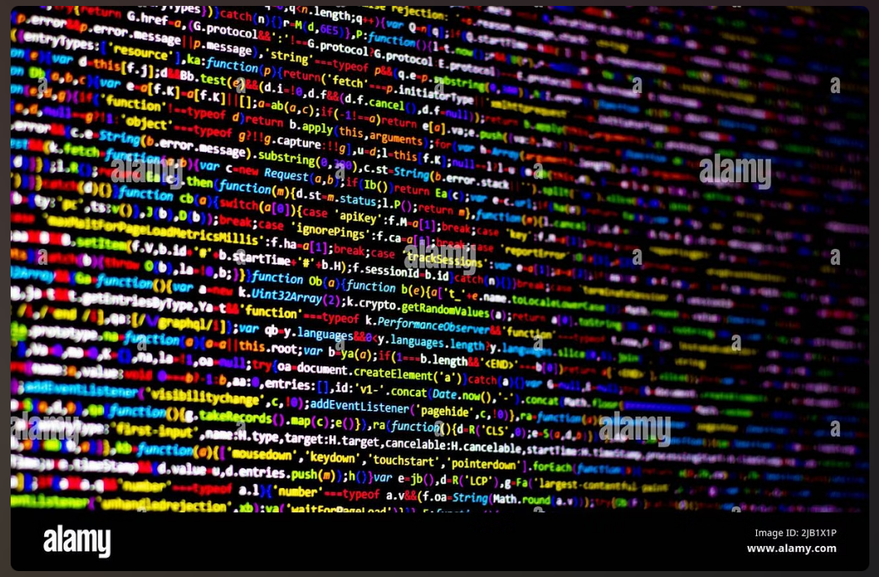


Welcome to the World of Node.js

Welcome to the world of Node.js. You're about to embark on a journey into server-side JavaScript, learning how to build powerful web applications. This presentation will guide you through the basics, from installation to building your first app.

by Ojas Joshi

What is Node.js?

Node.js is a JavaScript runtime environment that allows you to run JavaScript on the server side. Unlike traditional server-side languages, Node.js employs an asynchronous, event-driven architecture, making it efficient for handling numerous requests simultaneously.

JavaScript on the Server

Node.js enables you to use JavaScript on both the front-end and back-end, simplifying development and increasing consistency.

Event-Driven, Non-blocking I/O

Node.js can handle multiple tasks concurrently without waiting for one to finish, making it ideal for real-time applications.

Built on Chrome's V8 Engine

Node.js leverages Google Chrome's fast and efficient V8 JavaScript engine, ensuring performance and speed.

Use Cases and Benefits of Node.js

Node.js is perfect for a variety of applications, ranging from simple web servers to complex real-time systems. Its speed and scalability make it ideal for handling large numbers of users or requests simultaneously.

Real-Time Applications

Node.js excels at building real-time applications like chat apps, online games, and live dashboards, where data needs to be updated instantly.

- Live chat
- Video conferencing
- Online gaming

Scalability

Node.js is highly scalable, meaning it can handle increasing traffic and user loads without compromising performance. This is essential for web applications with a growing user base.

- Streaming services
- Social media platforms
- E-commerce sites

Ease of Development

Using JavaScript for both front-end and back-end simplifies development and promotes code reuse. Node.js has a vast ecosystem of libraries and tools available through npm.

- Consistent development environment
- Rich set of libraries and frameworks
- Strong community support

Companies Using Node.js

Node.js has gained popularity in the industry and is used by some of the biggest tech companies in the world. Its speed, scalability, and ease of development make it a compelling choice for various applications.

Company	Application
Netflix	Streaming platform
PayPal	Payment processing
LinkedIn	Social networking platform

Installing Node.js

Installing Node.js is a straightforward process. You can download the appropriate installer for your operating system from the official Node.js website. The installer comes bundled with npm (Node Package Manager), a powerful tool for managing dependencies.

1

Download the Installer

Visit the Node.js website and download the LTS version of the installer for your operating system.

2

Run the Installer

Double-click the installer file and follow the on-screen instructions. Ensure that npm is selected for installation.

3

Verify the Installation

Open a terminal or command prompt and run the following commands to confirm the installation:

4

node -v

This will display the installed Node.js version.

5

npm -v

This will display the installed npm version.

Understanding Node.js Modules

Node.js modules are like building blocks that allow you to organize and reuse code. There are two types of modules: core modules (built-in) and custom modules (created by you).

1 Core Modules

Core modules are pre-installed with Node.js and provide essential functionality for common tasks.

2 fs

For working with the file system, such as reading and writing files.

3 http

For creating web servers that can handle requests and send responses.

4 path

For manipulating file paths and ensuring consistency across different operating systems.

5 events

For handling events, allowing you to trigger actions when specific events occur.

Creating Custom Modules

Custom modules allow you to organize your code into reusable components, making your projects more maintainable and scalable. They enable you to create specialized functions that are specific to your application.

Create a Module File

Create a JavaScript file for your module, e.g., myModule.js.

Define Functions

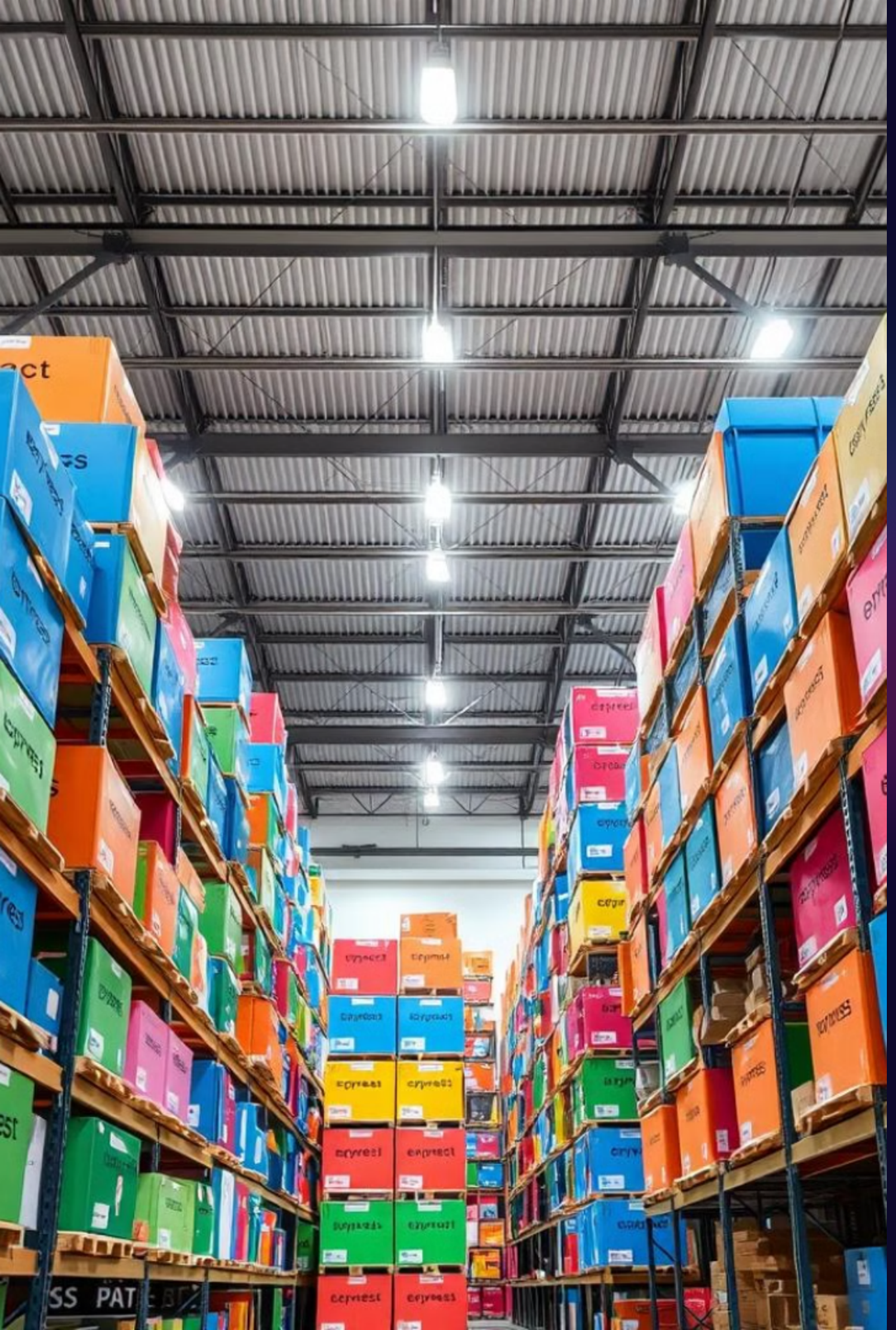
Define the functions you want to export from the module.

Export Functions

Use module.exports to export the functions so they can be used in other files.

Import the Module

Use require() to import your module in other files where you need to use its functions.



Node Package Manager (npm)

npm (Node Package Manager) is a powerful tool for managing dependencies in Node.js projects. It allows you to easily install, update, and remove packages, which are pre-built modules created by the community.

1. ## npm init

   Initializes your Node.js project by creating a package.json file, which lists all the packages you need.

2. ## npm install

   Installs packages from the npm registry. You can specify the package name and version, e.g., npm install express@4.17.1.

3. ## npm update

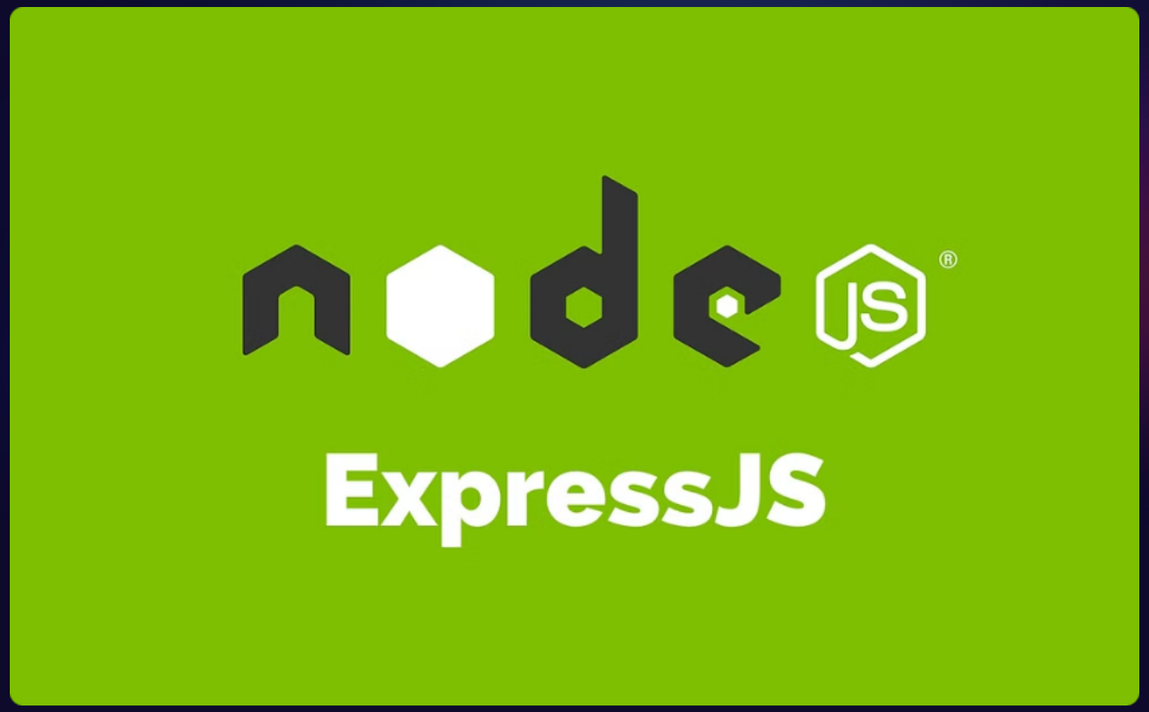
   Updates installed packages to their latest versions.

4. ## npm uninstall

   Removes installed packages from your project. You can specify the package name, e.g., npm uninstall express.

Popular npm Packages

npm has a vast collection of packages that you can use to extend the functionality of your Node.js applications. These packages offer pre-built solutions for various tasks, saving you time and effort.



Express.js

A popular web framework for building robust and scalable web applications.

Mongoose

A MongoDB ODM (Object Document Mapper) that simplifies interaction with MongoDB databases.

Next Steps: Building Your First Node.js App

Now that you have a basic understanding of Node.js, its modules, and npm, you're ready to start building your first Node.js application. The next step is to choose a project idea and explore the vast possibilities offered by this powerful technology.

Simple Web Server

Start with creating a basic web server that serves static content or handles basic HTTP requests.

REST API

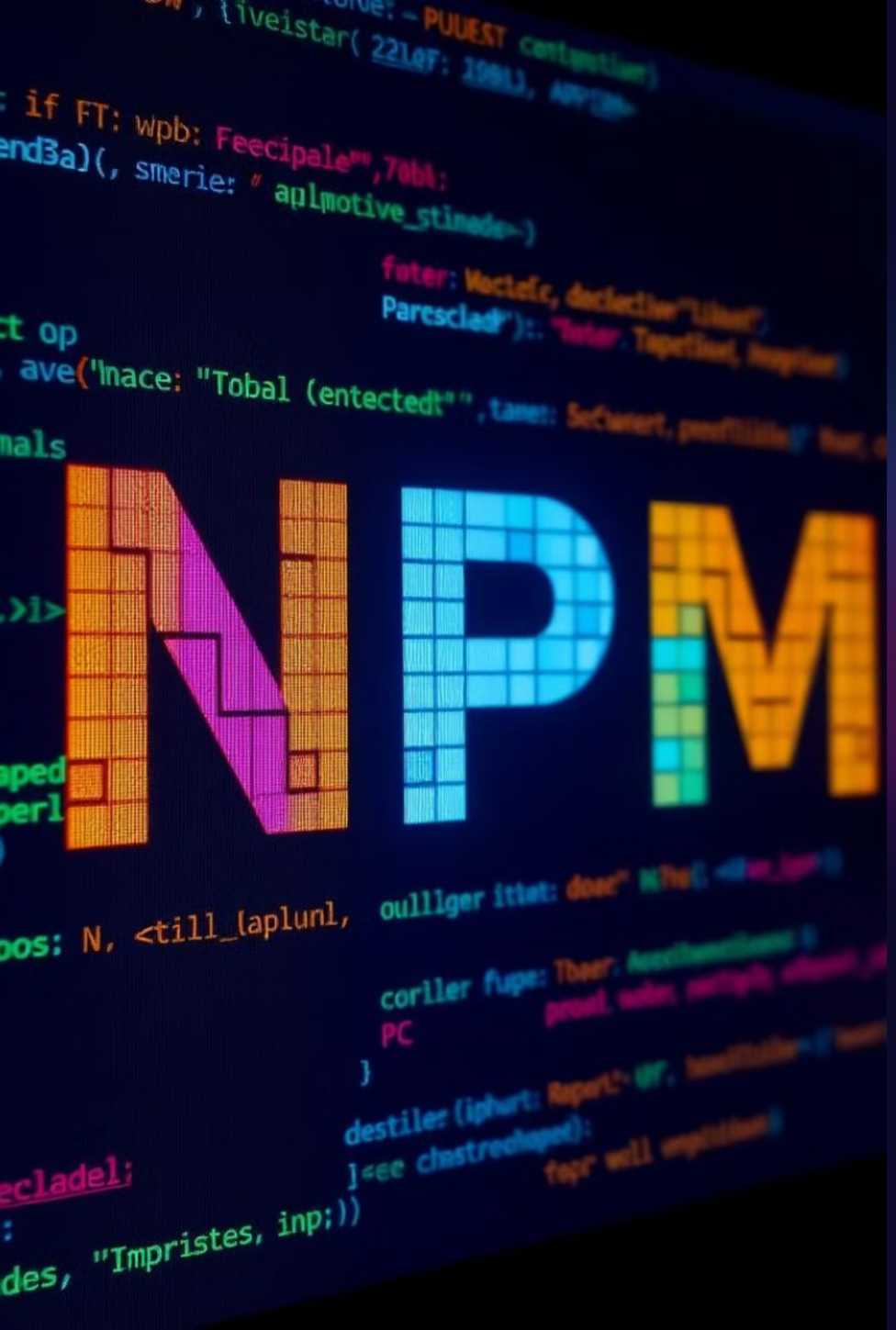
Build a RESTful API to interact with data sources like databases or external services.

Real-Time Chat App

Challenge yourself by building a real-time chat application using Node.js and WebSockets.

npm: The Package Manager for Node.js

npm is a package manager that allows you to manage libraries (dependencies) for your Node.js projects. Think of npm like Amazon for developers. You want to build something? Head to npm, search for the tool (package) you need, and with one command (npm install), you’ll have it in your project.



Recap: npm Basics

1. **npm init**

   Initializes a project with a package.json file.

2. **npm install**

   Installs packages from npm.

3. **npm update**

   Updates installed packages to their latest versions.

4. **npm uninstall**

   Removes packages from your project.

Creating a Web Server with Node.js

1

Setting Up the Project

Create a folder for your project and a server.js file.

2

Creating a Simple HTTP Server

Import the http module and create a server using http.createServer.

3

Running the Server

Run node server.js to start the server. Open your browser to http://localhost:3000.

Handling Routes

1

req.url

Represents the URL path the client requested.

2

Check the URL

Use if/else statements to respond differently based on the URL.

3

Send Response

Send an HTML message back to the client.

File System Module in Node.js

fs Module

Provides methods to interact with the file system.

Asynchronous Operations

Allows other tasks to run while file operations are in progress.

fs.readFile

Reads the content of a file asynchronously.

fs.writeFile

Writes content to a file asynchronously.

Reading Files Asynchronously

fs.readFile

Reads the content of a file asynchronously.

1. File name
2. Encoding
3. Callback function

Callback Function

Handles the result of the file read operation.

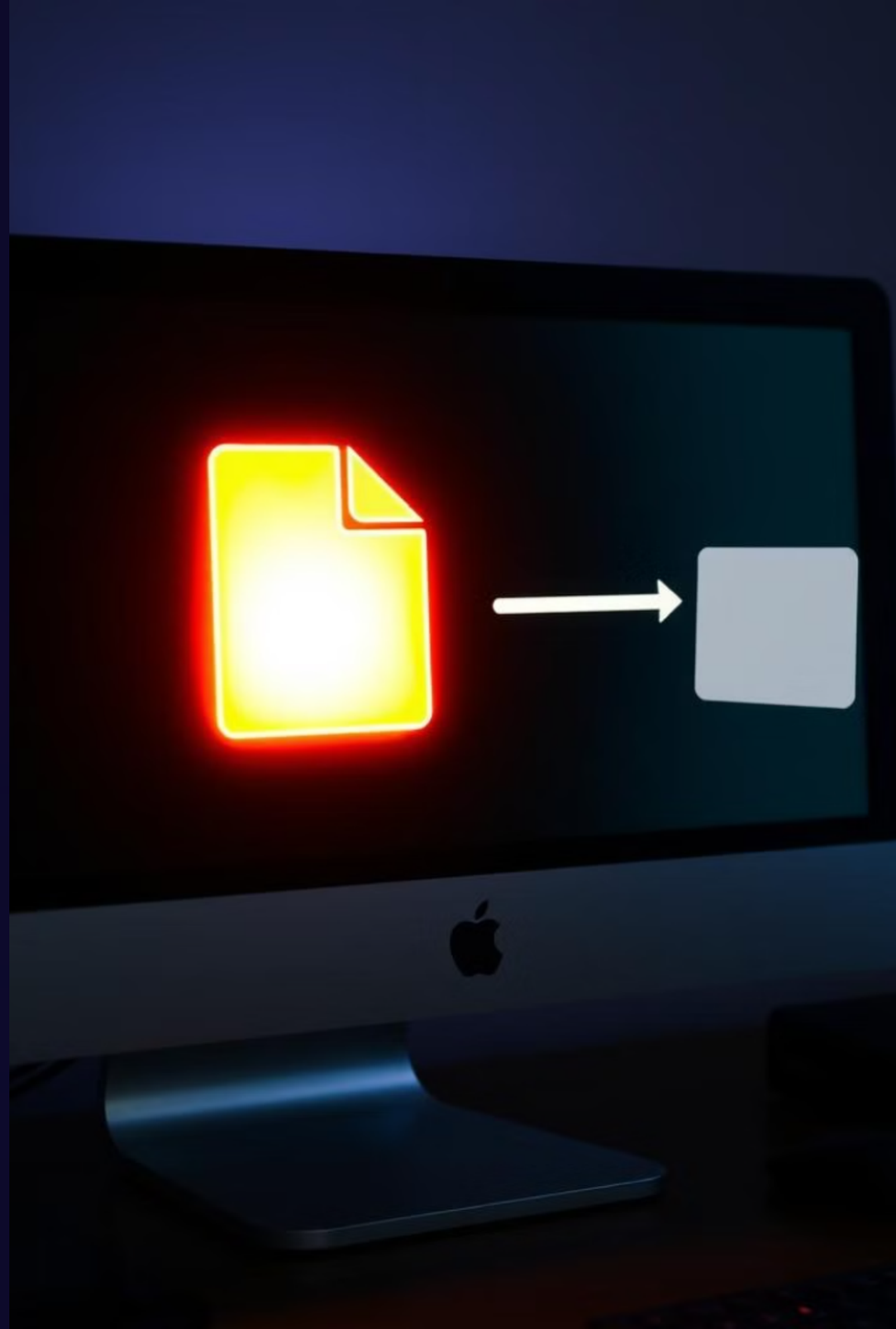
1. Error (if any)
2. File data (if successful)

Example

fs.readFile('example.txt', 'utf8', (err, data) => { ... });

Writing Files Asynchronously

fs.writeFile	Writes content to a file asynchronously.
File name	The name of the file to write to.
Content	The text to write to the file.
Callback function	Handles the result of the write operation.



Express.js Framework

Routing

Handles different URL requests.

Middleware

Handles request data before sending a response.

Scalability

Makes it easier to build large-scale apps.

Setting Up Express

1

Create Project Folder

Create a folder for your project and a package.json file.

2

Install Express

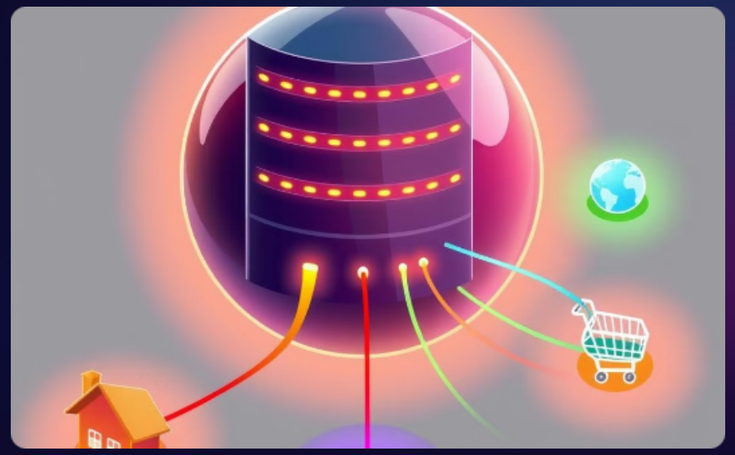
Run npm install express to install Express.js.

3

Create a Basic Express Application

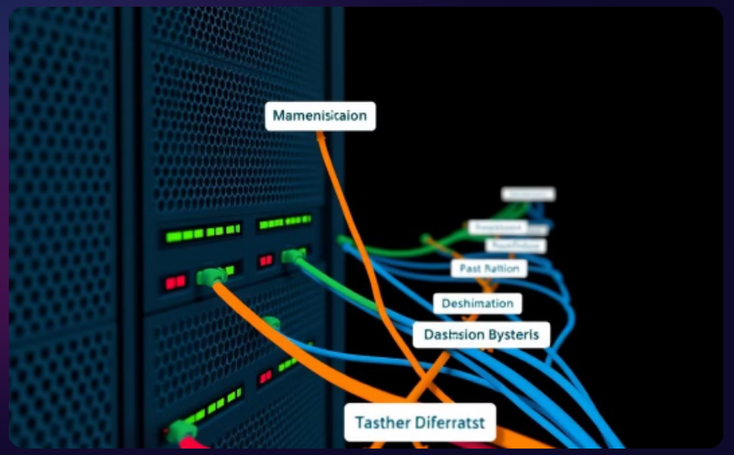
Import Express and create an Express app instance.

Handling Routes in Express



app.get()

Defines a route for GET requests.



app.post()

Defines a route for POST requests.

Route Parameters

Use :name to create dynamic routes.

Middleware in Express.js

Think of middleware as security checkpoints when you're entering a building.

Before you reach your destination (route), you pass through multiple checkpoints that check your ID (authorization), scan you for metal (data validation), and sometimes direct you to the right path (routing).

Error Handling in Express.js

1 Analogy: Customer Service

Imagine a customer service desk where something goes wrong. The situation can be handled by solving the problem or forwarding it to a higher level.

2 Error Handling Middleware

In Express, error handling middleware catches errors and either handles them or forwards them to a centralized error handler.

3 Error Forwarding with next()

If you want to skip regular middleware and directly handle an error, you can use next() with an error object.

4 Error-handling Middleware

This middleware is used to catch errors. By convention, it has four arguments, with the first being the error object (err).

Middleware Flow Example (Bringing it Together)

1

Request Logging

Logs every request to the console.

2

Authentication Check

Verifies user authentication.

3

Dashboard Route

Accessible only to authenticated users.

4

Error Handling

Catches and handles errors.

Middleware functions run in order, processing the request before it reaches the route handler. Errors are caught by the error-handling middleware at the end.

What are Static Resources?

Static Resources

Static resources are parts of a website that don’t change depending on the user or time.

Examples include images, stylesheets, or scripts.

Dynamic Content

Dynamic content is the opposite of static resources.

It changes based on user actions or time, such as movie listings or news articles.

Serving Static Resources

Restaurant Analogy

Think of your Express server as a restaurant kitchen. The public folder is like the pantry where you store pre-made food (static files).

Accessing Static Files

When you run the server, you can visit http://localhost:3000 in your browser, and the index.html file will be served automatically.

Introduction

1

Node.js and MongoDB

This session will cover connecting a Node.js application to MongoDB, both locally and in the cloud using MongoDB Atlas.

2

Mongoose ODM

We'll use Mongoose, an Object Document Mapper (ODM) library, to interact with MongoDB in a structured way.

3

Learning Objectives

- Install and set up MongoDB locally
- Connect a Node.js app to a MongoDB database
- Use MongoDB Atlas for production-ready applications

Middleware Flow Example

1. **Request Arrives**

   The client sends a request to the server.

2. **Middleware Execution**

   Middleware functions are executed in the order they are defined.

3. **Route Handler**

   The request reaches the appropriate route handler.

4. **Response Sent**

   The server sends a response back to the client.

Serving Static Resources

Static Files

Express.js can serve static files like HTML, CSS, and JavaScript from a designated directory.

Public Directory

By default, Express looks for static files in a 'public' directory within your project.

Serving Files

Use the `express.static()` middleware to configure the directory for serving static files.

Setting Up Node.js with MongoDB

While running MongoDB locally is useful for development, you often need a scalable, cloud-based solution for production. This is where MongoDB Atlas comes in.

MongoDB Atlas is a cloud-based database service that makes it easy to deploy and manage MongoDB databases. You don’t have to worry about server maintenance, backups, or scaling.

Create an Account

Visit MongoDB Atlas and create a free account.

Create a Cluster

Follow the setup instructions to create a new cluster.

Whitelist Your IP

MongoDB Atlas requires you to whitelist the IP addresses from which you’ll connect.

Create a Database User

Add a database user and password that Node.js will use to connect to MongoDB.

Get the Connection String

After your cluster is ready, you’ll get a connection string.

Setting Up Node.js with MongoDB

Node.js Setup

We've covered how to set up a Node.js app to connect with MongoDB locally using Mongoose.

MongoDB Atlas

We've also explored how to use MongoDB Atlas for a cloud-based database.

Mongoose Models

We've learned how to create and retrieve documents (users) using Mongoose models.