend?

Node.js is widely used because:

- It uses JavaScript on both frontend and backend.
- It is lightweight and efficient.
- It handles asynchronous operations effectively.
- It has a large ecosystem of packages through npm.

Node.js is especially suitable for:

- REST APIs
- Real-time applications
- Scalable backend services

3. What Is npm?

npm stands for Node Package Manager.

It is:

- A package manager for Node.js
- A registry of reusable open-source packages
- A dependency management tool

npm helps us:

- Install external libraries
- Manage project dependencies
- Run scripts
- Maintain version control of packages

4. Initializing a Node.js Project

To start a backend project, we first initialize a Node.js application.

Command:

npm init -y

What Does This Command Do?

It creates a file called:

package.json

The -y flag automatically accepts default values.

5. Understanding package.json

The package.json file is the configuration file for a Node.js project.

It contains:

- Project name
- Version
- Entry point file
- Scripts
- Dependencies
- Developer dependencies

Example structure:

```
{
  "name": "monsta-backend",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {},
  "devDependencies": {}
}
```

6. Entry Point of the Application

The main property in package.json defines the entry file.

Example:

"main": "server.js"

This means the backend starts execution from:

server.js

7. Understanding Node Execution

To run the server manually:

node server.js

Node reads the file and executes it.

Later, we will configure scripts to simplify this process.

8. Project Scripts

Inside package.json, we define scripts:

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

These allow us to run:

npm start

npm run dev

Scripts improve development workflow.

9. Installing Dependencies

In backend development, we rely on external packages.

To install a package:

npm install express

To install a development-only package:

npm install nodemon --save-dev

10. Difference Between Dependencies and Dev Dependencies

Dependencies

These are required for the application to run in production.

Examples:

- express
- mongoose
- bcryptjs
- jsonwebtoken
- cors
- dotenv
- cookie-parser

They are listed under:

"dependencies": {}

Dev Dependencies

These are used only during development.

Example:

- nodemon

Listed under:

"devDependencies": {}

Why Separate Them?

Because:

- Production servers do not need development tools.
- Keeps production environment lightweight.
- Follows best practices.

11. Example Dependencies for This Project

Below are the core dependencies used in this backend.

Express

Backend web framework.

Handles:

- Routing
- Middleware
- Request and response

Mongoose

Library to interact with MongoDB.

Provides:

- Schema validation
- Model structure
- Query abstraction

dotenv

Loads environment variables from a .env file.

Prevents hardcoding sensitive values like:

- Database URLs
- JWT secrets

jsonwebtoken

Used to:

- Generate access tokens
- Verify tokens

Used for authentication.

bcryptjs

Used for:

- Hashing passwords
- Comparing hashed passwords

Ensures password security.

cors

Enables cross-origin requests.

Required when frontend and backend run on different ports.

cookie-parser

Used to read cookies from incoming requests.

Important for handling refresh tokens.

nodemon (Development Only)

Automatically restarts the server when files change.

Used only during development.

12. Summary of Section 2

By the end of this section, students should understand:

- What Node.js is
- What npm does
- How to initialize a backend project
- What package.json contains
- The difference between dependencies and devDependencies
- Why we install specific backend libraries

This establishes the technical foundation required before structuring the application.