

# **NodeJs + ExpressJs + MongoDB**

## Course Notes

Contains 1 PDF lessons

Generated: 16 February 2026

**KnowledgeGate**

Module

# **Nodejs**

Topic

## **NodeJS**

Subtopic

### **NodeJS**

Lesson

Lesson 1 of 1

#### **NodeJs\_Notes**

# COMPLETE



## Introduction to NodeJS



**CERTIFICATE**

**NOTES**

**Ex-amazon Microsoft**

YouTube [Playlist Link](#)



# NodeJS

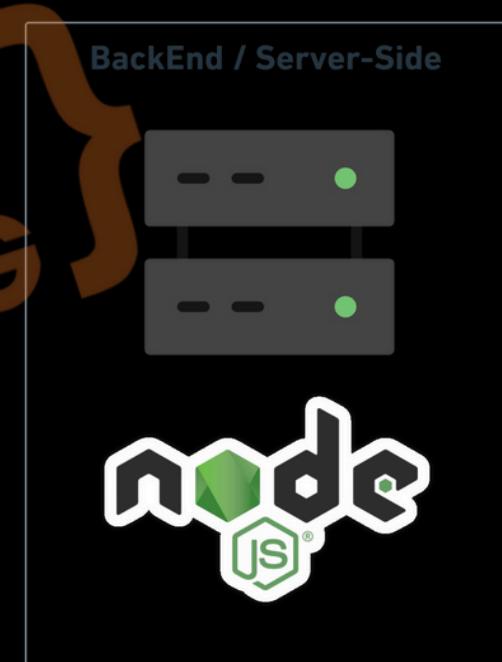
## Complete Course





# Introduction to NodeJS

1. Pre-requisites
  2. What is NodeJS
  3. NodeJs Features
  4. JavaScript on Client
  5. JavaScript on Server
  6. Client Code vs Server Code
  7. Other uses of NodeJs
  8. Server architecture with NodeJs
- complete CODING





JS is required for NodeJS

# COMPLETE JS JAVASCRIPT 14 HOURS



MYNTRA  
PROJECT



CERTIFICATE

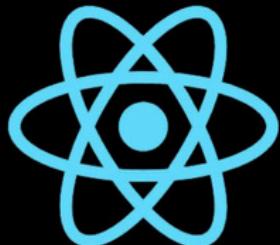
NOTES





React is recommended before NodeJS

# COMPLETE



React



REDUX

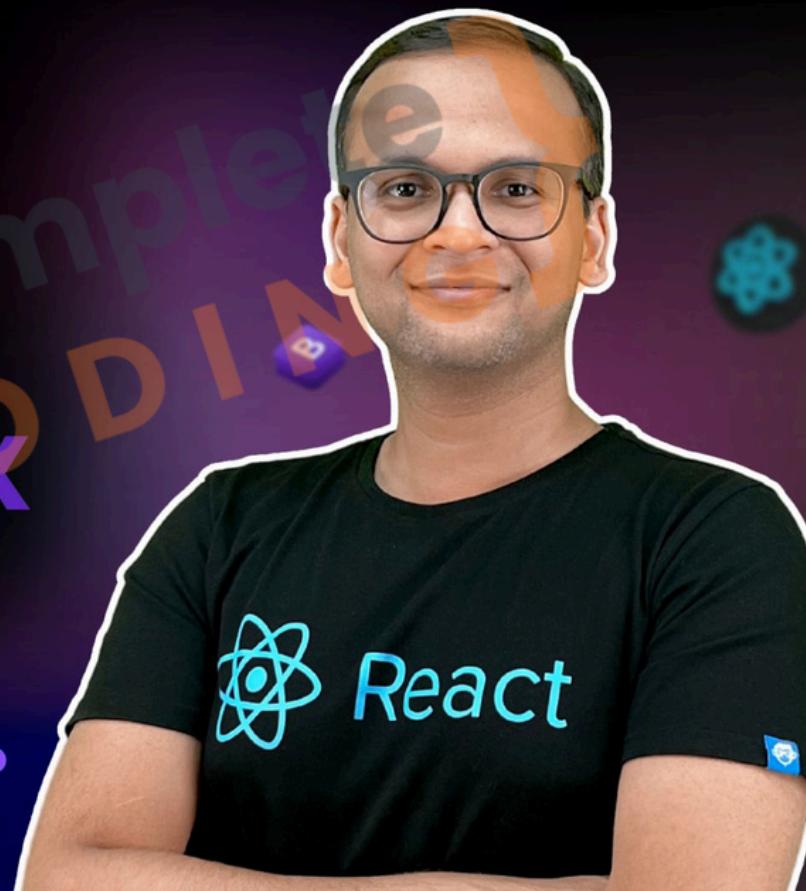
20 HOURS

6 PROJECTS

B using  
Bootstrap

CERTIFICATE

NOTES





## 2.What is NodeJS

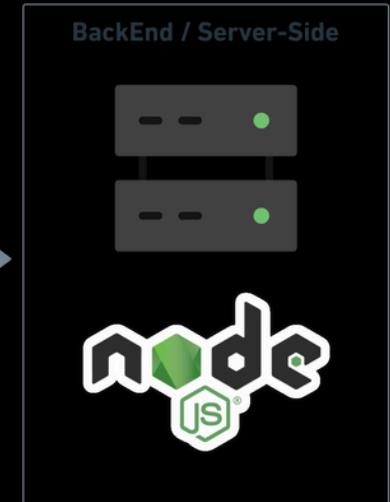


1. **JavaScript Runtime:** Node.js is an open-source, cross-platform runtime environment for executing JavaScript code outside of a browser.
2. **NodeJs is a JavaScript in a different environment** means Running JS on the server or any computer.
3. **Built on Chrome's V8 Engine:** It runs on the V8 engine, which compiles JavaScript directly to native machine code, enhancing performance.
4. V8 is written in C++ for speed.
5. **V8 + Backend Features = NodeJs**



## 2.What is NodeJS

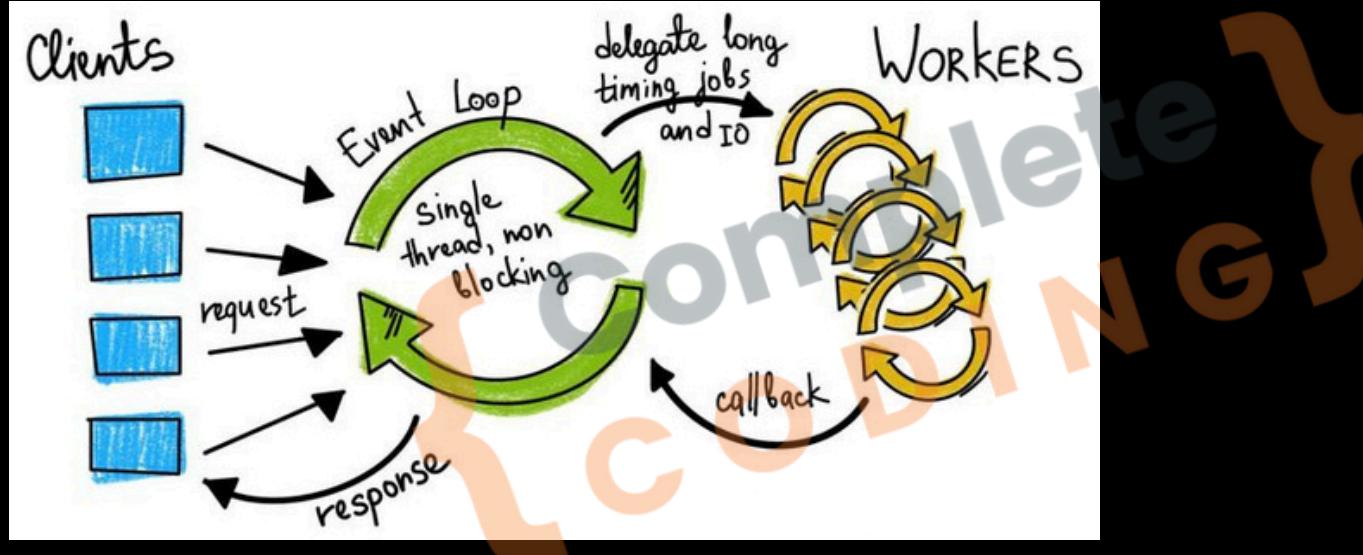
1. **Design:** Features an **event-driven**, **non-blocking I/O** model for efficiency.
2. **Full-Stack JavaScript:** Allows using JavaScript on both **server** and **client** sides.
3. **Scalability:** Ideal for **scalable** network applications due to its architecture.
4. **Versatility:** Suitable for web, real-time chat, and **REST API** servers.





# 3. NodeJs Features

(Added)



1. Non-blocking I/O: Designed to perform non-blocking operations by default, making it suitable for I/O-heavy operations.
2. Networking Support: Supports TCP/UDP sockets, which are crucial for building lower-level network applications that browsers can't handle.



# 3. NodeJs Features

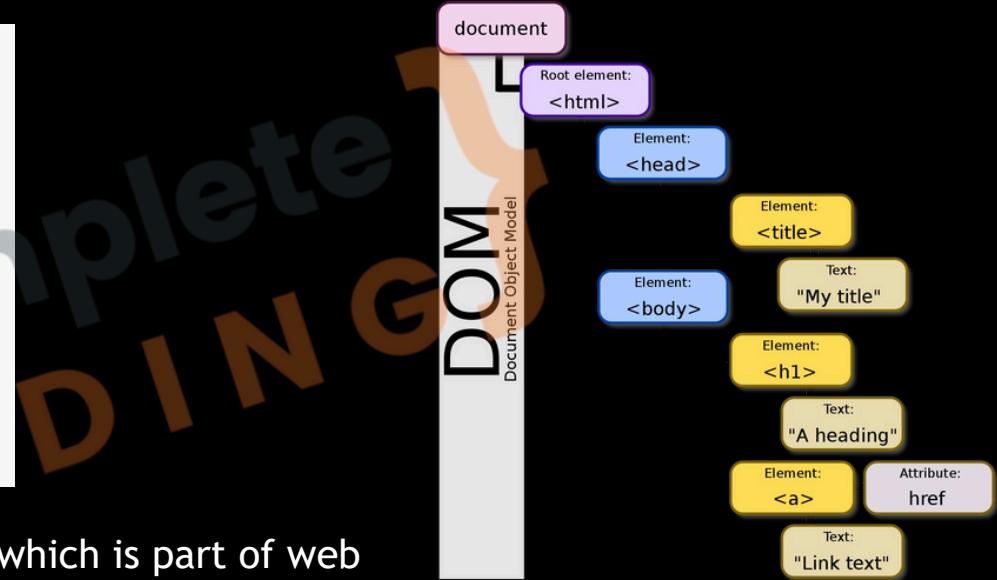
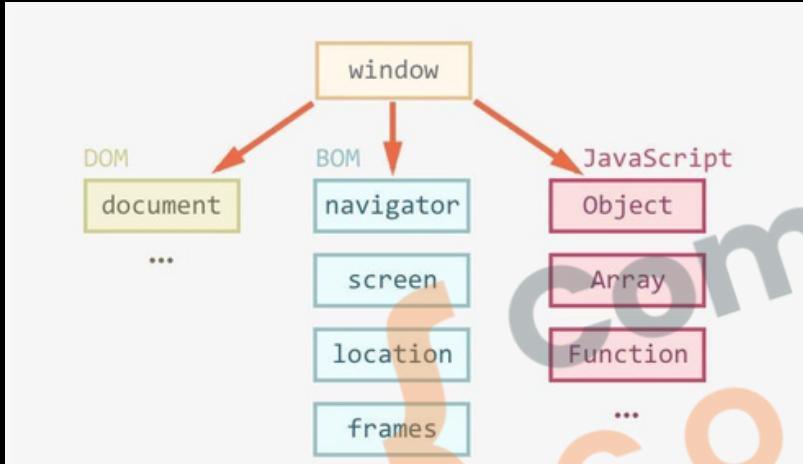
(Added)



1. **File System Access:** Provides APIs to **read and write files** directly, which is **not possible in browser environments** for security reasons.
2. **Server-Side Capabilities:** Node.js enables JavaScript to run on the server, **handling HTTP requests, file operations**, and other server-side functionalities.
3. **Modules:** Organize code into **reusable modules** using `require()`.

# 3. NodeJs Features

(Removed)

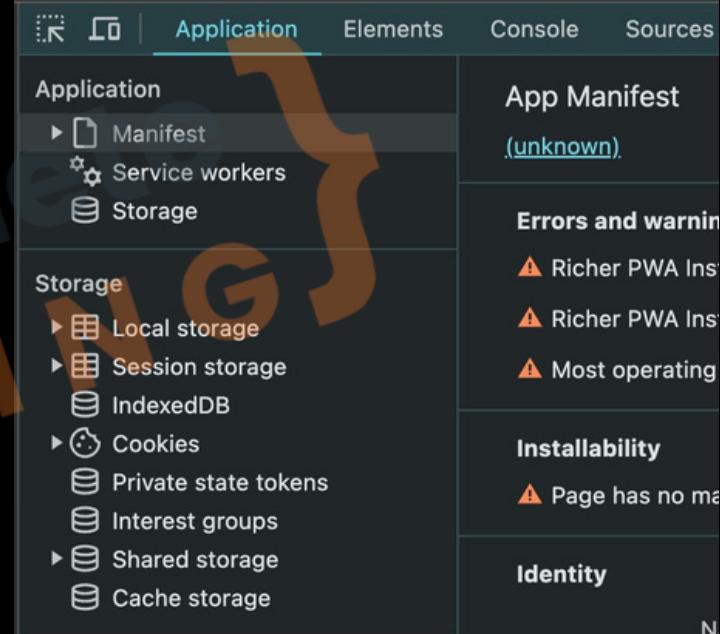
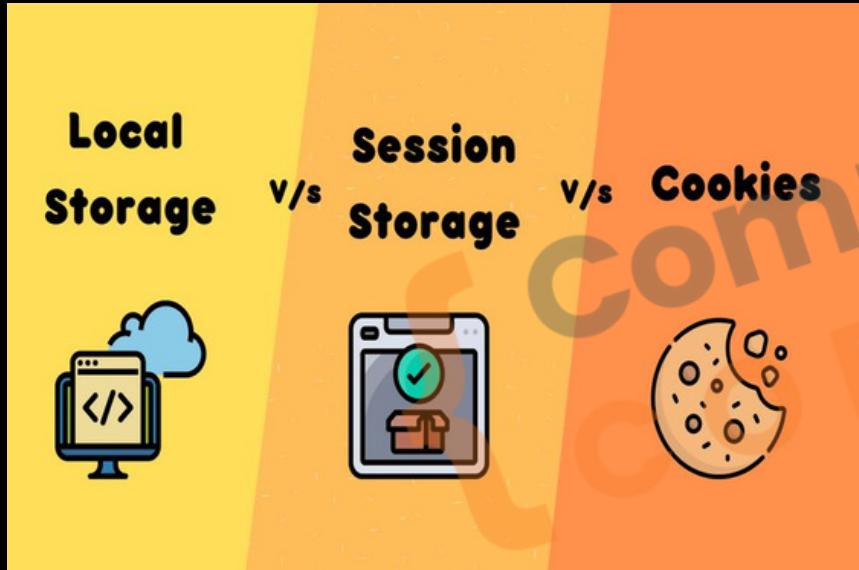


1. **Window Object:** The global **window object**, which is part of web browsers, is absent in **Node.js**.
2. **DOM Manipulation:** **Node.js** does not have a built-in Document Object Model (DOM), as it is **not intended to interact with a webpage's content**.
3. **BOM (Browser Object Model):** No direct interaction with things like **navigator** or **screen** which are part of **BOM** in browsers.



# 3. NodeJs Features

(Removed)



Web-Specific APIs: APIs like `localStorage`, `sessionStorage`, and `fetch` are not available in Node.js.



# 4. JavaScript on Client

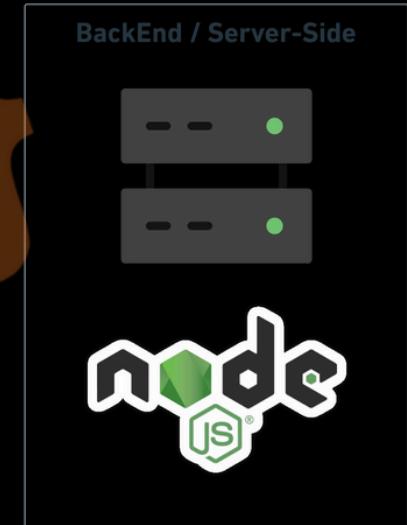


1. **Displays Web Page:** Turns HTML code into what you see on screen.
2. **User Clicks:** Helps you interact with the web page.
3. **Updates Content:** Allows changes to the page using JavaScript.
4. **Loads Files:** Gets HTML, images, etc., from the server.



# 5. JavaScript on Server

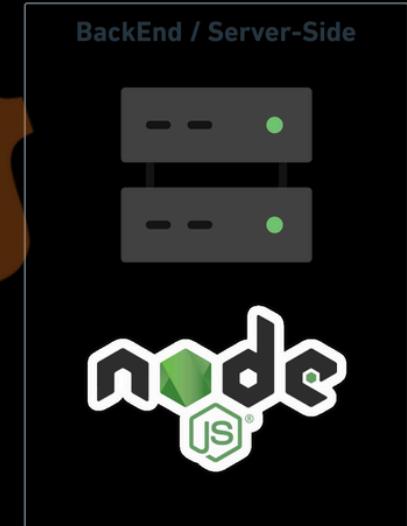
1. **Database Management:** Stores, retrieves, and manages data efficiently through operations like CRUD (Create, Read, Update, Delete).
2. **Authentication:** Verifies user identities to control access to the system, ensuring that users are who they claim to be.
3. **Authorization:** Determines what authenticated users are allowed to do by managing permissions and access controls.
4. **Input Validation:** Checks incoming data for correctness, completeness, and security to prevent malicious data entry and errors.
5. **Session Management:** Tracks user activity across various requests to maintain state and manage user-specific settings.





# 5. JavaScript on Server

6. **API Management:** Provides and handles interfaces for applications to interact, ensuring smooth data exchange and integration.
7. **Error Handling:** Manages and responds to errors effectively to maintain system stability and provide useful error messages.
8. **Security Measures:** Implements protocols to protect data from unauthorized access and attacks, such as SQL injection and cross-site scripting (XSS).
9. **Data Encryption:** Secures sensitive information by encrypting data stored in databases and during transmission.
10. **Logging and Monitoring:** Keeps records of system activity to diagnose issues and monitor system health and security.





# 6. Client Code vs Server Code

1. User/client can't access server code directly.
2. Client must raise requests for particular APIs to
3. access certain features or data.

**Environment Access:** Server-side JavaScript accesses server features like file systems and databases.

4. **Security:** Server-side code can handle sensitive operations securely, while client-side code is exposed and must manage security risks.
5. **Performance:** Heavy computations are better performed on the server to avoid slowing down the client.





# 6. Client Code vs Server Code

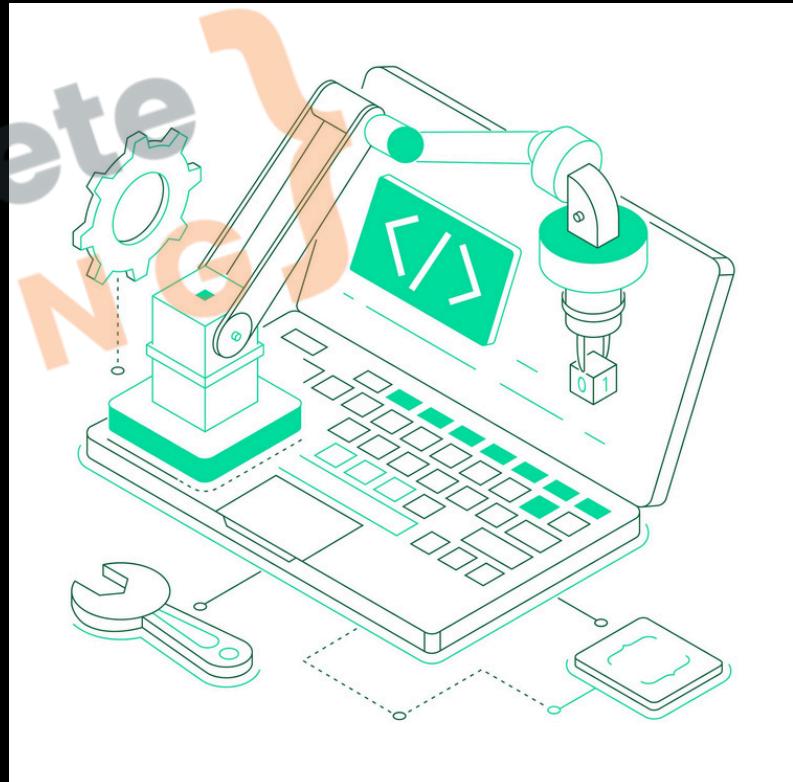
6. **Resource Utilization:** Servers generally offer more powerful processing capabilities than client devices.
7. **Data Handling:** Server-side can directly manage large data sets and database interactions, unlike client-side JavaScript.
8. **Asynchronous Operations:** Server-side JavaScript is optimized for non-blocking I/O to efficiently manage multiple requests.
9. **Session Management:** Servers handle sessions and user states more comprehensively.
10. **Scalability:** Server-side code is designed to scale and handle requests from multiple clients





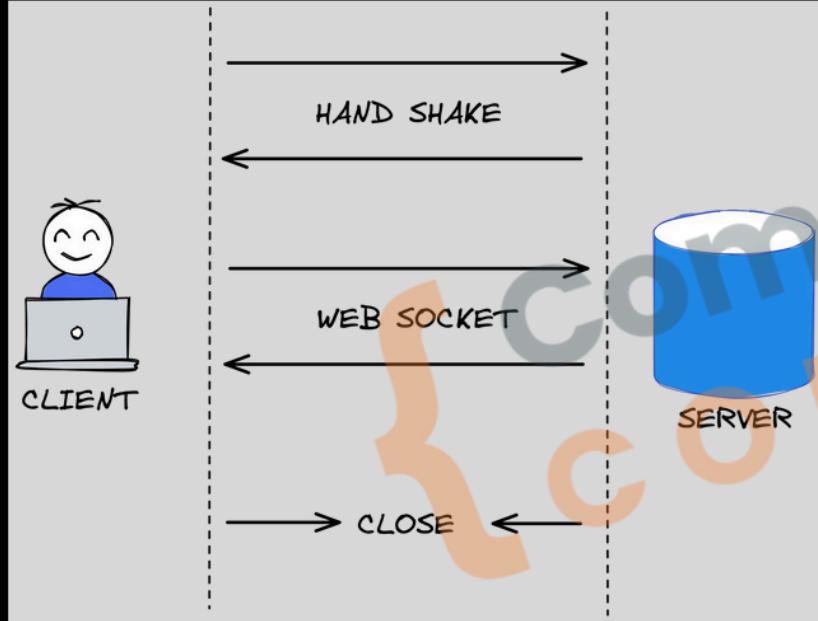
## 7. Other uses of Node.js

1. **Local Utility Scripts:** Automates tasks and processes files locally, like using shell scripts but with JavaScript.
2. **Internet of Things (IoT):** Develops server-side applications for IoT devices, managing communications and data processing.
3. **Scripting for Automation:** Automates repetitive tasks in software development processes, such as testing and deployment.





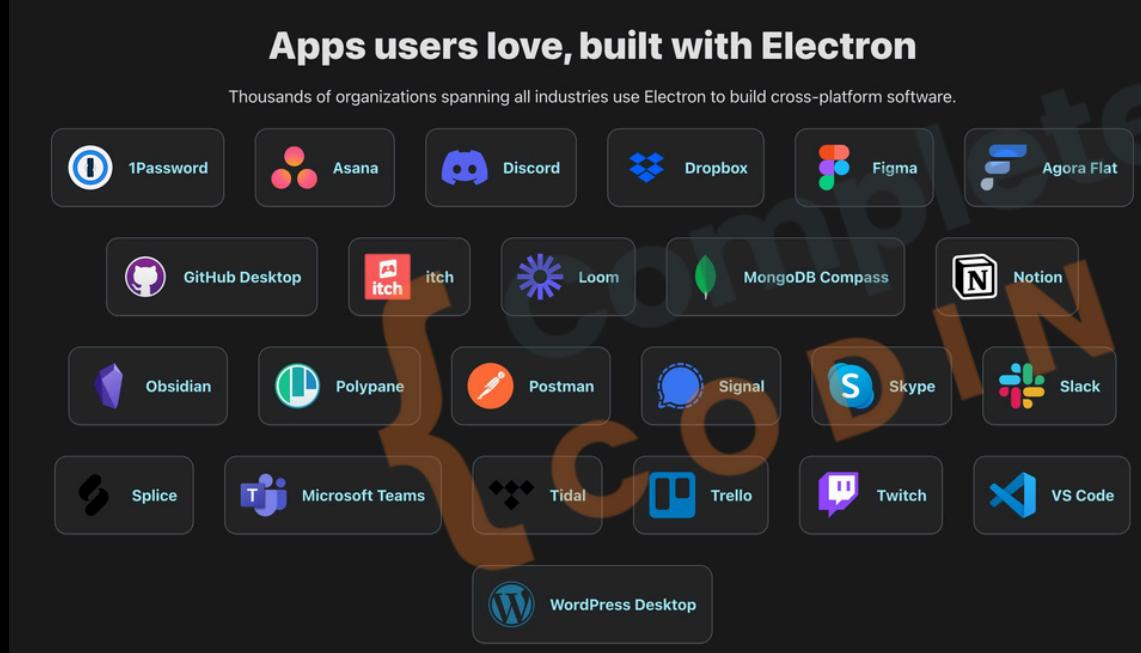
## 7. Other uses of NodeJs



Real-Time Applications: Efficiently manages real-time data applications, such as chat apps and live updates, using WebSockets.



# 7. Other uses of NodeJs



Desktop Applications: Creates cross-platform desktop applications using frameworks like Electron.



## 7. Other uses of NodeJs

Build Tools: Powers build processes for front-end technologies using tools like:

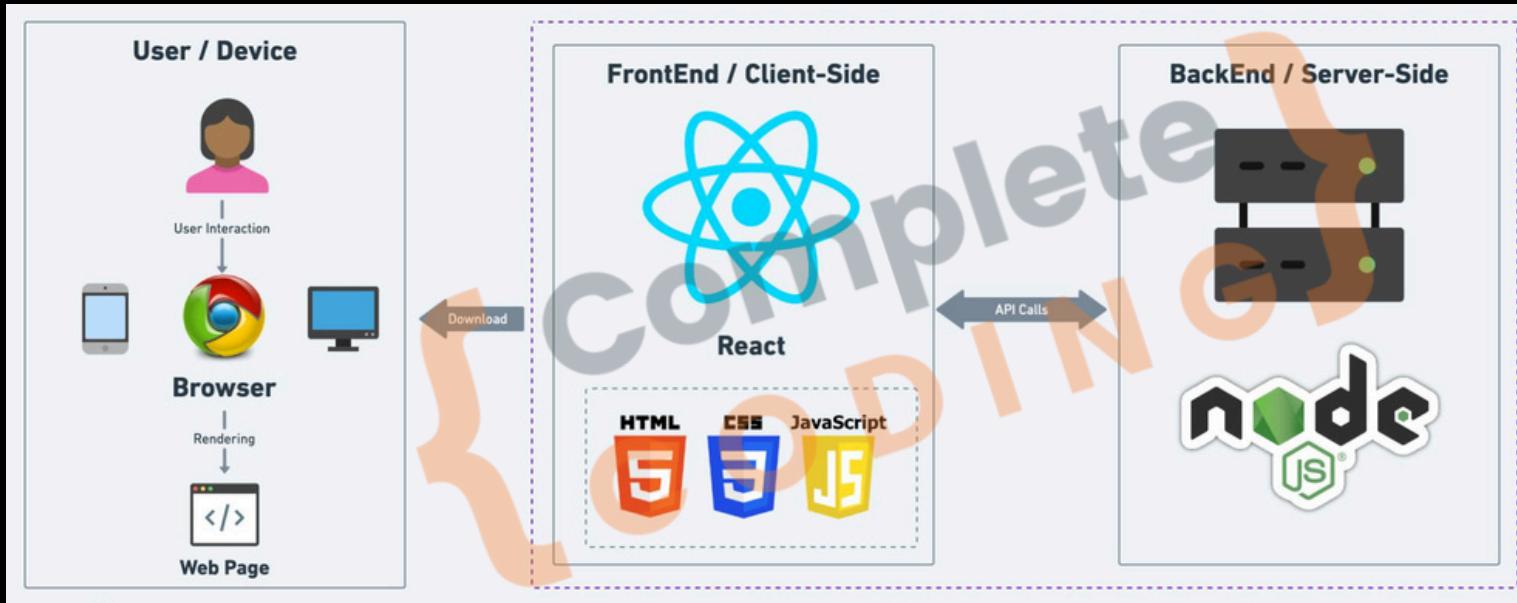
- Webpack
- Grunt
- Gulp
- Browserify
- Brunch
- Yeoman

{ Coding





# 8. Server architecture with NodeJs



Nodejs server will:

- 1.Create server and listen to incoming requests
- 2.Business logic: validation, connect to db, actual processing of data
- 3.Return response HTML, JSON, CSS, JS



# Revision

1. P r e - r e q u i s i t e s
2. What is NodeJS
3. NodeJs Features
4. JavaScript on Client
5. JavaScript on Server
6. Client Code vs Server Code
7. Other uses of NodeJs
8. Server architecture with NodeJs





{ complete  
C O D I N G }



## 2. Installation of NodeJS

1. What is IDE
2. Need of IDE
3. MAC Setup
  - Install latest Node & VsCode
4. Windows Setup
  - Install latest Node & VsCode
5. Linux Setup
  - Install latest Node & VsCode
6. VsCode (Extensions and Settings)
7. Executing first .js file
8. What is REPL
9. Executing Code via REPL





# 2.1 What is IDE

1. IDE stands for **Integrated Development Environment**.
2. Software suite that consolidates basic tools required for **software development**.
3. Central hub for **coding**, finding problems, and testing.
4. Designed to improve **developer efficiency**.





## 2.2 Need of IDE

1. Streamlines development.
2. Increases productivity.
3. Simplifies complex tasks.
4. Offers a unified workspace.
5. IDE Features
  1. Code Autocomplete
  2. Syntax Highlighting
  3. Version Control
  4. Error Checking

```
MainActivity.kt
```

```
@Composable
fun MessageCard(msg: Message) {
    Row(modifier = Modifier.padding(all = 8.dp)) {
        Image(
            painter = painterResource(R.drawable.android_studio_logo),
            contentDescription = "Profile Picture",
            modifier = Modifier
                .size(45.dp)
        )
        Spacer(modifier = Modifier.width(8.dp))
        Column (Modifier
            .background(color = Color.White)) {
            Text(text = msg.author, color = Color.Black)
            Spacer(modifier = Modifier.height(1.dp))
            Text(text = msg.body, color = Color.Black)
        }
    }
}
```



# 2.3 MAC Setup





nodejs.org/en/download

Rare READ KG Tools youtube Resume ChatGPT GitHub whatsapp direct m... KG Coding LeetCode Morgan Stanley Lo... Notes KG Coding

node Learn About Download Blog Docs Certification ↗

# Download Node.js®

Download Node.js the way you want.

Prebuilt Installer Prebuilt Binaries Package Manager Source Code

I want the v22.1.0 (Current) version of Node.js for macOS run

Download Node.js v22.1.0

Node.js includes npm (10.7.0) ↗  
Read the changelog for this version ↗  
Read the blog post for this version ↗  
Learn how to verify signed SHASUMS ↗  
Check out all available Node.js download options ↗  
Learn about Node.js Releases ↗

## 2.3 MAC Setup

### (Install latest Node)

Search Download NodeJS



# 2.3 MAC Setup

## (Install VsCode)



S e a r c h   V S o g l e o d e o n G o

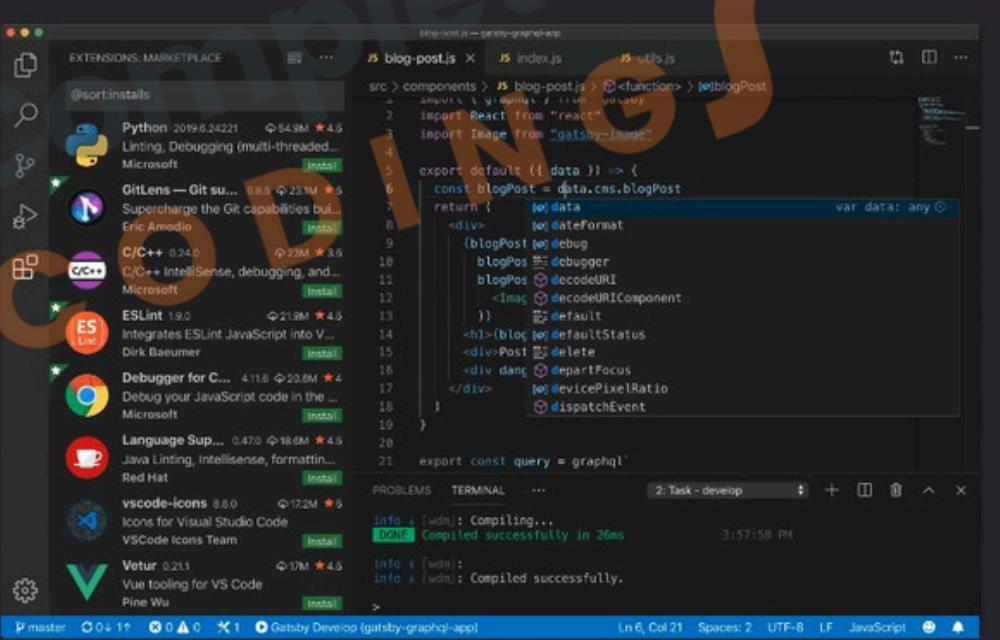
- Third level
  - Fourth level
    - Fifth level

Code editing.  
Redefined.

Free. Built on open source. Runs everywhere.

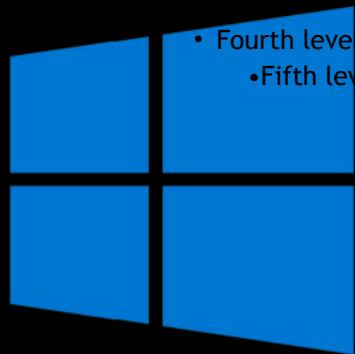


By using VS Code, you agree to its  
license and privacy statement.





# 2.4 Windows Setup



- Fourth level
  - Fifth level

complete  
Windows



nodejs.org/en/download

KG Tools youtube Resume ChatGPT GitHub whatsapp direct m... KG Coding LeetCode Morgan Stanley Lo... Notes KG Coding

Learn About Download Blog Docs ↗ Certification ↗

## Download Node.js®

Download Node.js the way you want.

Prebuilt Installer Prebuilt Binaries Package Manager Source Code

I want the v22.1.0 (Current) version of Node.js for Windows running x64

[Download Node.js v22.1.0](#)

Node.js includes npm (10.7.0) ↗  
Read the changelog for this version ↗  
Read the blog post for this version ↗  
Learn how to verify signed SHASUMS ↗  
Check out all available Node.js download options ↗  
Learn about Node.js Releases ↗



## 2.4 Windows Setup (Install latest Node)

Search Download NodeJS



# 2.4 Windows Setup

## (Install VsCode)

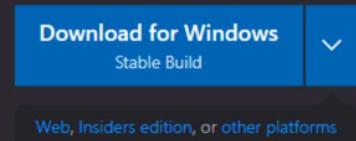


S e a r c h   V S   C o d e o n   G o o g l e

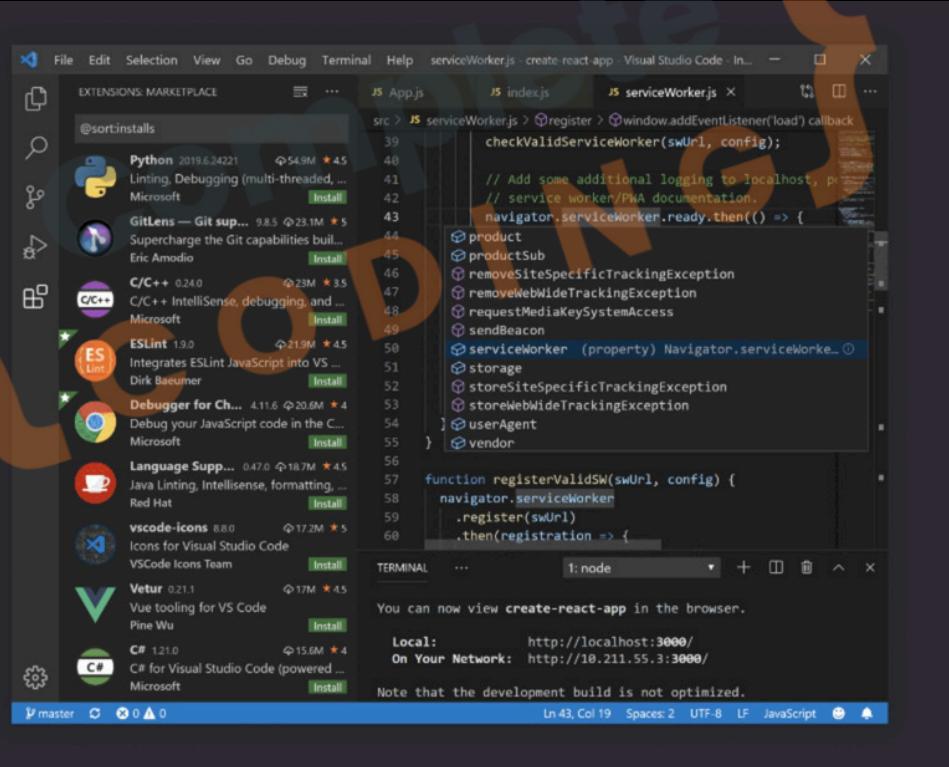
- Second level
- Third level

Code editing.  
Redefined.

Free. Built on open source. Runs everywhere.



By using VS Code, you agree to its  
license and privacy statement.



The image shows the Visual Studio Code interface. On the left, the Extensions Marketplace sidebar lists various extensions: Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, vscode-icons, Vetur, and C#. The main code editor window displays a portion of a JavaScript file named serviceWorker.js, which contains code related to service workers and registration. The bottom right corner of the code editor shows the status bar with "In 43, Col 19" and "JavaScript". The bottom left corner shows the terminal with the command "node" entered.



# 2.5 Linux Setup





## 2.5 Linux Setup (Install latest Node)

Search Download NodeJS

Node.js — Download Nod X nodejs for linux - Google X Node.js — Download Nod X +

https://nodejs.org/en/download/package-manager

Node.js v22 is now available! ↗

Learn About Download Blog Docs ↗ Certification ↗

### Download Node.js®

Download Node.js the way you want.

Prebuilt Installer Prebuilt Binaries **Package Manager** Source Code

Install Node.js v22.1.0 (Current) on Linux using NVM

```
1 # installs NVM (Node Version Manager)
2 curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | b
3
4 # download and install Node.js
5 nvm install 22
6
7 # verifies the right Node.js version is in the environment
8 node -v # should print 'v22.1.0'
9
10 # verifies the right NPM version is in the environment
11 npm -v # should print '10.7.0'
```

Bash

Copy

Please ensure you have the right package manager installed before running a script.  
Package managers and their installation scripts are not maintained by the Node.js project.



# 2.5 Linux Setup

## (Install VsCode)



S e a r c h   V S   C o d e o n   G o o g l e

- Second level

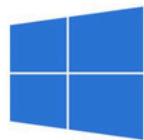
- Third level

- Fourth level

- Fifth level

Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.



↓ Windows

Windows 10, 11



↓ .deb

Debian, Ubuntu



↓ Mac

macOS 10.15+

↓ .rpm

Red Hat, Fedora, SUSE

User Installer	x64	Arm64
System Installer	x64	Arm64
.zip	x64	Arm64
CLI	x64	Arm64

.deb	x64	Arm32	Arm64
.rpm	x64	Arm32	Arm64
.tar.gz	x64	Arm32	Arm64
Snap	Snap Store		

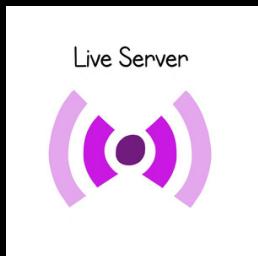
.zip	Intel chip	Apple silicon	Universal
CLI	Intel chip	Apple silicon	



# 2.6 VsCode

(Extensions and Settings)

1. Prettier (Format on Save)
2. Line Wrap
3. Tab Size from 4 to 2





# 2.7 Executing first .js file

```
1  const fs = require('fs');
2
3  // Define two variables
4  let a = 10;
5  let b = 5;
6
7  // Basic arithmetic operations
8  let sum = a + b;
9  let product = a * b;
10
11 // Prepare data to write
12 let data = `Sum: ${sum}\nProduct: ${product}`;
13 console.log(data);
14
15 // Write data to a local file
16 fs.writeFile('output.txt', data, (err) => {
17   if (err) throw err;
18   console.log('Data written to file');
19 })
```

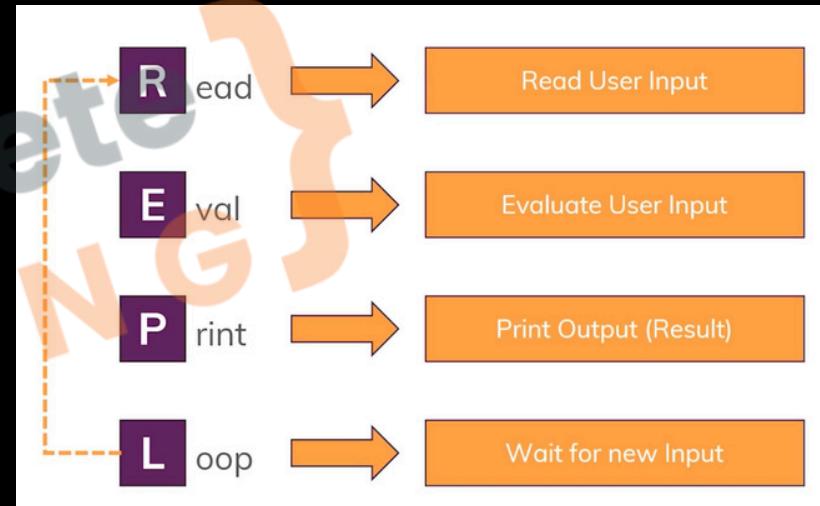
```
prashantjain@Mac-mini Desktop % node test.js
Sum: 15
Product: 50
Data written to file
```

1. Streamlines Node Command: Use `node filename.js` to execute a JavaScript file in the Node.js environment.
2. Require Syntax: Use `require('module')` to include built-in or external modules, or other JavaScript files in your code.
3. Modular Code: `require` helps organize code into reusable modules, separating concerns and improving maintainability.
4. Caching: Modules loaded with `require` are cached, meaning the file is executed only once even if included multiple times.



## 2.8 What is REPL

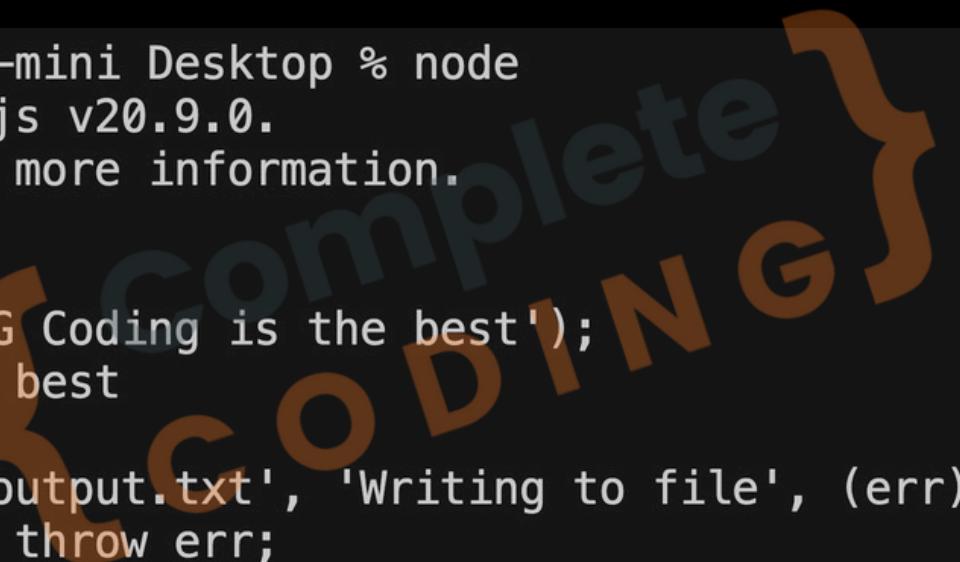
1. Streamlines Interactive Shell: Executes JavaScript code interactively.
2. Quick Testing: Ideal for testing and debugging code snippets on the fly.
3. Built-in Help: Offers help commands via .help.
4. Session Management: Supports saving (.save) and loading (.load) code sessions.
5. Node.js API Access: Provides direct access to Node.js APIs for experimentation.
6. Customizable: Allows customization of prompt and behaviour settings.





## 2.9 Executing Code via REPL

```
prashantjain@Mac-mini Desktop % node
Welcome to Node.js v20.9.0.
Type ".help" for more information.
> 5 + 6
11
> console.log('KG Coding is the best');
KG Coding is the best
undefined
> fs.writeFileSync('output.txt', 'Writing to file', (err) => {
...     if (err) throw err;
...     console.log('Data written to file');
... });
undefined
> Data written to file
```

A large, semi-transparent watermark is overlaid on the code block. It contains the word "Complete" in a light blue font and "CODING" in a larger, bold brown font, with a curly brace graphic connecting them.



# Revision

1. What is IDE
  2. Need of IDE
  3. MAC Setup
    - Install latest Node & VsCode
  4. Windows Setup
    - Install latest Node & VsCode
  5. Linux Setup
    - Install latest Node & VsCode
  6. VsCode (Extensions and Settings)
  7. Executing first .js file
  8. What is REPL
- Executing Code via REPL





{ complete  
C O D I N G }



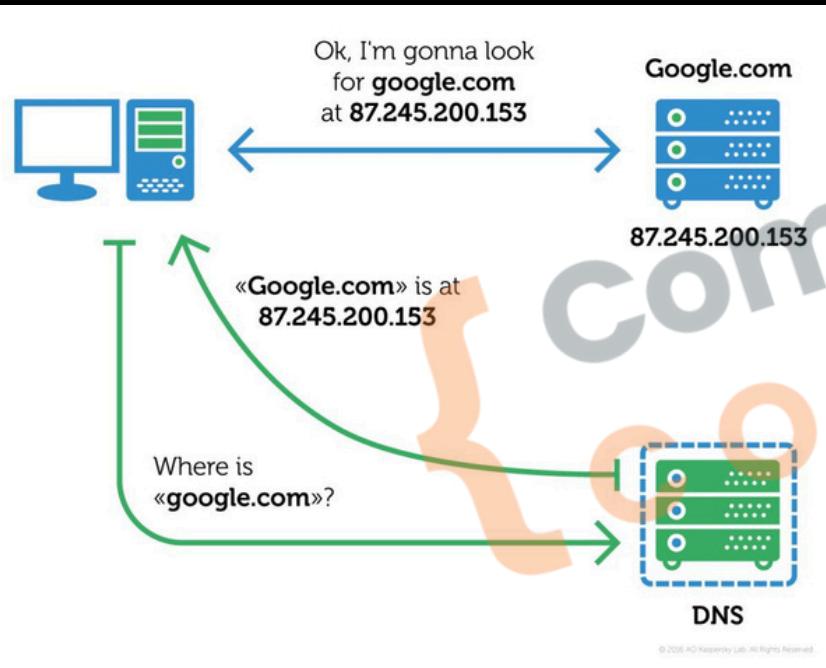
# 3. First Node Server

1. How DNS Works?
2. How Web Works?
3. What are Protocols?
4. Node Core Modules
5. Require Keyword
6. Creating first Node Server





# 3.1 How DNS Works?

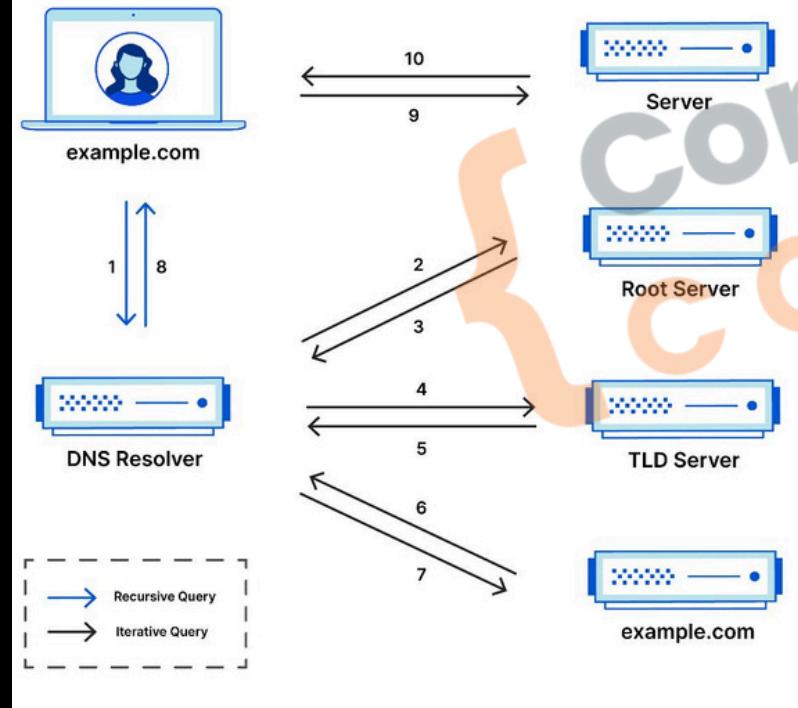


1. **Domain Name Entry:** User types a domain (e.g., [www.example.com](#)) into the browser.
2. **DNS Query:** The browser sends a **DNS query** to resolve the domain into an IP address.
3. **DNS Server:** Provides the correct IP address for the domain.
4. **Browser Connects:** The browser uses the IP to connect to the web server and loads the website.



# 3.1 How DNS Actually Works?

## Complete DNS Lookup and Webpage Query

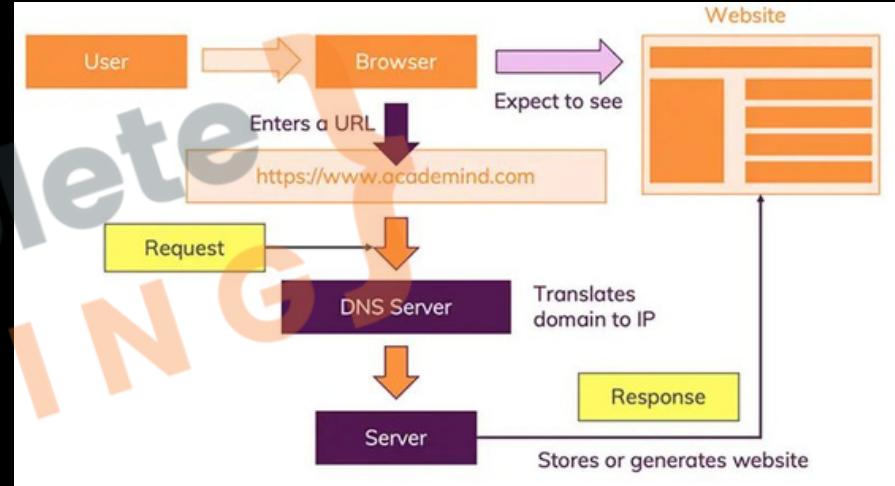


1. **Root DNS:** Acts as the starting point for DNS resolution. It directs queries to the correct TLD server (e.g., .com, .org).
2. **TLD (Top-Level Domain) DNS:** Handles queries for specific top-level domains (e.g., .com, .net) and directs them to the authoritative DNS server (e.g., Verisign for .com, PIR for .org)
3. **Authoritative DNS:** Contains the actual IP address of the domain and answers DNS queries with this information. (e.g., Cloudflare, Google DNS).



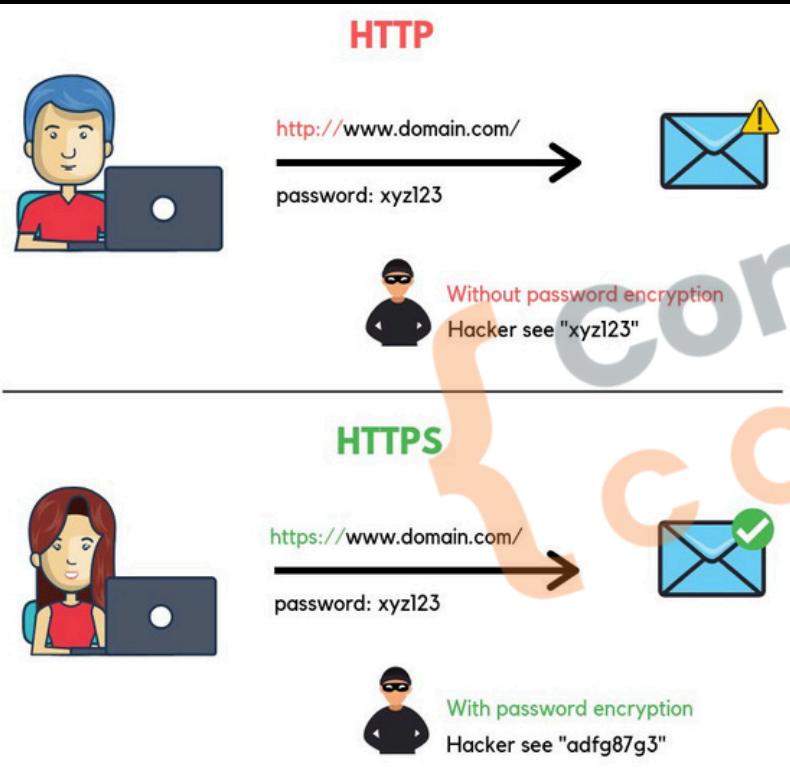
## 3.2 How Web Works?

1. Client Request Initiation: The client (browser) initiates a network call by entering a URL.
2. DNS Resolution: The browser contacts a DNS server to get the IP address of the domain.
3. TCP Connection: The browser establishes a TCP connection with the server's IP address.
4. HTTP Request: The browser sends an HTTP request to the server.
5. Server Processing: The server processes the request and
6. prepares a response.  
HTTP Response: The server sends an HTTP response back to the client.
7. Network Transmission: The response travels back to the client over the network.
8. Client Receives Response: The browser receives and interprets the response.
9. Rendering: The browser renders the content of the response and displays it to the user.





### 3.3 What are Protocols?



Http (HyperText Transfer Protocol):

- Facilitates communication between a web browser and a server to transfer web pages.
- Sends data in plain text (no encryption).
- Used for basic website browsing without security.

HTTPS (HyperText Transfer Protocol Secure):

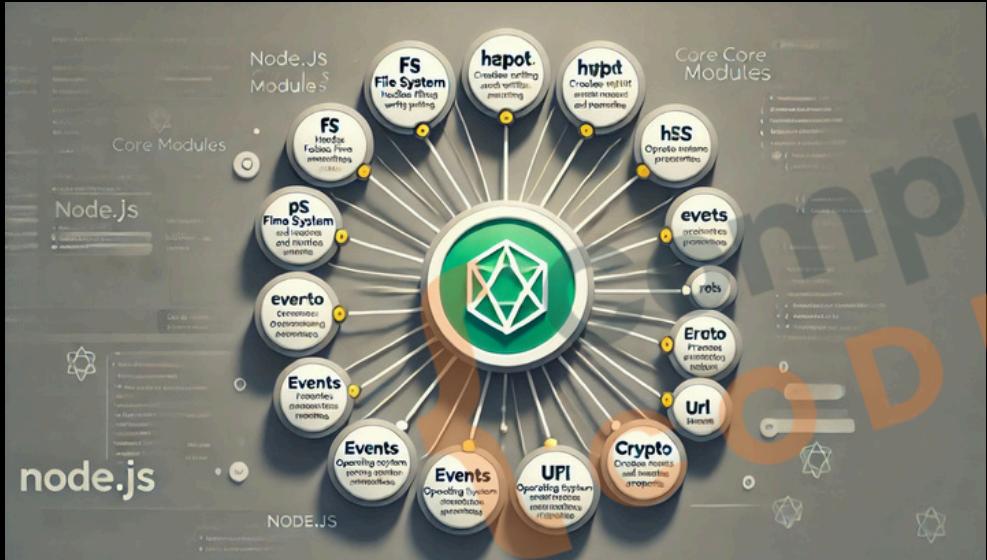
- Secure version of HTTP, encrypts data for secure communication.
- Uses SSL/TLS to encrypt data.
- Used in online banking, e-commerce.

TCP (Transmission Control Protocol):

- Ensures reliable, ordered, and error-checked data delivery over the internet.
- Establishes a connection before data is transferred.



# 3.4 Node Core Modules



1. Built-in: Core modules are included with Node.js installation.
2. No Installation Needed: Directly available for use without npm install.
3. Performance: Highly optimized for performance.



## 3.4 Node Core Modules

1. **fs (File System)**: Handles file operations like reading and writing files.
2. **http**: Creates HTTP servers and makes HTTP requests.
3. **https**: Launch a SSL Server.
4. **path**: Provides utilities for handling and transforming file
5. **paths.os**: Provides operating system-related utility methods and properties.
6. **events**: Handles events and event-driven programming.
7. **crypto**: Provides cryptographic functionalities like hashing and encryption.
8. **url**: Parses and formats URL strings.



# 3.5 Require Keyword

- 1.Purpose: Imports modules in Node.js.
- 2.Caching: Modules are cached after the first require call.
- 3..js is added automatically and is not needed to at the end of module name.
- 4.Path Resolution: Node.js searches for modules in core, node\_modules, and file paths.

Syntax:

```
const moduleName = require('module');
```

```
// Load the built-in http module  
const http = ...;
```

```
// Load the third party express module  
const express = require('express');
```

```
// Load the custom myModule module  
const myModule = require('./myModule');
```

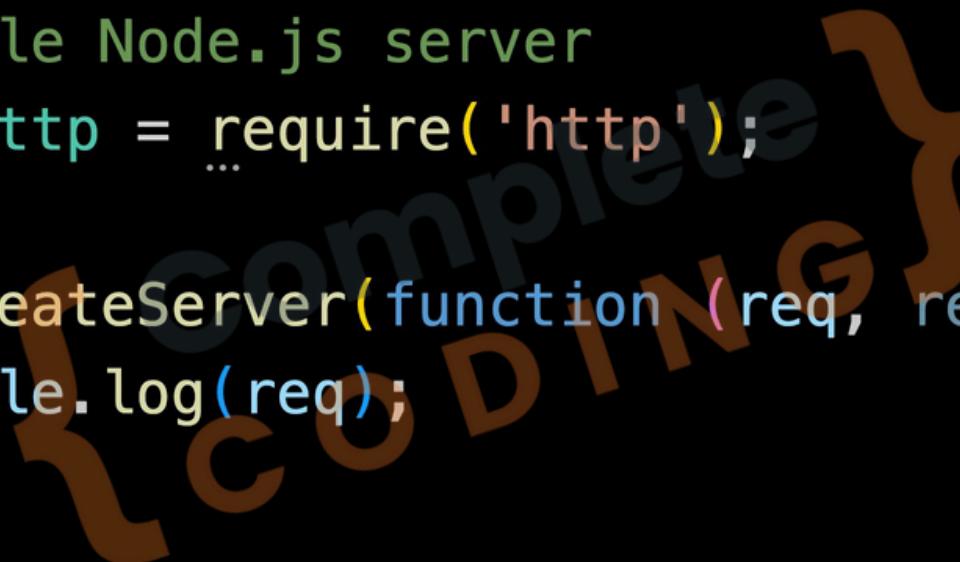


# 3.6 Creating first Node Server

```
1 // Simple Node.js server
2 const http = require('http');
3
4 function requestListener(req, res) {
5   console.log(req);
6 }
7
8 http.createServer(requestListener);
```

# 3.6 Creating first Node Server

```
1 // Simple Node.js server
2 const http = require('http');
3
4 http.createServer(function (req, res) {
5   console.log(req);
6 });


```



# 3.6 Creating first Node Server

```
1 // Simple Node.js server
2 const http = require('http');
3 ...
4 http.createServer((req, res) => {
5   console.log(req);
6 });

```

Run the code with:

***node app.js***



# 3.6 Creating first Node Server

```
1 // Simple Node.js server
2 const http = require('http');
3
4 const server = http.createServer((req, res) => {
5   console.log(req);
6 });
7
8 server.listen(3000);
```

```
insecureHTTPParser: undefined,
requestTimeout: 300000,
headersTimeout: 60000,
keepAliveTimeout: 5000,
connectionsCheckingInterval: 30000,
requireHostHeader: true,
joinDuplicateHeaders: undefined,
rejectNonStandardBodyWrites: false,
_events: [Object: null prototype],
_eventsCount: 3,
_maxListeners: undefined,
_connections: 2,
_handle: [TCP],
_usingWorkers: false,
_workers: [],
_unref: false,
_listeningId: 2,
allowHalfOpen: true,
pauseOnConnect: false,
noDelay: true,
keepAlive: false,
keepAliveInitialDelay: 0,
highWaterMark: 65536,
httpAllowHalfOpen: false,
timeout: 0,
maxHeadersCount: null,
```

# 3.6 Creating first Node Server

```
1 // Simple NodeJS server
2 const http = require('http');
3
4 const server = http.createServer((req, res) => {
5   console.log(req);
6 });
7
8 const PORT = 3000;
9 server.listen(PORT, () => {
10   console.log(`Server running at http://localhost:${PORT}/`);
11});
```



# Revision

1. How Does It Work?
2. How Does It Work?
3. What are Protocols?
4. Node Core Modules
5. Require Keyword
6. Creating first Node Server





{ complete  
C O D I N G }

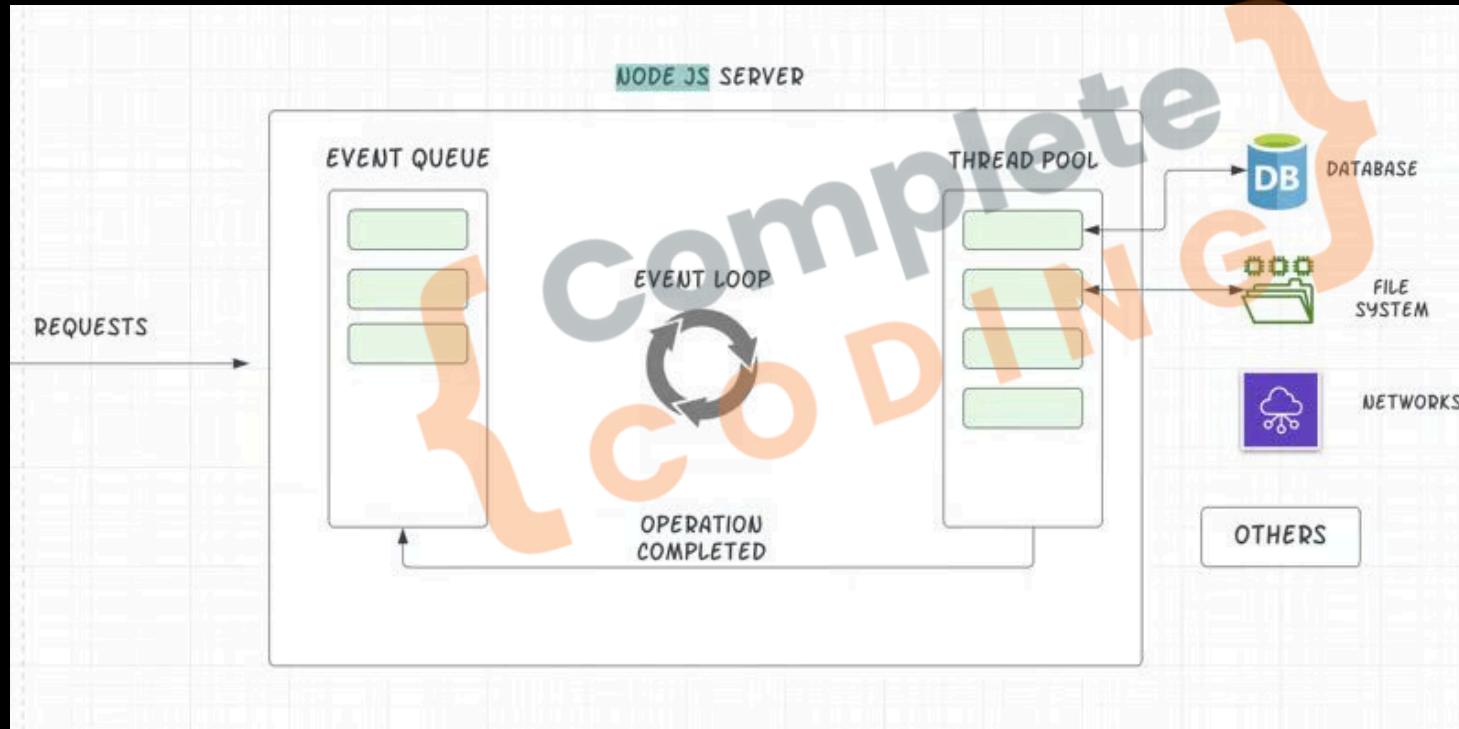


# 4. Request & Response

1. Node Lifecycle & Event Loop
2. How to exit Event Loop
3. Understand Request Object
4. Sending Response
5. Routing Requests
6. Taking User Input
7. Redirecting Requests



# node 4.1 Node Lifecycle & Event Loop





## 4.2 How to exit Event Loop

```
1 // Simple Node.js server
2 const http = require('http');
3
4 const server = http.createServer((req, res) => {
5   console.log(req);
6   process.exit(); // Stops event loop
7 });
8
9 const PORT = 3000;
10 server.listen(PORT, () => {
11   console.log(`Server running at http://localhost:${PORT}/`);
12 });
```

# 4.3 Understand Request Object

```
[Symbol(kHeaders)]: {  
  host: 'localhost:3000',  
  connection: 'keep-alive',  
  'cache-control': 'max-age=0',  
  'sec-ch-ua': '"Chromium";v="128", "Not;A=Brand";v="24", "Google Chrome";v="128"',  
  'sec-ch-ua-mobile': '?0',  
  'sec-ch-ua-platform': '"macOS"',  
  'upgrade-insecure-requests': '1',  
  'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.6482.121 Safari/537.36',  
  accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.7',  
  'sec-fetch-site': 'none',  
  'sec-fetch-mode': 'navigate',  
  'sec-fetch-user': '?1',  
  'sec-fetch-dest': 'document',  
  'accept-encoding': 'gzip, deflate, br, zstd',  
  'accept-language': 'en-US,en-IN;q=0.9,en;q=0.8,hi-IN;q=0.7,hi;q=0.6',  
  cookie: 'token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImM3M2Y0MzNjLTFlYzYtNDIqvUSRMLqbP1uy5bTjnJEQHXc1c'  
,  
  [Symbol(kHeadersCount)]: 32,  
  [Symbol(kTrailers)]: null,  
  [Symbol(kTrailersCount)]: 0  
}
```



# 4.3 Understand Request Object

```
// Simple NodeJS server
const http = require('http');

const server = http.createServer((req, res) => {
  console.log(req.url, req.method, req.headers);
});

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

Use the browser to access:

<http://localhost:3000/>

<http://localhost:3000/products>



# 4.4 Sending Response

```
1 // Simple NodeJS server
2 const http = require('http');
3
4 const server = http.createServer((req, res) => {
5   //res.setHeader('Content-Type', 'json');
6   res.setHeader('Content-Type', 'text/html');
7   res.write('<html>');
8   res.write('<head><title>Complete Coding</title></head>');
9   res.write('<body><h1>Like / Share / Subscribe</h1></body>');
10  res.write('</html>');
11  res.end();
12 });
13
14 const PORT = 3000;
15 server.listen(PORT, () => {
16   console.log(`Server running at http://localhost:${PORT}/`);
17 })
```



# 4.4 Sending Response

A screenshot of a browser's developer tools Network tab. The URL in the address bar is `localhost:3000`. The Network tab is selected, showing a list of requests. The first request, for `localhost`, has a duration of `10 ms`. The response body is displayed as an HTML document:

```
1 <html>
-   <head>
-     |   <title>Complete Coding</title>
-   </head>
-   <body>
-     |   <h1>Like / Share / Subscribe</h1>
-   </body>
- </html>
```

The `localhost` entry in the list also shows a preview of the HTML content.

complete  
Coding



# 4.5 Routing Requests

```
const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>Complete Coding</title></head>');
  if (req.url === '/') {
    res.write('<h1>Welcome to Home page</h1>');
    res.end();
  } else if (req.url.toLowerCase() === '/products') {
    res.write('<h1>Products</h1>');
    res.end();
  }
  res.write('<body><h1>Like / Share / Subscribe</h1></body>');
  res.write('</html>');
  res.end();
});
```



# 4.5 Routing Requests

```
⑥ prashantjain@Prashants-Mac-mini node % node app.js
```

```
Server running at http://localhost:3000/
```

```
node:_http_outgoing:699
```

```
    throw new ERR_HTTP_HEADERS_SENT('set');
```

```
^
```

```
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
```

```
at ServerResponse.setHeader (node:_http_outgoing:699:11)
```

```
at Server.<anonymous> (/Users/prashantjain/workspace/Test Project/node/app.js:14:7)
```

```
at Server.emit (node:events:520:28)
```

```
at parserOnIncoming (node:_http_server:1146:12)
```

```
at HTTPParser.parserOnHeadersComplete (node:_http_common:118:17) {
```

```
  code: 'ERR_HTTP_HEADERS_SENT'
```

```
}
```

```
Node.js v22.5.1
```



# 4.5 Routing Requests

```
const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>Complete Coding</title></head>');
  if (req.url === '/') {
    res.write('<h1>Welcome to Home page</h1>');
    return res.end();
  } else if (req.url.toLowerCase() === '/products') {
    res.write('<h1>Products</h1>');
    return res.end();
  }
  res.write('<body><h1>Like / Share / Subscribe</h1></body>');
  res.write('</html>');
  return res.end();
});
```



# 4.6 Taking User Input

```
if (req.url === '/') {  
    res.write('<h1>Welcome to Home page</h1>');  
    res.write('<form action="/submit-details" method="POST">');  
    res.write('<input type="text" id="name" name="name" placeholder="Enter your  
name"><br><br>');  
    res.write('<label for="gender">Gender:</label>');  
    res.write('<input type="radio" id="male" name="gender" value="male">');  
    res.write('<label for="male">Male</label>');  
    res.write('<input type="radio" id="female" name="gender" value="female">');  
    res.write('<label for="female">Female</label><br><br>');  
    res.write('<button type="submit">Submit</button>');  
    res.write('</form>');  
    return res.end();  
}
```



## 4.7 Redirecting Requests

```
 } else if (req.method == 'POST' &&
 | | | | | req.url.toLowerCase() === '/submit-details') {
fs.writeFileSync('user-details.txt', 'Prashant Jain');
res.statusCode = 302;
res.setHeader('Location', '/');
return res.end();
}
```

node > ≡ user-details.txt

1 Prashant Jain



# Practise Set

Create a page that shows a navigation bar of Myntra with the following links:

- A. Home B.
- Men C.
- Women D.
- Kids E. Cart

Complete CODING



Clicking on each link **page** should **navigate** to that **page** and a **welcome to section text** is shown there.



# Revision

1. Node.js & Event Loop
2. How to Fix Event Loop
3. Understand Request Object
4. Sending Response
5. Routing Requests
6. Taking User Input
7. Redirecting Requests





{ complete  
C O D I N G }



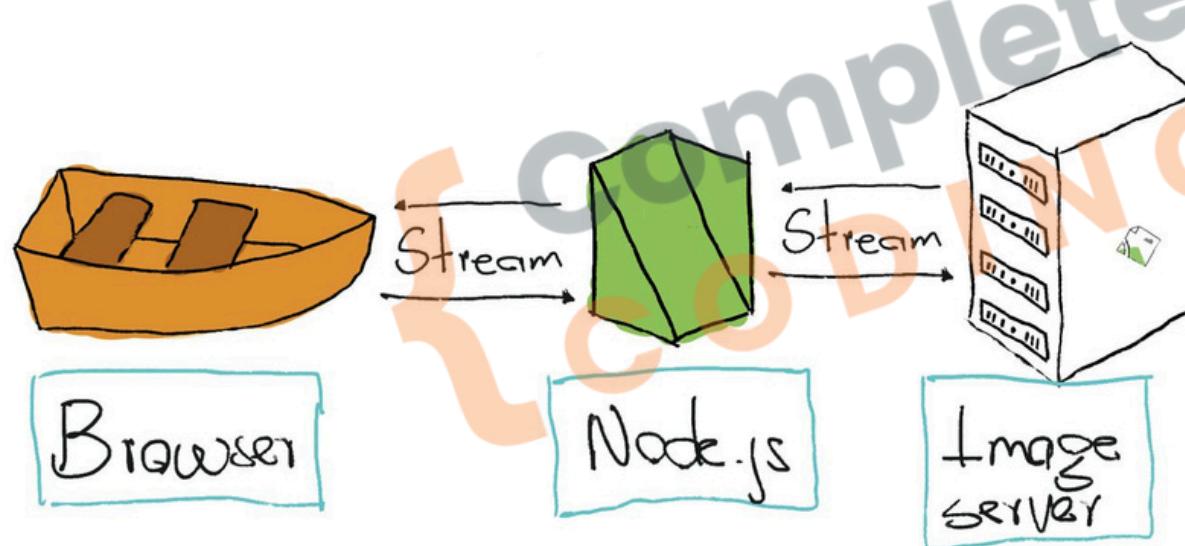
# 5. Parsing Request

1. Streams
2. Chunks
3. Buffers
4. Reading Chunk
5. Buffering Chunks
6. Parsing Request
7. Using Modules



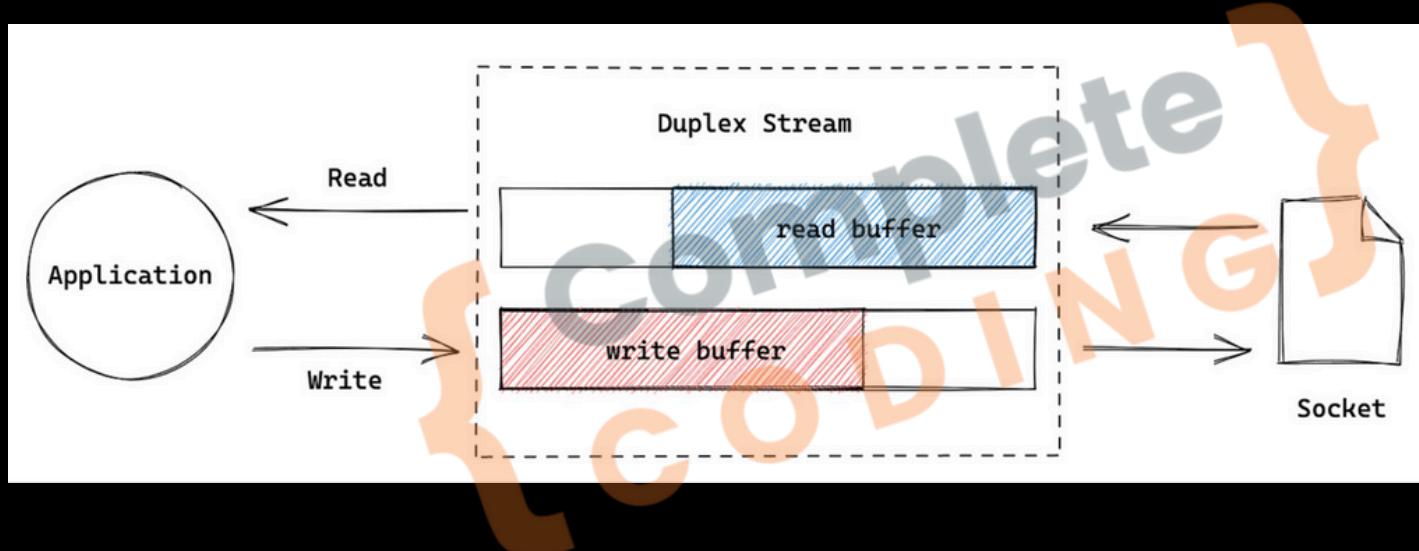


# 5.1 Streams



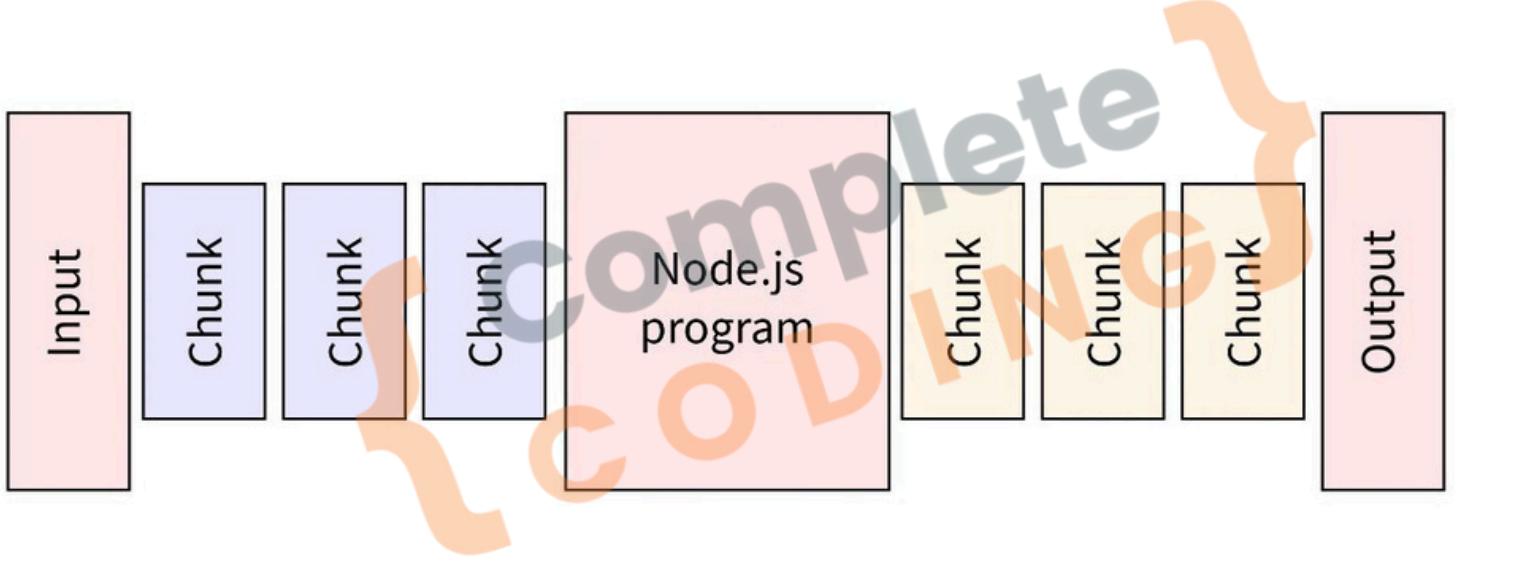


# 5.1 Streams



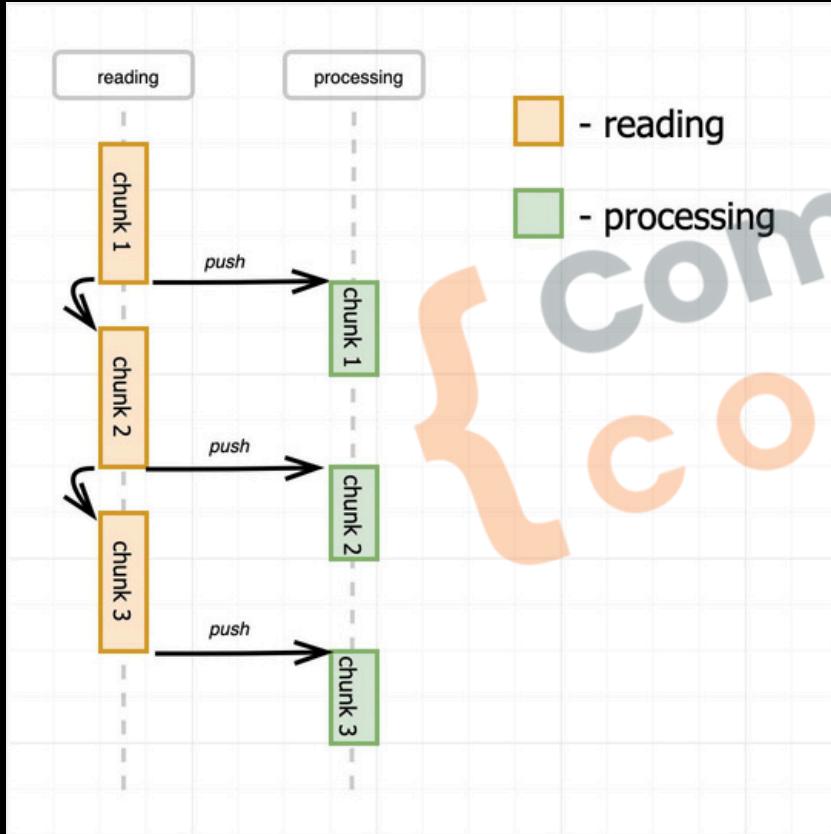


## 5.2 Chunks





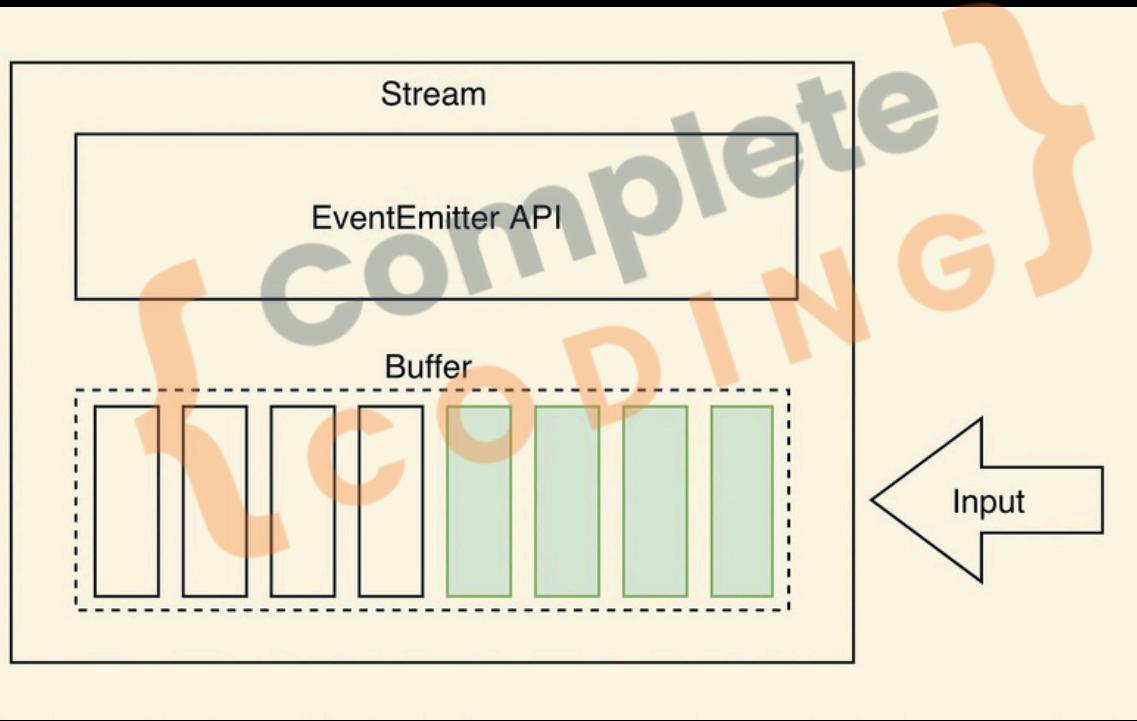
# 5.2 Chunks



complete  
C O M P L E T E  
D I N G



# 5.3 Buffers





# 5.4 Reading Chunk

```
    } else if (
      req.method == "POST" &&
      req.url.toLowerCase() === "/submit-details"
    ) {
      req.on("data", (chunk) => {
        console.log(chunk);
      });
      fs.writeFileSync("user-details.txt", "Prashant Jain");
      res.setHeader("Location", "/");
      res.statusCode = 302;
      return res.end();
    }
  }
```

```
prashantjain@Prashants-Mac-mini node % node app.js
Server running at http://localhost:3000/
<Buffer 6e 61 6d 65 3d 50 72 61 73 68 61 6e 74 26 67 65 6e 64 65 72 3d 6d 61 6c 65>
```



# 5.5 Buffering Chunks

```
const body = [];
req.on("data", (chunk) => {
  console.log(chunk);
  body.push(chunk);
});
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
});
fs.writeFileSync("user-details.txt", "Prashant Jain");
```

A large, semi-transparent watermark reading "Complete CODING" is overlaid on the code block.

```
.prashantjain@Prashants-Mac-mini % node app.js
Server running at http://localhost:3000/
<Buffer 6e 61 6d 65 3d 50 72 61 73 68 61 6e 74 26 67 65 6e 64 65 72 3d 6d 61 6c 65>
name=Prashant&gender=male
%
```



# 5.6 Parsing Request

```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  console.log(jsonObject);
  // Output: { name: 'Prashant', gender: 'male' }
});
fs.writeFileSync("user-details.txt", "Prashant Jain");
res.setHeader("Location", "/");
res.statusCode = 302;
return res.end();
```



# 5.6 Parsing Request

```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  const jsonString = JSON.stringify(jsonObject);
  console.log(jsonString);
  fs.writeFileSync("user-details.txt", jsonString);
});
res.setHeader("Location", "/");
res.statusCode = 302;
return res.end();
```

---

```
node > ≈ user-details.txt
```

```
1  {"name":"Prashant","gender":"male"}
```



# 5.7 Using Modules

JS app.js

JS handler.js

```
JS handler.js > ...
const fs = require("fs");
```

```
const requestHandler = (req, res) => {
  if (req.url === "/") {
    res.setHeader("Content-Type", "text/html");
  }
```

```
module.exports = requestHandler
```

JS app.js > ...

```
// Simple NodeJS server
const http = require('http');
const requestHandler = require('./handler');

const server = http.createServer(requestHandler);
```



## 5.7 Using Modules

```
// Method 1: Multiple exports using object  
module.exports = {  
  handler: requestHandler,  
  extra: "Extra"  
};
```

```
// Method 2: Setting multiple properties  
module.exports.handler = requestHandler;  
module.exports.extra = "Extra";
```

```
// Method 3: Shortcut using exports  
exports.handler = requestHandler;  
exports.extra = "Extra";
```

Complete Coding



# Practise Set

## Create a Calculator

1. Create a new Node.js project named “Calculator”.
2. On the home page (route “/”), show a welcome message and a link to the calculator page.
3. On the “/calculator” page, display a form with two input fields and a “Sum” button.

When the user clicks the “Sum” button, they should be taken to the “/calculate-result” page, which shows the sum of the two numbers.

- Make sure the request goes to the server.
- Create a **separate module** for the addition function.
- Create **another module** to handle incoming requests.
- On the “/calculate-result” page, parse the user input, use the addition module to calculate the sum, and **display the result on a new HTML page**.





# Revision

1. Streams

2. Chunks

3. Buffers

4. Reading Chunk

5. Buffering Chunks

6. Parsing Request

7. Using Modules





{ complete  
C O D I N G }



# 6. Event Loop

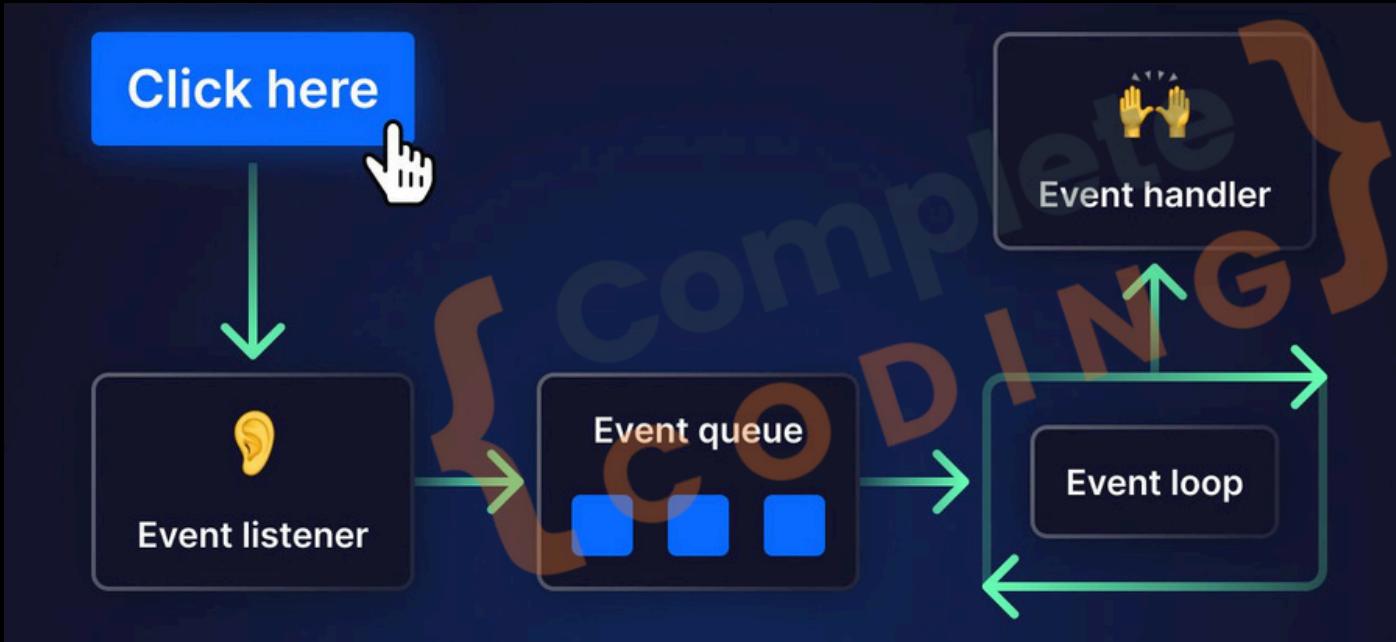
1. Event Driven
2. Single Threaded
3. V8 vs libuv
4. Node Runtime
5. Event Loop
6. Async Code
7. Blocking Code

X **Completed** CODING



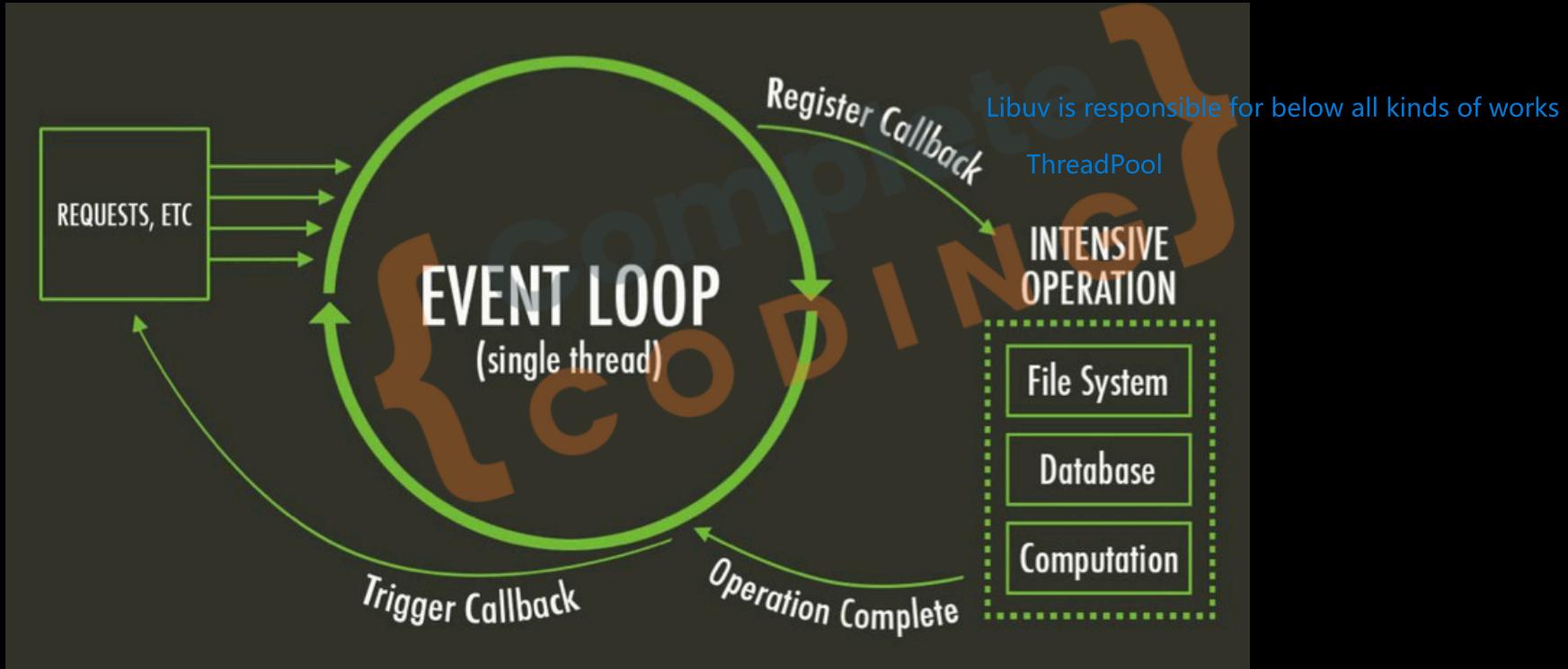


# 6.1 Event Driven





## 6.2 Single Threaded





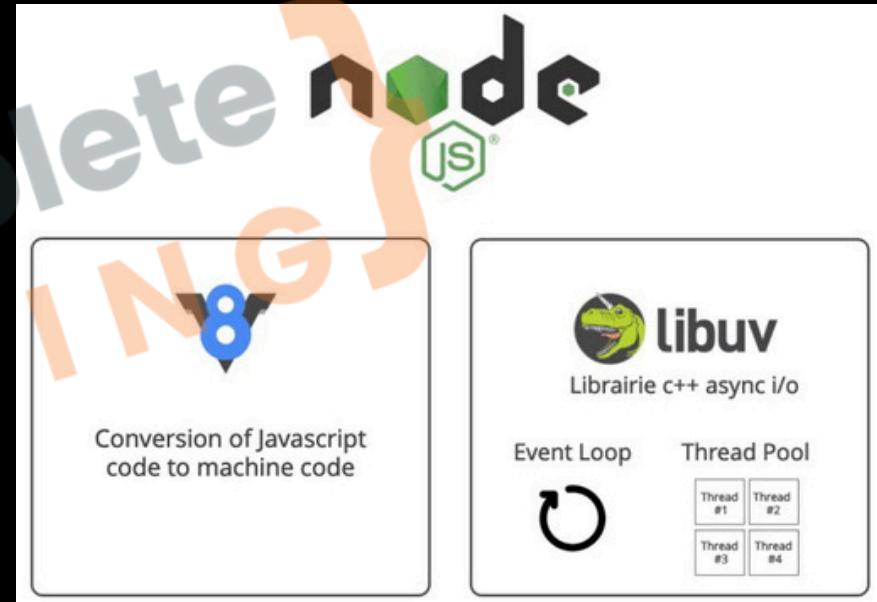
# 6.3 V8 vs libuv

## V8:

1. Open-source JavaScript engine by Google.
2. Used in Chrome and Node.js.
3. Compiles JavaScript to native machine code.
4. Ensures high-performance JavaScript execution.

## libuv:

1. Multi-platform support library for Node.js.
2. Handles asynchronous I/O operations.
3. Provides event-driven architecture.
4. Manages file system, networking, and timers non-blockingly across platforms.





# 6.4 Node Runtime

An invoked function is added to the call stack. Once it returns a value, it is popped off.

```
● ● ●  
console.log("Starting Node.js");  
  
db.query("SELECT * FROM public.cars", function (err, res) {  
  console.log("Query executed");  
});  
  
console.log("Before query result");
```





# 6.4 Node Runtime

Database queries or other I/O ops do not block Node.js single thread because Libuv API handles them.

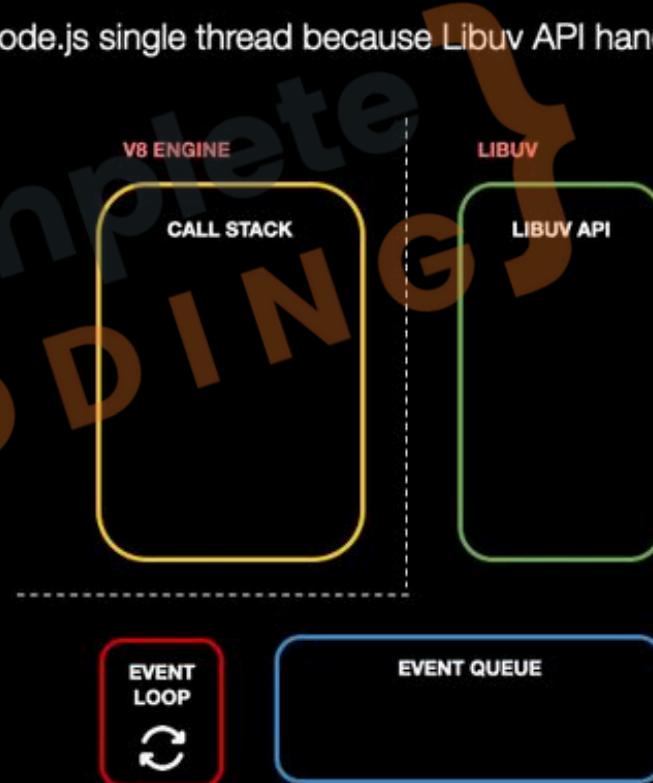
```
→
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res) {
  console.log("Query executed");
});

console.log("Before query result");
```

## OUTPUT

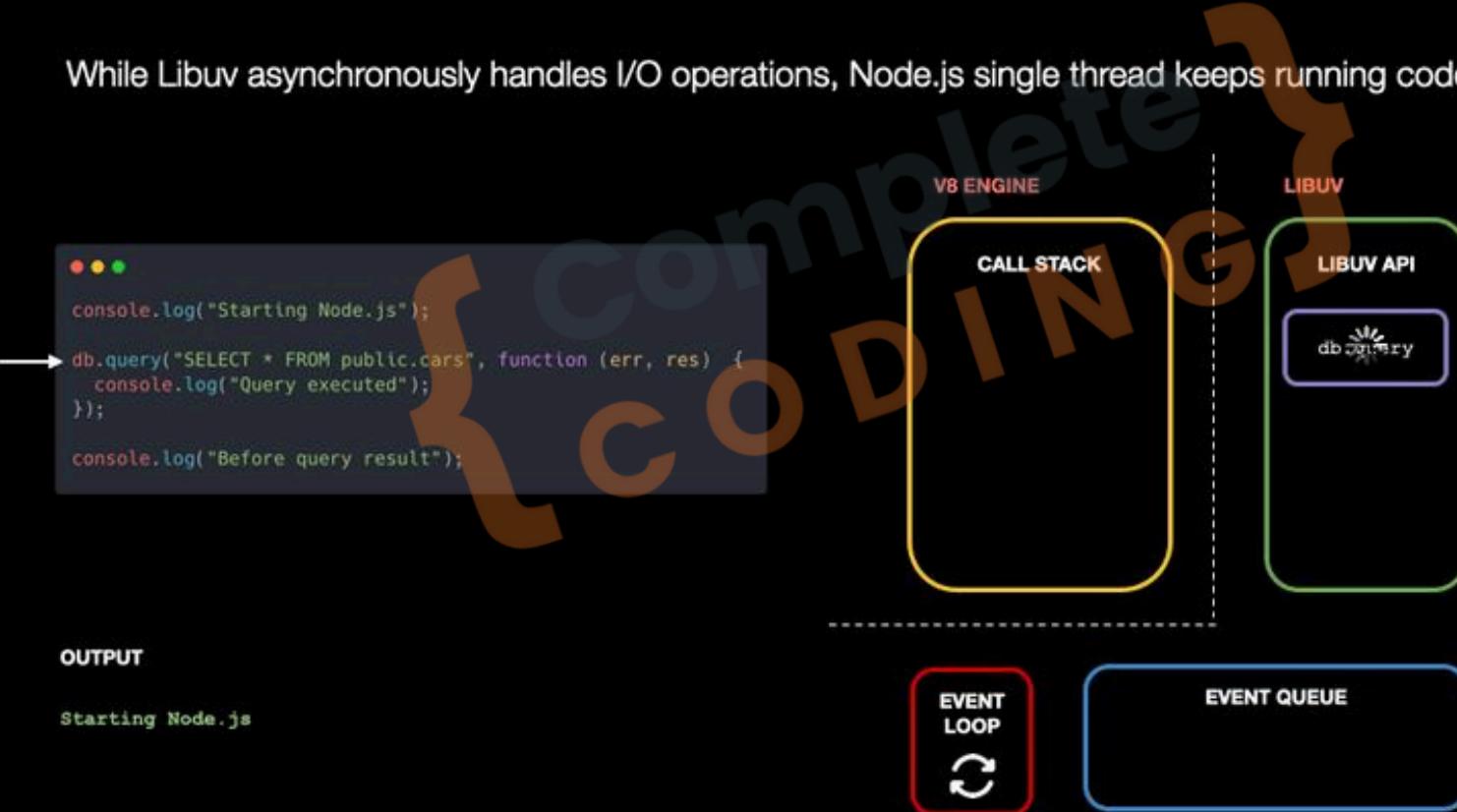
```
Starting Node.js
```





# 6.4 Node Runtime

While Libuv asynchronously handles I/O operations, Node.js single thread keeps running code.

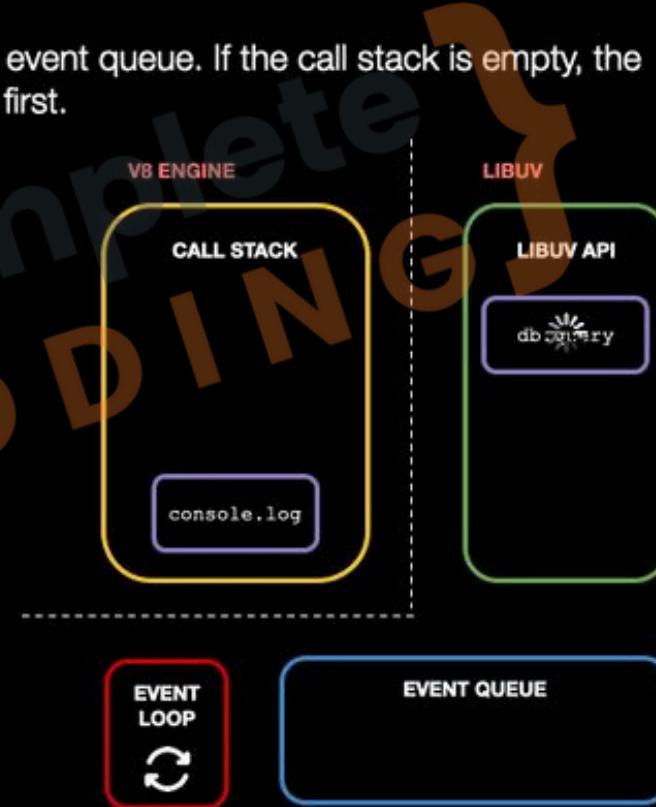




# 6.4 Node Runtime

Callbacks of completed queries are moved to the event queue. If the call stack is empty, the event loop checks for callbacks and transfers the first.

```
● ● ●  
console.log("Starting Node.js");  
  
db.query("SELECT * FROM public.cars", function (err, res) {  
  console.log("Query executed");  
});  
  
→ console.log("Before query result");
```



## OUTPUT

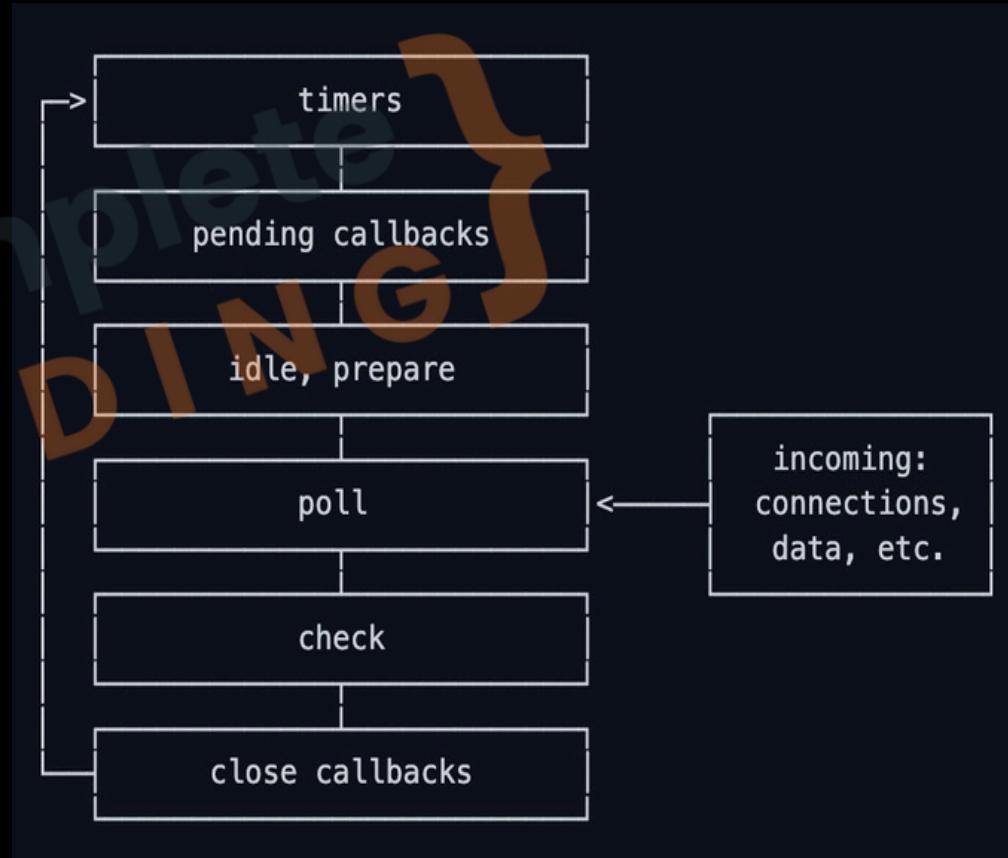
Starting Node.js

Before query result



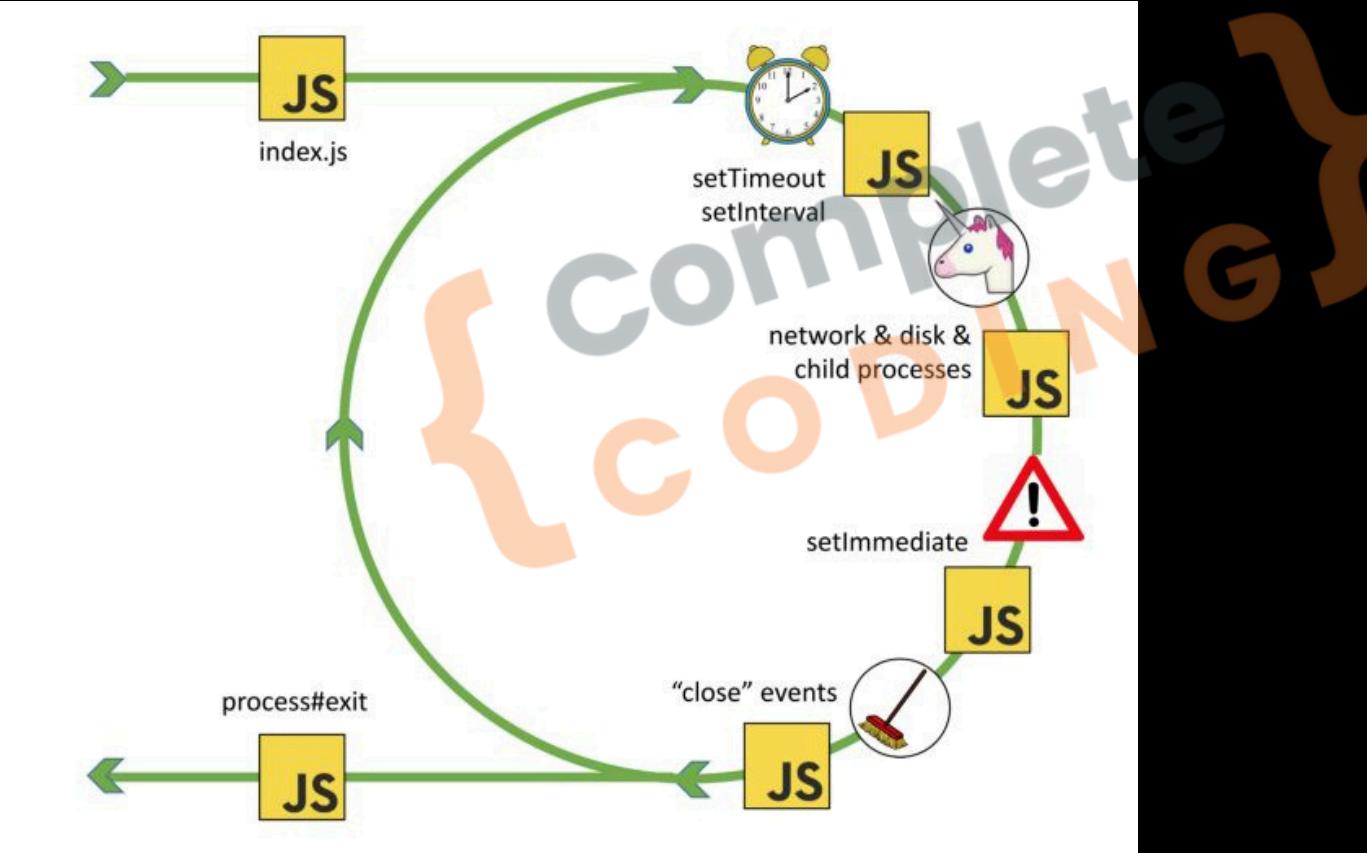
# 6.5 Event Loop

- **timers**: this phase executes callbacks scheduled by `setTimeout()` and `setInterval()`.
- **pending callbacks**: executes I/O callbacks deferred to the next loop iteration.
- **idle, prepare**: only used internally.
- **poll**: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by **timers**, and `setImmediate()`); node will block here when appropriate. check: `setImmediate()`
- callbacks are invoked here.
- **close callbacks**: some close callbacks, e.g. `socket.on('close', ...)`.





# 6.5 Event Loop





# 6.6 Async Code

```
const jsonString = JSON.stringify(jsonObject);
console.log(jsonString);
fs.writeFileSync("user-details.txt", jsonString);
res.setHeader("Location", "/");
res.statusCode = 302;
res.end();
};

res.write('<body><h1>Like / Share / Subscribe</h1></
body>');
res.write('</html>');
return res.end();
```

```
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
at ServerResponse.setHeader (node:_http_outgoing:699:11)
at IncomingMessage.<anonymous> (/Users/prashantjain/workspace/Test Project/node/app.js:44:11)
at IncomingMessage.emit (node:events:532:35)
at endReadableNT (node:internal/streams/readable:1696:12)
at process.processTicksAndRejections (node:internal/process/task_queues:82:21) {
  code: 'ERR_HTTP_HEADERS_SENT'
}
```



# 6.6 Async Code

```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  const jsonString = JSON.stringify(jsonObject);
  console.log(jsonString);
  fs.writeFileSync("user-details.txt", jsonString);
  res.setHeader("Location", "/");
  res.statusCode = 302;
  return res.end();
});
```



# 6.7 Blocking Code

```
const jsonString = JSON.stringify(jsonObject);
console.log(jsonString);
// BLOCKING EVERYTHING
fs.writeFileSync("user-details.txt", jsonString);
res.setHeader("Location", "/");
```



# 6.7 Blocking Code

```
console.log(jsonString);
// Async Operation
fs.writeFile("user-details.txt", jsonString, error => {
  res.setHeader("Location", "/");
  res.statusCode = 302;
  return res.end();
});
```





# Run & Observe

## Blocking vs Async

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

console.log('1. Start of script');

// Synchronous (blocking) operation
console.log('2. Reading file synchronously');
const dataSync = fs.readFileSync('user-details.txt', 'utf8');
console.log('3. Synchronous read complete');

// Asynchronous (non-blocking) operation
console.log('4. Reading file asynchronously');
fs.readFile('user-details.txt', 'utf8', (err, dataAsync) => {
if (err) throw err;
console.log('6. Asynchronous read complete');
});

console.log('5. End of script');
```

complete



1. Start of script
2. Reading file synchronously
3. Synchronous read complete
4. Reading file asynchronously
5. End of script
6. Asynchronous read complete



# Run & Observe

## Event Loop Sequence

```
console.log('1. Start of script');

// Microtask queue (Promise)
Promise.resolve().then(() => console.log('2. Microtask 1'));

// Timer queue
setTimeout(() => console.log('3. Timer 1'), 0);

// I/O queue
const fs = require('fs');
fs.readFile('user-details.txt', () => console.log('4. I/O operation'));

// Check queue
setImmediate(() => console.log('5. Immediate 1'));

// Close queue
process.on('exit', (code) =>
{
  console.log('6. Exit event');
});

console.log('7. End of script');
```



1. Start of script
2. Microtask 1
3. Timer 1
4. I/O operation
5. Immediate 1
6. Exit event
7. End of script



# Revision

1. Event Driven
2. Single Threaded
3. V8 vs libuv
4. Node Runtime
5. Event Loop
6. Async Code
7. Blocking Code





{ complete  
C O D I N G }



# 7. NPM & Tools



1. Install Material Icons
2. npm init
3. npm Scripts
4. npm Packages
5. Installing Packages
6. Installing nodemon
7. Using nodemon

A large, semi-transparent watermark reading "COMPLETED CODING" in a stylized, orange font, angled across the center of the slide.





# 7.1 Install Material Icons



## Material Icon Theme v5.11.1

Philipp Kief

25,491,060

★★★★★ (345)

Sponsor

Material Design Icons for Visual Studio Code

[Set File Icon Theme](#)

[Disable](#) ▾

[Uninstall](#) ▾

Auto Update



# 7.2 npm init



## npm init

```
prashantjain@Prashants-Mac-mini user % npm init
```

This utility will walk you through creating a package.json file.  
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields  
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and  
save it as a dependency in the package.json file.

Press ^C at any time to quit.

```
package name: (user) User Backend
```

Sorry, name can only contain URL-friendly characters and name can no longer contain capital letters.

```
package name: (user) user-backend
```

```
version: (1.0.0)
```

```
description: This project will have the backend code of our User project.
```

```
entry point: (user.js) app.js
```

```
test command:
```

```
git repository:
```

```
keywords:
```

```
author: Complete Coding
```

```
license: (ISC)
```

About to write to /Users/prashantjain/workspace/NodeJS\_Complete\_YouTube/Chapter 6 – Event Loop/user/package.json:



# 7.3 npm Scripts



```
{  
  "name": "user-backend",  
  "version": "1.0.0",  
  "description": "This project will have the backend code of our User  
  project.",  
  "main": "app.js",  
  > Debug  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1",  
    "start": "node app.js",  
    "khul-ja-sim-sim": "node app.js"  
  },  
  "author": "Complete Coding",  
  "license": "ISC"  
}
```

npm start

npm run khul-ja-sim-sim

Complete Coding



# 7.4 npm Packages



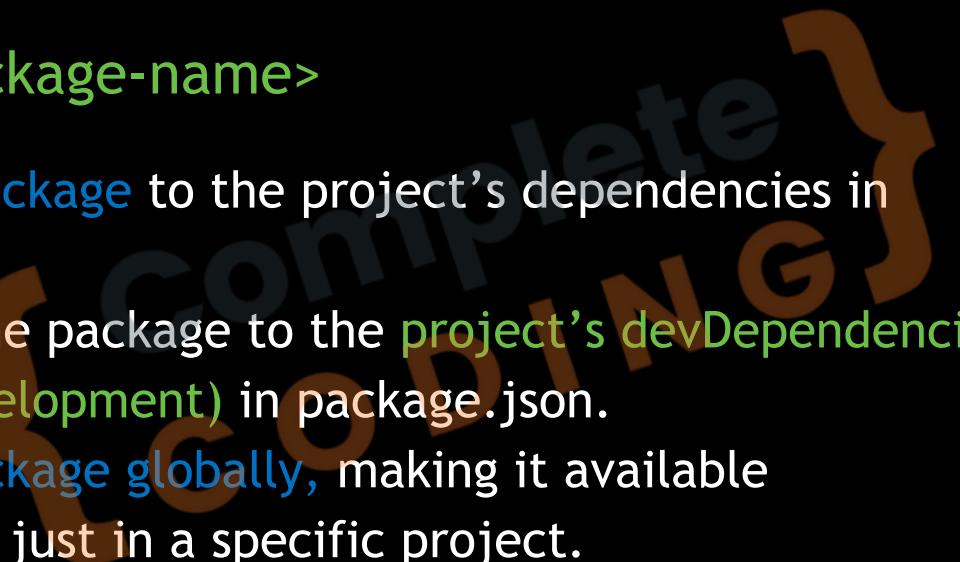
1. npm: Node.js package manager for code sharing.
2. Package: Reusable code or library.
3. package.json: Defines package metadata and dependencies.
4. Versioning: Manages different package versions.
5. Local/Global: Install packages locally or globally.
6. Registry: Public storage for open-source packages.
7. Examples: Express, React, Lodash.





# 7.5 Installing Packages

`npm install <package-name>`

- 
- A large, semi-transparent watermark reading "Learning Node.js" is positioned diagonally across the slide. The text is in a stylized, rounded font with "Learning" in blue and "Node.js" in orange.
- 1.-**save**: Adds the package to the project's dependencies in `package.json`.
  - 2.-**save-dev**: Adds the package to the `project's devDependencies` (used only in development) in `package.json`.
  - 3.-**g**: Installs the package globally, making it available system-wide, not just in a specific project.
  - 4.-**save-exact**: Installs the exact version specified without updating for newer versions.
  - 5.-**force**: Forces npm to fetch and install packages even if they are already installed.



# 7.6 Installing nodemon



npm install nodemon --save-dev

```
user
├── node_modules
├── app.js
├── package-lock.json
├── package.json
└── user.js
    └── user.txt
```

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node app.js",
  "khul-ja-sim-sim": "node app.js"
},
"author": "Complete Coding",
"license": "ISC",
"devDependencies": {
  "nodemon": "^3.1.7"
}
```

npm install

Recreates node\_modules



## 7.7 Using nodemon



```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "nodemon app.js",  
  "khul-ja-sim-sim": "node app.js"  
},
```

```
prashantjain@Prashants-Mac-mini user % nodemon app.js
```

```
zsh: command not found: nodemon
```

```
prashantjain@Prashants-Mac-mini user % npm start
```

```
> user-backend@1.0.0 start
```

```
> nodemon app.js
```

```
[nodemon] 3.1.7
```

```
[nodemon] to restart at any time, enter `rs`
```

```
[nodemon] watching path(s): .*
```

```
[nodemon] watching extensions: js,mjs,cjs,json
```

```
[nodemon] starting `node app.js`
```

```
Server running on address http://localhost:3001
```

npm install nodemon -g



# Revision

1. Installation Icons
2. npm init
3. npm Scripts
4. npm Packages
5. Installing Packages
6. Installing nodemon
7. Using nodemon



Complete  
Coding



{ complete  
C O D I N G }



# 8. Errors & Debugging

1. Types of Errors
2. Syntax Errors
3. Runtime Errors
4. Logical Errors
5. Using the Debugger
6. Debugger with Async Code
7. Restart Debug with nodemon





# 8.1 Types of Errors



- 1. Syntax Error:** An **error** in the **code's structure**, causing it to not compile or run (e.g., missing semicolon).
- 2. Logical Error:** The code runs but **produces incorrect results** due to **faulty logic** (e.g., wrong formula).
- 3. Runtime Error:** An **error** that occurs while the program is running, often due to **invalid operations**



## 8.2 Syntax Errors

```
// Missing parenthesis in function call  
console.log("Hello, world"
```

```
// Unclosed string literal  
let message = "Welcome to Node.js;
```

```
// Improper use of reserved keywords  
let new = 5;
```

```
// Incorrect variable declaration (const needs an initial value)  
const myVar;
```



## 8.3 Runtime Errors

```
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
  at ServerResponse.setHeader (node:_http_outgoing:699:11)
  at IncomingMessage.<anonymous> (/Users/prashantjain/workspace/Test Project/node/app.js:44:11)
  at IncomingMessage.emit (node:events:532:35)
  at endReadableNT (node:internal/streams/readable:1696:12)
  at process.processTicksAndRejections (node:internal/process/task_queues:82:21) {
  code: 'ERR_HTTP_HEADERS_SENT'
}

// Reference Error (x is not defined)
console.log(x);

// Type Error (num is not a function)
let num = 10;
num();

// Invalid JSON parse (SyntaxError)
let jsonString = "{ name: 'John' }"; // Invalid JSON (single quotes)
JSON.parse(jsonString);

// File not found error (fs module)
const fs = require('fs');
fs.readFileSync('nonexistentFile.txt'); // Throws Error: ENOENT (file not found)
```



## 8.4 Logical Errors

```
let x = 5;
if (x = 10) { // Assignment instead of comparison
|   console.log("x is 10"); // Incorrectly prints this
}

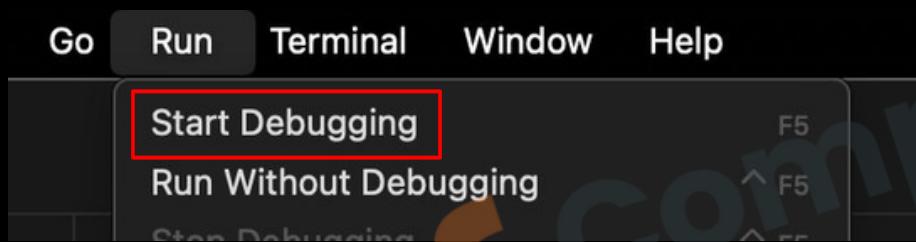
let arr = [1, 2, 3, 4, 5];
for (let i = 0; i <= arr.length; i++) {
|   console.log(arr[i]); // Prints undefined at the end of the loop
}

let num = "10";
console.log(num + 5); // Expected result: 15, prints 105
```

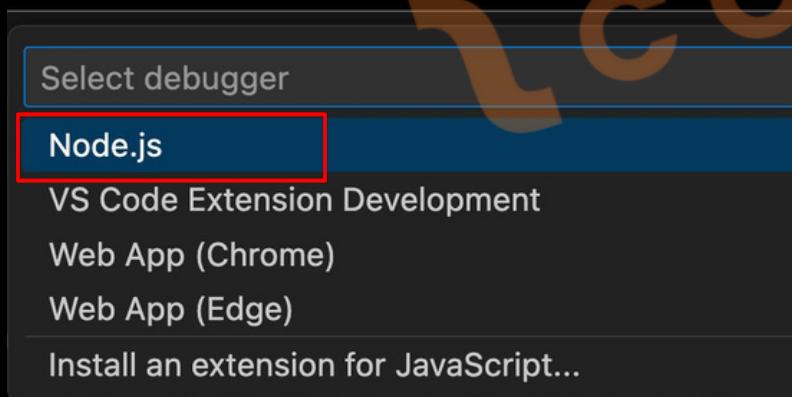


# 8.5 Using the Debugger

Step 1



Step 2



Step 3: Put a breakpoint

```
1 let x = 5;
2 if (x = 10) { // Assignment instead of comparison
3   console.log("x is 10"); // Incorrectly prints this
4 }
5
```



# 8.5 Using the Debugger

Step 4: Use the tools

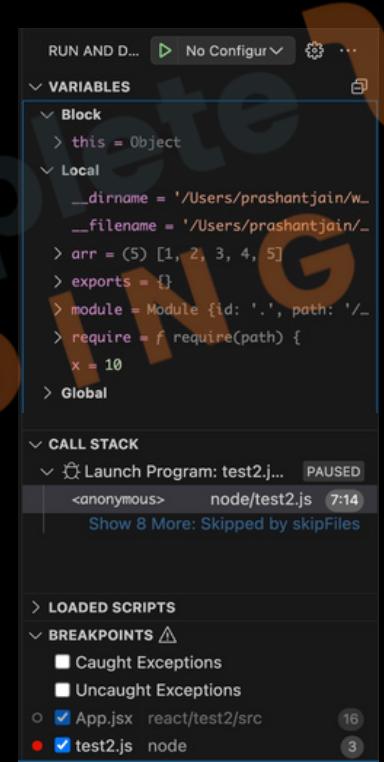


Step 5: Hover

```
test2.js
let arr = [1, 2, 3, 4, 5];
for (let i = 0; i <= arr.length; i++) {
    console.log(arr[i]); // Prints undefined
}

(5) [1, 2, 3, 4, 5]
  0 = 1
  1 = 2
  2 = 3
  3 = 4
  4 = 5
  length = 5
  > [[Prototype]] = Array(0)
  > [[Prototype]] = Object

Hold Option key to switch to editor language hover
```





# 8.5 Using the Debugger

## Step 7: Using Debug Console

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
/opt/homebrew/bin/node ./node/test2.js
```

```
→ x  
10  
→ x++  
10  
→ x === 15  
false
```

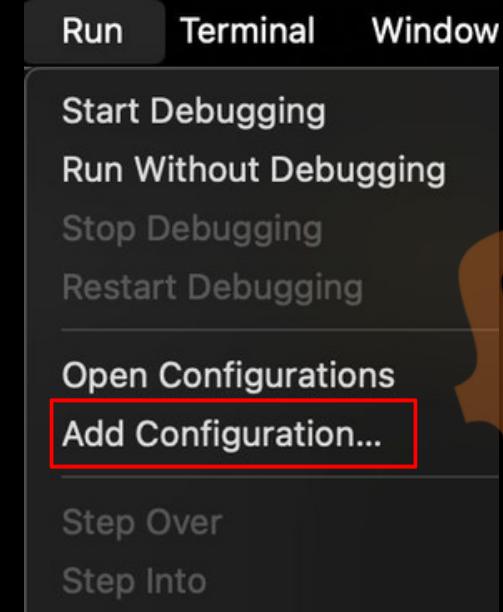
```
>
```



# 8.6 Debugger with Async Code

```
• 31  console.log("Here");
32  req.on("end", () => {
• 33    const parsedBody = Buffer.concat(body).toString();
34    console.log(parsedBody);
35    const params = new URLSearchParams(parsedBody);
36    const jsonObject = {};
37    for (const [key, value] of params.entries()) {
38      jsonObject[key] = value;
39    }
40    const jsonString = JSON.stringify(jsonObject);
41    console.log(jsonString);
42    // Async Operation
43    fs.writeFile("user-details.txt", jsonString, error => {
• 44      res.setHeader("Location", "/");
45      res.statusCode = 302;
46      return res.end();
47    });
48  });
49 }
```

# node 8.7 Restart Debug with nodemon



```
"version": "0.2.0",
"configurations": [
  {
    "type": "node",
    "request": "launch",
    "name": "Launch Program",
    "skipFiles": [
      "<node_internals>/**"
    ],
    "program": "${workspaceFolder}/node/test2.js",
    "restart": true,
    "runtimeExecutable": "nodemon",
    "console": "integratedTerminal"
  }
]
```





# Practise Set

Debug and fix Syntax, Runtime and Logical Errors

```
function calculateArea(width, height {  
    return width + height;  
}
```

```
let width = 10 height = 5;
```

```
if (area > 100) {  
    console.log("The area is large.");  
} else {  
    console.log("The area is small.");  
}
```

```
if (width + height > 100) {  
    console.log("Area is greater than or equal to 100");  
}
```

Complete CODING





# Revision

1. Try passing Errors
2. Syntax Errors
3. Runtime Errors
4. Logical Errors
5. Using the Debugger
6. Debugger with Async Code
7. Restart Debug with nodemon





{ complete  
C O D I N G }



# 9. Starting with Express.js

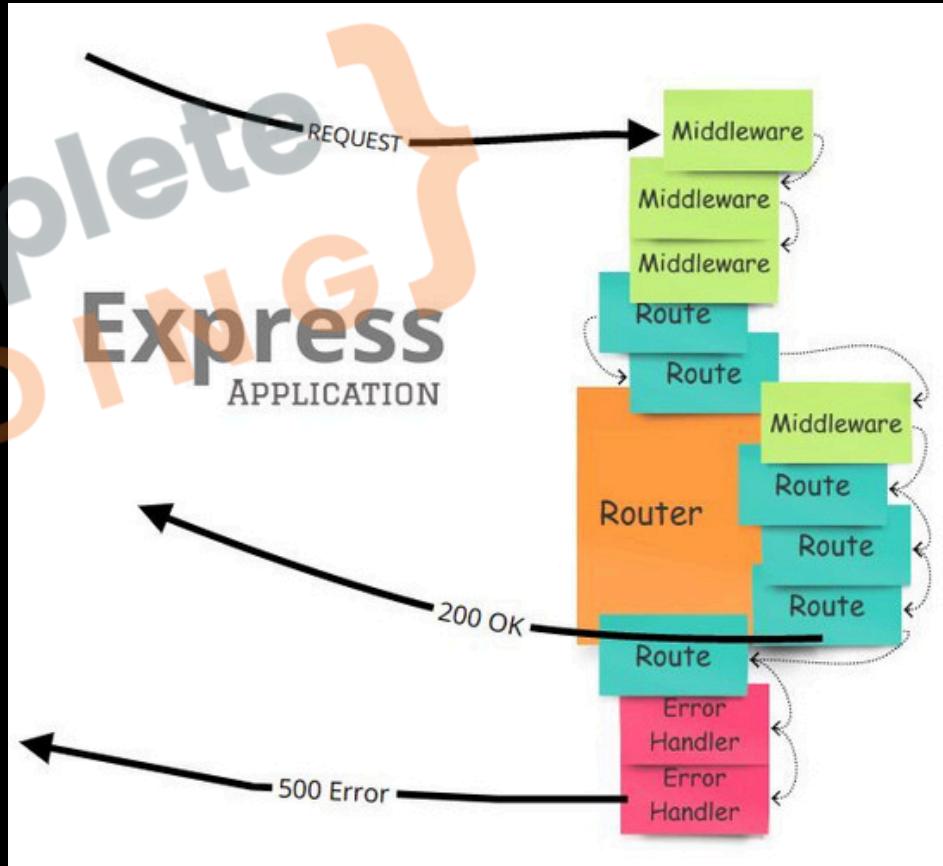
- 1.What is Express.js
- 2.Need of Express.js
- 3.Installing Express.js
- 4.Adding Middleware
- 5.Sending Response
- 6.Express DeepDive
- 7.Handling Routes

Complete  
Coding



# 9.1 What is Express.js

1. Express.js is a **minimal and flexible** web application framework for Node.js.
2. It provides a **robust set of features** for building **single-page, multi-page, and hybrid web applications**.
3. Express.js **simplifies server-side coding** by providing a layer of fundamental web application features.

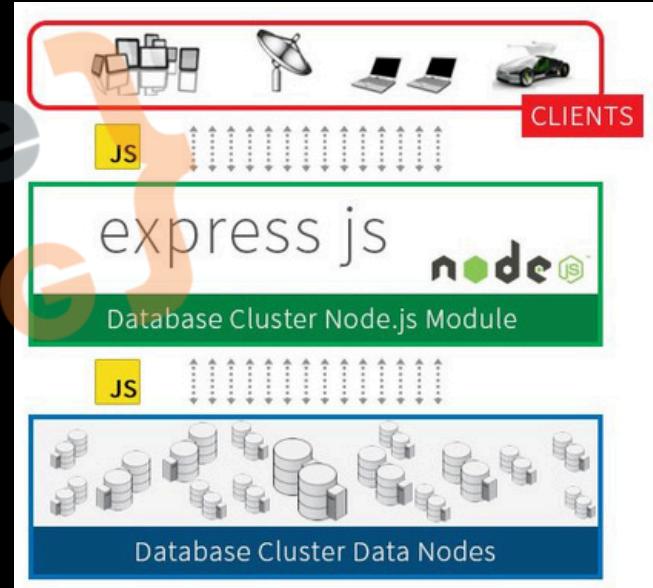




## 9.2 Need of Express.js

EXPRESS Js

1. **Express.js Simplifies Server Creation:** Helps in quickly setting up and running a **web server** without the need for complex coding.
2. **Routing Management:** Provides a **powerful routing mechanism** to handle URLs and HTTP methods effectively.
3. **Middleware Support:** Allows the **use of middleware** to handle requests, responses, and any middle operations, making code modular and maintainable.
4. **API Development:** Facilitates **easy and efficient** creation of RESTful APIs.
5. **Community and Plugins:** Has a **large ecosystem** with numerous **plugins and extensions**, accelerating development time.





# 9.3 Installing Express.js

EXPRESS JS

```
$ npm install express
```

To install Express temporarily and not add it to the dependencies list:

```
$ npm install express --no-save
```

By default with version npm 5.0+, `npm install` adds the module to the `dependencies` list in the `package.json` file; with earlier versions of npm, you must specify the `--save` option explicitly. Then, afterwards, running `npm install` in the app directory will automatically install modules in the `dependencies` list.



# 9.3 Installing Express.js

EXPRESS JS

```
// Core Modules
const http = require('http');
// External Modules
const express = require('express');

const app = express();

const server = http.createServer(app);

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

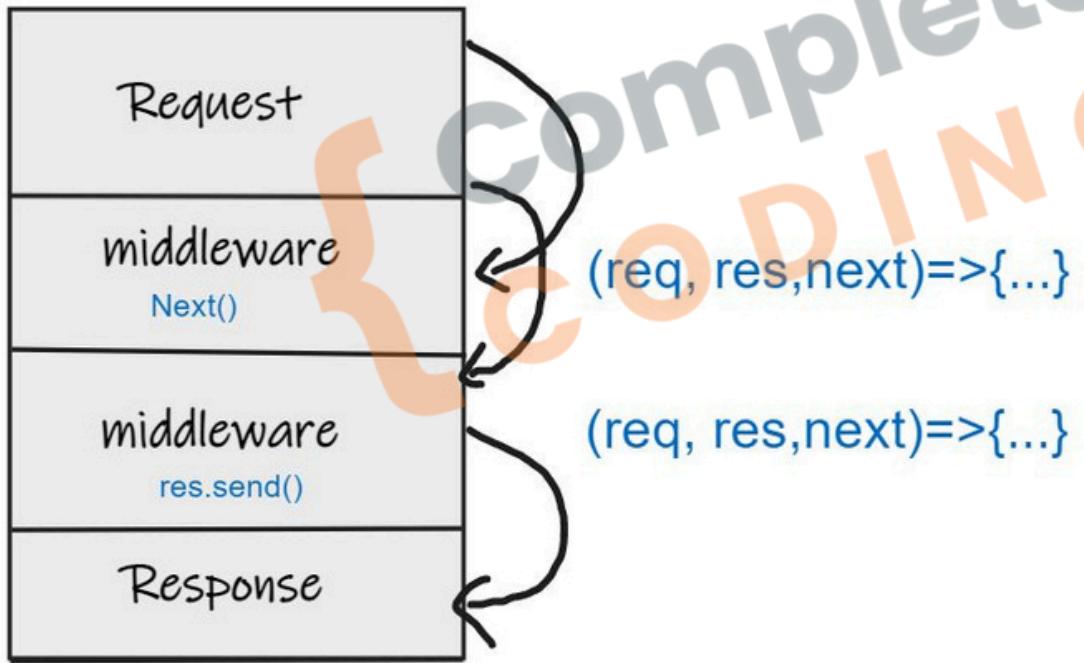
prashantjain@Prashants-Mac-mini ~ % node % npm start

```
> node@1.0.0 start
> nodemon app.js

[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Server running at http://localhost:3000/
```

# 9.4 Adding Middleware

it is ALL About middleware





# 9.4 Adding Middleware

EXPRESS Js

```
// Adding Middleware
app.use((req, res, next) => {
  console.log("First Middleware", req.url, req.method);
  next();
});

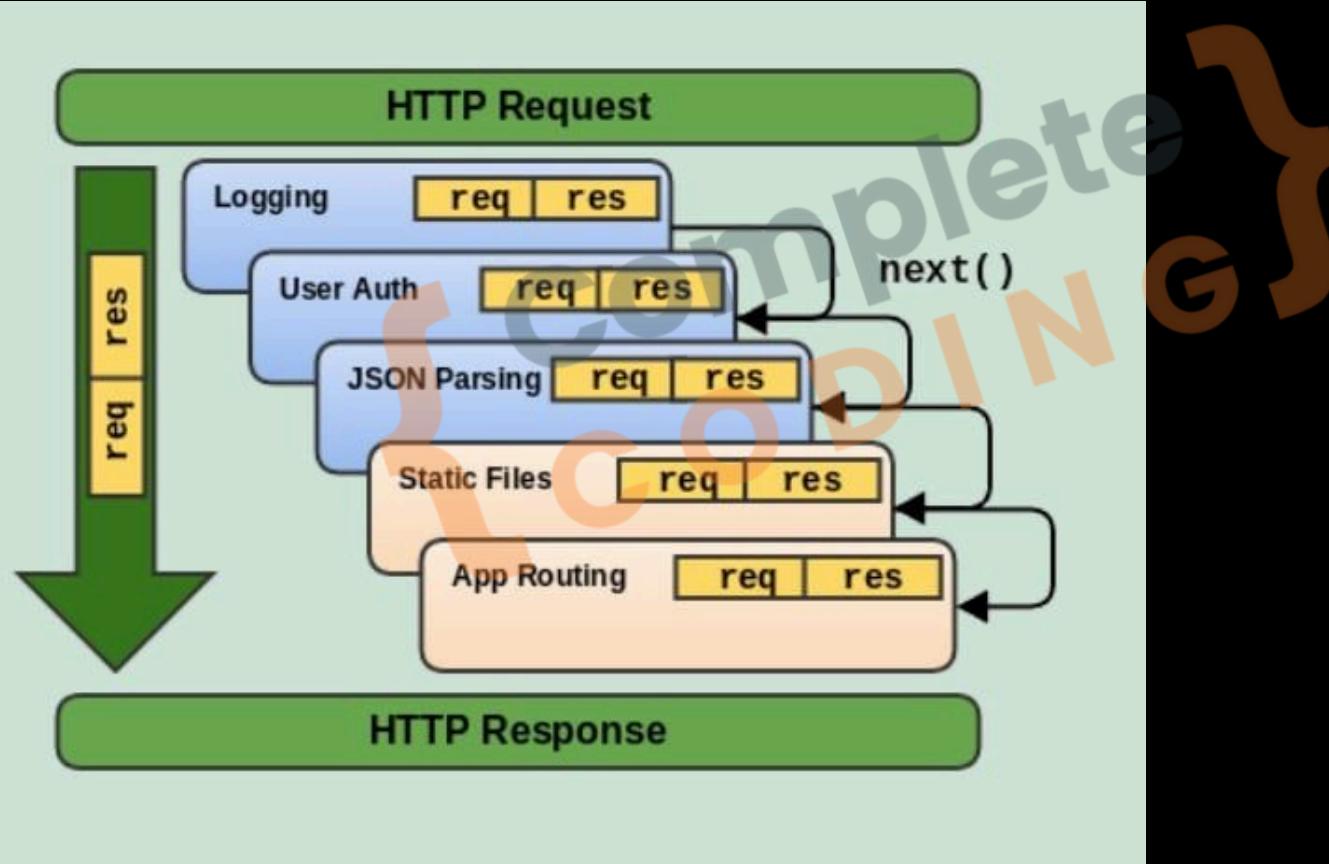
app.use((req, res, next) => {
  console.log("Second Middleware", req.url, req.method);
});

const server = http.createServer(app);
```

Complete Coding

```
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Server running at http://localhost:3000/
First Middleware / GET
Second Middleware / GET
```

# 9.4 Adding Middleware

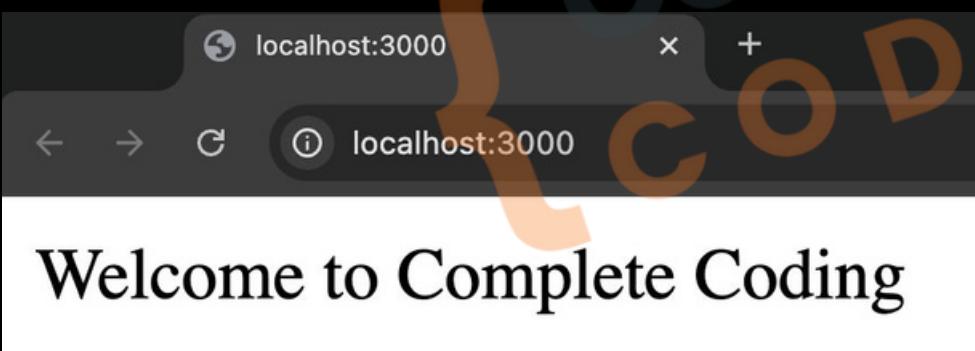




# 9.5 Sending Response

EXPRESS Js

```
app.use((req, res, next) => {
  console.log("Second Middleware", req.url, req.method);
  res.send('<p>Welcome to Complete Coding</p>');
});
```



Response Headers	
<input type="checkbox"/>	Raw
Connection:	keep-alive
Content-Length:	33
Content-Type:	text/html; charset=utf-8
Date:	Wed, 02 Oct 2024 07:45:48 GMT
Etag:	W/"21-FoFd61RXqayGnfodqIQO62RgMzY"
Keep-Alive:	timeout=5
X-Powered-By:	Express



# 9.6 Express DeepDive

EXPRESS Js

expressjs / express

Type ⌘ to search

Code Issues 110 Pull requests 66 Discussions Actions Wiki Security 2 Insights

express Public Watch 1695

master 16 Branches 295 Tags Go to file Add file Code

jonchurch remove --bail from test script (#5962) 6340d15 · yesterday 5,980 Commits

.github/workflows update CI, remove unsupported versions, clean up 3 weeks ago

benchmarks docs: add documentation for benchmarks 8 months ago

examples Delete back as a magic string (#5933) 3 weeks ago

lib Delete back as a magic string (#5933) 3 weeks ago

test Delete back as a magic string (#5933) 3 weeks ago

.editorconfig build: Add .editorconfig 7 years ago



# 9.6 Express DeepDive

EXPRESS Js

```
101  /**
102   * Send a response.
103   *
104   * Examples:
105   *
106   *     res.send(Buffer.from('wahoo'));
107   *     res.send({ some: 'json' });
108   *     res.send('<p>some html</p>');
109   *
110  * @param {string|number|boolean|object|Buffer} body
111  * @public
112  */
113
114  res.send = function send(body) {
115    var chunk = body;
116    var encoding;
117    var req = this.req;
118    var type;
119
120    // settings
121    var app = this.app;
122
```

```
// string defaulting to html
case 'string':
  if (!this.get('Content-Type')) {
    this.type('html');
  }
break;

case 'boolean':
```



# 9.6 Express DeepDive

EXPRESS JS

```
// Core Modules
const http = require('http');
// External Modules
const express = require('express');

const app = express();

// Adding Middleware
app.use((req, res, next) => {
  console.log("First Middleware", req.url, req.method);
  next();
});

app.use((req, res, next) => {
  console.log("Second Middleware", req.url, req.method);
  res.send('<p>Welcome to Complete Coding</p>');
});

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}/`);
});
```

Complete Coding



# 9.7 Handling Routes

EXPRESS Js

## app.use([path,] callback [, callback...])

Mounts the specified [middleware](#) function or functions at the specified path: the middleware function is executed when the base of the requested path matches `path`.

### Arguments

Argument	Description	Default
<code>path</code>	<p>The path for which the middleware function is invoked; can be any of:</p> <ul style="list-style-type: none"><li>• A string representing a path.</li><li>• A path pattern.</li><li>• A regular expression pattern to match paths.</li><li>• An array of combinations of any of the above.</li></ul> <p>For examples, see <a href="#">Path examples</a>.</p>	'/' (root path)



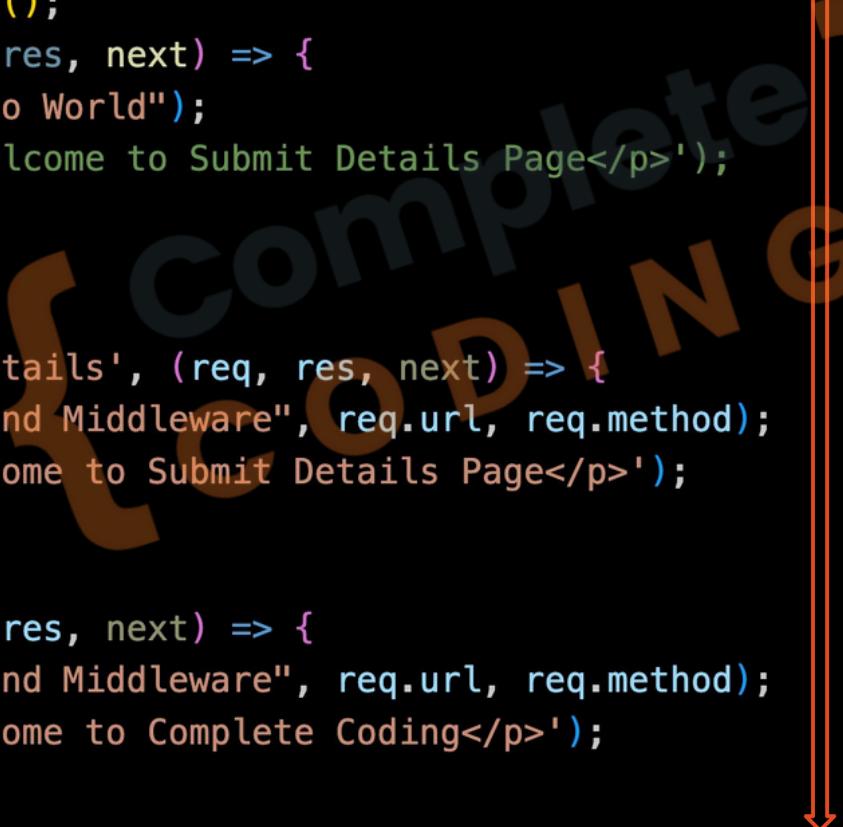
# 9.7 Handling Routes

EXPRESS Js

```
const app = express();
app.use('/', (req, res, next) => {
  console.log("Hello World");
  //res.send('<p>Welcome to Submit Details Page</p>');
  next();
});

app.use('/submit-details', (req, res, next) => {
  console.log("Second Middleware", req.url, req.method);
  res.send('<p>Welcome to Submit Details Page</p>');
});

app.use('/', (req, res, next) => {
  console.log("Second Middleware", req.url, req.method);
  res.send('<p>Welcome to Complete Coding</p>');
});
```

- 
1. Order matters
  2. Can not call `next()` after `send()`
  3. "/" matches everything
  4. Calling `res.send` implicitly calls `res.end()`

# 9.7 Handling Routes

```
var express = require('express');
var app = express();
app.get('/', function(req, res, next) {
  next();
})
app.listen(3000);
```

HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.



# 9.7 Handling Routes

EXPRESS Js

```
const app = express();
app.post('/', (req, res, next) => {
    console.log("Hello World");
    //res.send('<p>Welcome to Submit Details Page</p>');
    next();
});

app.use('/submit-details', (req, res, next) => {
    console.log("Second Middleware", req.url, req.method);
    res.send('<p>Welcome to Submit Details Page</p>');
});

app.use('/', (req, res, next) => {
    console.log("Second Middleware", req.url, req.method);
    res.send('<p>Welcome to Complete Coding</p>');
});
```



# Practise Set

Create a new project.

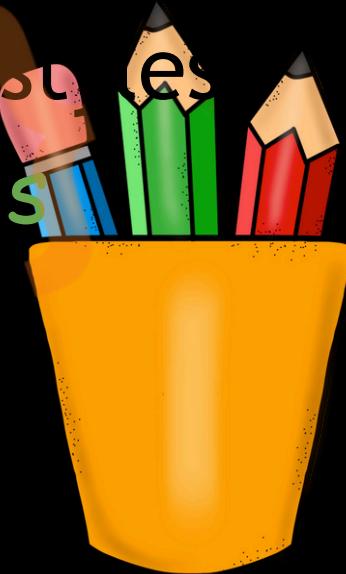
1. Install nodemon and express.
2. Add two dummy middleware that logs request path and request method respectively.
3. Add a third middleware that returns a response.
4. Now add handling using two more middleware that handle path /, a request to /contact-us page.
5. Contact us should return a form with name and email as input fields that submits to /contact-us page also.
6. Also handle POST incoming request to /contact-us path using a separate middleware.





# Revision

1. What is Express.js
2. Need of Express.js
3. Installing Express.js
4. Adding Middleware
5. Sending Response
6. Express DeepDive
7. Handling Routes





{ complete  
C O D I N G }



# 10. Express.js DeepDive

1. Parsing Requests
2. Express Router
3. Adding 404
4. Common Paths
5. Adding HTML Files
6. Serving HTML Files
7. Using File Helper

Complete CODING





# 10.1 Parsing Requests

EXPRESS Js

```
// External Modules
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

app.use(bodyParser.urlencoded({}));

app.get('/details', (req, res, next) => {
  console.log("Hello World");
  res.send(`

    <h1>Enter Your Details:</h1>
    <form action="/submit-details" method="POST">
      <input type="text" name="username" placeholder="Enter your name"><br>
      <br><input type="submit" value="Submit">
    </form>
  `);
});

app.post('/submit-details', (req, res, next) => {
  console.log(req.url, req.method, req.body);
  res.redirect('/');
});
```

Where Search destinations

Check in Add dates

Check out Add dates

Who Add guests

Stays Experiences Airbnb your home

Icons Amazing pools Farms Amazing views Surfing Ski-in/out Rooms Cabins Countryside Treehouses OMG! Luxe Beach

00h | 30m | 04s

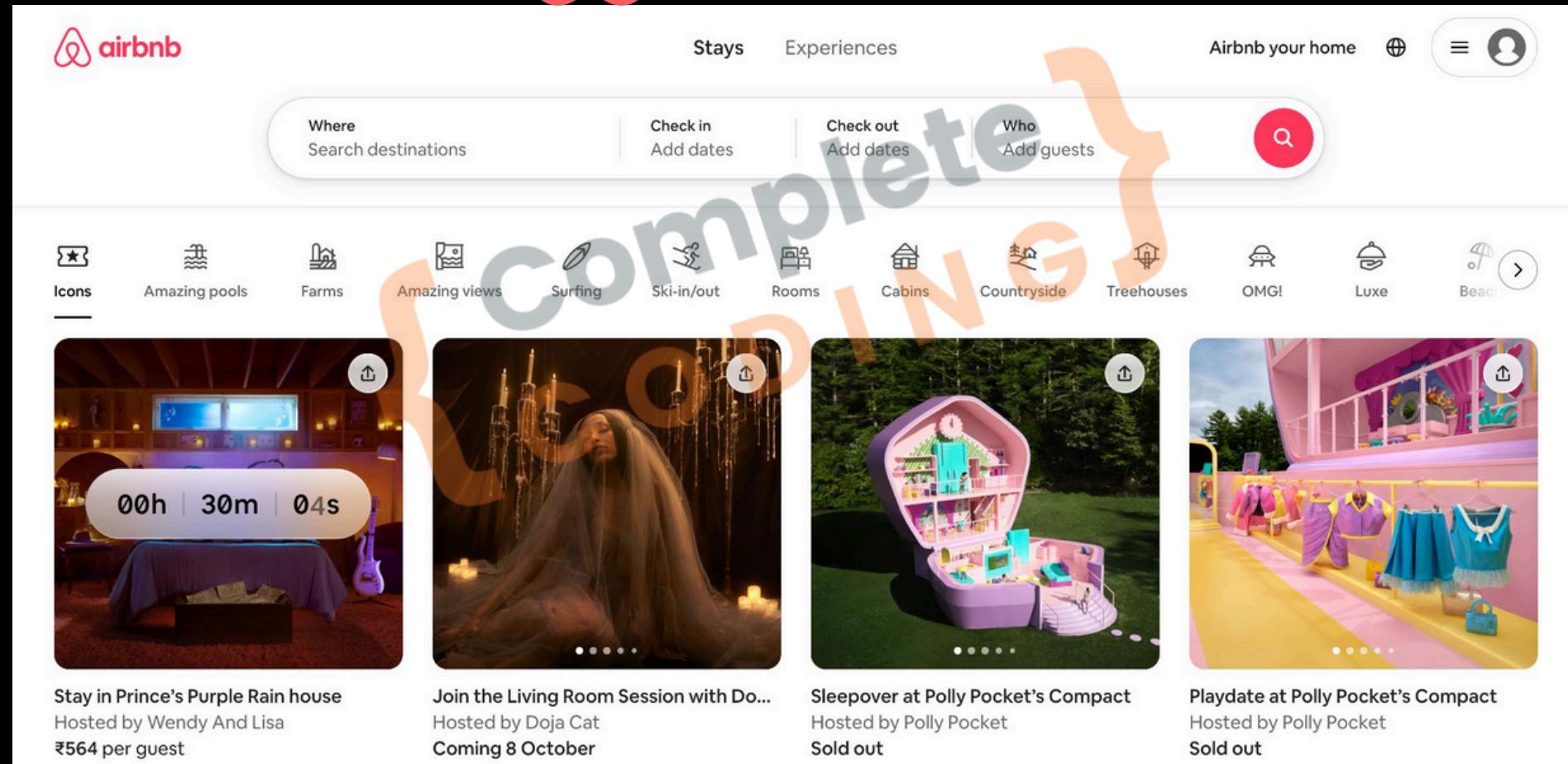
Stay in Prince's Purple Rain house Hosted by Wendy And Lisa ₹564 per guest

Join the Living Room Session with Do... Hosted by Doja Cat Coming 8 October

Sleepover at Polly Pocket's Compact Hosted by Polly Pocket Sold out

Playdate at Polly Pocket's Compact Hosted by Polly Pocket Sold out

complete





# 10.2 Express Router

EXPRESS JS

node > air-bnb > app.js > ...

```
1 // External Modules
2 const express = require('express');
3 // Local Modules
4 const hostRouter = require('./routes/host');
5 const userRouter = require('./routes/user');
6
7 const app = express();
8
9 app.use(express.urlencoded({extended:true}));
10 app.use(hostRouter);
11 app.use(userRouter);
12
13 const PORT = 3000;
14 app.listen(PORT, () => {
15   |   console.log(`Server running at http://localhost:${PORT}/`);
16 })
```



# 10.2 Express Router

EXPRESS JS

node > air-bnb > routes > host.js > ...

```
1 // External Modules
2 const express = require('express');
3
4 const router = express.Router();
5
6 router.get('/add-home', (req, res, next) => {
7   res.send(`
8     <h1>Register Your Home on AirBnB</h1>
9     <form action="/submit-home" method="POST">
10       <input type="text" name="houseName" placeholder="Enter your House Name"><br>
11       <br><input type="submit" value="Submit">
12     </form>
13   `);
14 });
15
16 router.post('/submit-home', (req, res, next) => {
17   console.log("Post Request details", req.url, req.method, req.body);
18   res.redirect('/');
19 });
20
21 module.exports = router;
```



# 10.2 Express Router

EXPRESS Js

node > air-bnb > routes > `js` user.js > ...

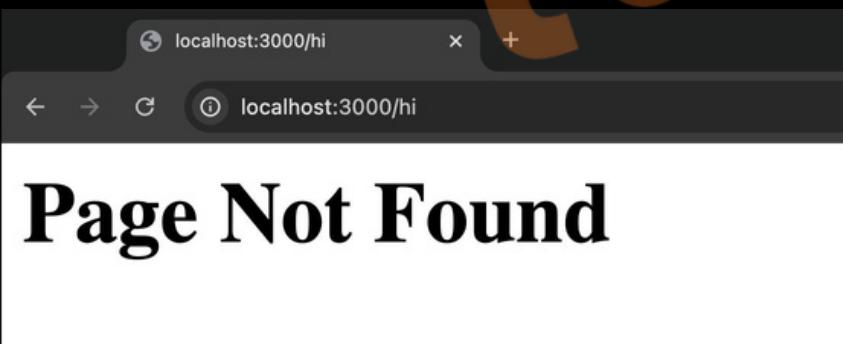
```
1 // External Modules
2 const express = require('express');
3 const userRouter = express.Router();
4
5 userRouter.use('/', (req, res, next) => {
6   console.log("Second Middleware", req.url, req.method);
7   res.send(`
8     <h2>Welcome to AirBnb</h2>
9     <a href="/add-home">Register Your Home on AirBnb</a>
10    `);
11 });
12
13 module.exports = userRouter;
```



# 10.3 Adding 404

EXPRESS Js

```
app.use(express.urlencoded({extended:true}));  
  
app.use(hostRouter);  
app.use(userRouter);  
  
app.use((req, res, next) => {  
  res.status(404).send('<h1>Page Not Found</h1>');  
});  
  
const PORT = 3000;  
app.listen(PORT, () => {  
  console.log(`Server running at http://localhost:${PORT}/`);  
});
```





# 10.4 Common Paths

EXPRESS Js

```
// GET path
router.get('/host/add-home', (req, res, next) => {
  res.send(`<h1>Register Your Home on AirBnB</h1>
<form action="/add-home" method="POST">
  <input type="text" name="houseName"
    placeholder="Enter your House Name"><br>
  <br><input type="submit" value="Submit">
</form>
`);
});
```

```
// POST path
router.post('/host/add-home', (req, res, next) => {
  console.log("Post Request details", req.url, req.
  method, req.body);
  res.redirect('/');
});
```

```
const app = express();
app.use(express.urlencoded({extended:true}));

app.use("/host", hostRouter);
app.use("/user", userRouter);

app.use((req, res, next) => {
  res.status(404).send('<h1>Page Not Found</h1>');
});
```



# 10.5 Adding HTML Files

EXPRESS Js

```
airbnb > views > add-home.html > ...
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
  initial-scale=1.0" />
  <title>airbnb</title>
</head>
<body>
  <header>
    <nav>
      <ul>
        <li><a href="/">Go to Home</a></li>
        <li><a href="/host/add-home">Add Home</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <h1>Register Your Home on AirBnB</h1>
    <form action="/host/add-home" method="POST">
      <input
        type="text"
        name="houseName"
        placeholder="Enter your House Name"
      /><br />
      <br /><input type="submit" value="Submit" />
    </form>
  </main>
</body>
</html>
```

```
airbnb > views > user.html > ...
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
  initial-scale=1.0" />
  <title>airbnb</title>
</head>
<body>
  <header>
    <nav>
      <ul>
        <li><a href="/">Go to Home</a></li>
        <li><a href="/host/add-home">Add Home</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <h2>Welcome to AirBnb</h2>
  </main>
</body>
</html>
```

```
> air-bnb
  > node_modules
  > routes
    host.js
    user.js
  > views
    404.html
    add-home.html
    user.html
    app.js
    package-lock.json
    package.json
```



# 10.6 Serving HTML Files

EXPRESS Js

air-bnb > routes > `host.js` > ...

```
// Core Modules
const path = require('path');

// External Modules
const express = require('express');

const router = express.Router();

// GET path
router.get('/add-home', (req, res, next) => {
  res.sendFile(path.join(__dirname, '../', 'views', 'add-home.html'));
});
```

complete CODING



# 10.7 Using File Helper

EXPRESS Js

airbnb > util > path.js > ...

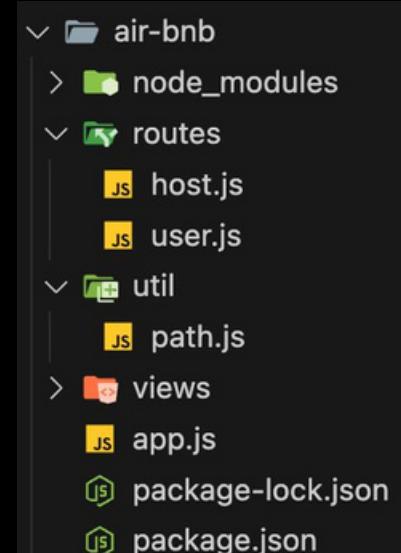
```
// Core Modules  
const path = require('path');
```

```
module.exports = path.dirname(require.main.filename);
```

```
// Local Modules
```

```
const rootDir = require('../util/path');
```

```
userRouter.get('/', (req, res, next) => {  
  console.log("Second Middleware", req.url, req.method);  
  res.sendFile(path.join(rootDir, 'views', 'user.html'));  
});
```





# Practise Set

Reuse the app from the **last assignment**

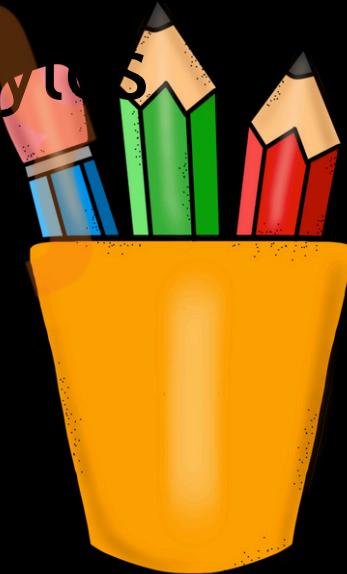
1. Parse the **body** of the **contact-us** request and log it to console.
2. Move the code to **separate local modules** and use the Express router to import and use them in app.js
3. Move all the **html** code to html files and serve them using the **file helper**.
4. Also add a **404** page for this app.





# Revision

1. Passing Requests
2. Express Router
3. Adding 404
4. Common Paths
5. Adding HTML Files
6. Serving HTML Files
7. Using File Helper



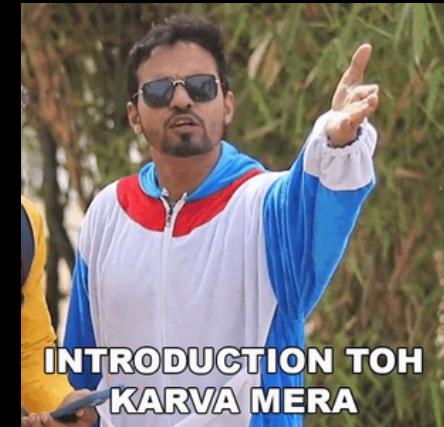


{ complete  
C O D I N G }



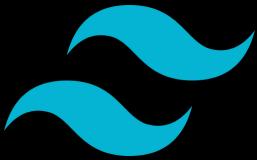
# 11. Styling using tailwindcss

1. Serving Static Files
2. Introduction to Tailwind CSS
3. Utility Classes
4. Installing Extension
5. Including Tailwind CSS
6. Installing Tailwind CSS
7. Using Tailwind CSS
8. Building Tailwind Automatically





# 11.1 Serving Static Files



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>airbnb</title>
    <link rel="stylesheet" href="airbnb.css">
  </head>
  <body>
    <header>
      <nav>
        <ul>
          <li><a href="/">Go to Home</a></li>
          <li><a href="/host/add-home">Add Home</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <h2>Welcome to AirBnb</h2>
    </main>
  </body>
</html>
```

```
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f5f5f5;
}

header {
  background-color: #ff5a5f;
  padding: 1rem 2rem;
  box-shadow: 0px 2px 8px rgba(0, 0, 0, 0.1);
}

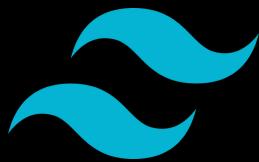
nav {
  display: flex;
  justify-content: space-between;
  align-items: center;
}

nav ul {
  list-style: none;
  margin: 0;
  padding: 0;
  display: flex;
}

nav li {
  margin-left: 20px;
}
```



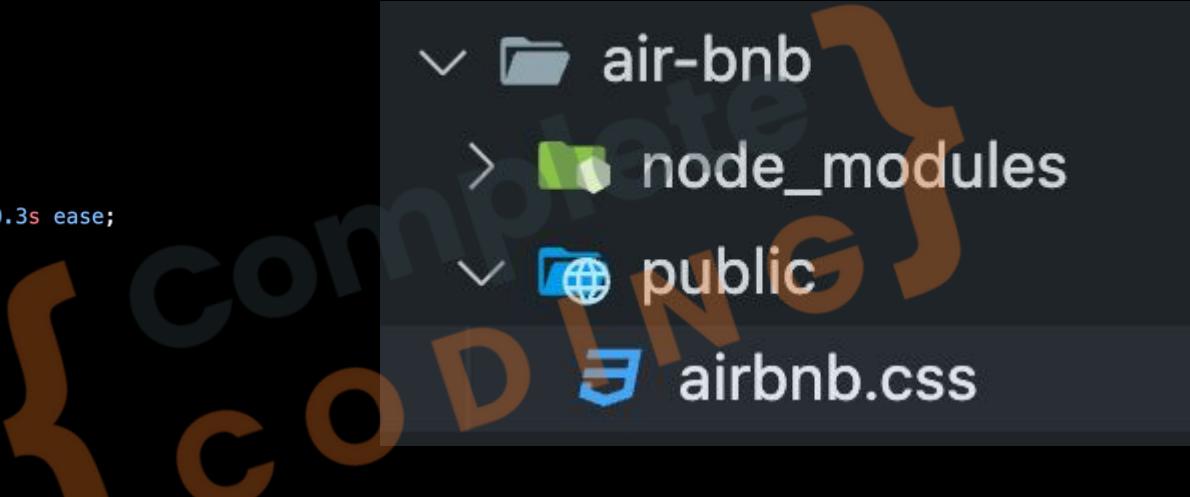
# 11.1 Serving Static Files



```
nav a {  
  text-decoration: none;  
  color: white;  
  font-weight: bold;  
  padding: 0.5rem 1rem;  
  border-radius: 4px;  
  transition: background-color 0.3s ease;  
}  
  
nav a:hover {  
  background-color: #e54e52;  
}
```

```
main {  
  text-align: center;  
  padding: 5rem 2rem;  
  background-color: white;  
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
  margin: 2rem auto;  
  max-width: 600px;  
  border-radius: 8px;  
}
```

```
h2 {  
  color: white;  
  font-size: 2rem;  
}
```



```
// Granting access to public folder  
app.use(express.static(path.join(rootDir, "public")));
```



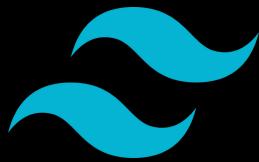
# 11.2 Introduction to Tailwind CSS

1. **Responsive:** Mobile-first design for all device sizes.
2. **Utility-First:** Provides low-level utility classes for building custom designs.
3. **Highly Customizable:** Easily extendable through a config file.
4. **Responsive Design:** Built-in responsive utilities (e.g., sm:, md:).
5. **No Predefined Components:** Focuses on building custom components.
6. **Purge CSS:** Removes unused styles in production for smaller files.
7. **Fast Development:** Style elements directly in markup for speed.





# 11.3 Utility Classes



```
1  /* Colors */
2  .text-primary { color: #007bff; }
3  .bg-primary { background-color: #007bff; }
4
5  /* Sizing */
6  .w-full { width: 100%; }
7  .h-full { height: 100%; }
8
9  /* Typography */
10 .text-center { text-align: center; }
11 .font-bold { font-weight: bold; }
12
13 /* Spacing */
14 .m-1 { margin: 0.25rem; }
15 .m-2 { margin: 0.5rem; }
16 .p-1 { padding: 0.25rem; }
17 .p-2 { padding: 0.5rem; }
```

```
19  /* Layout */
20 .d-flex { display: flex; }
21 .flex-col { flex-direction: column; }
22 .items-center { align-items: center; }
23 .justify-center { justify-content: center; }
24
25  /* Misc */
26 .rounded { border-radius: 0.25rem; }
27 .hidden { display: none; }
```



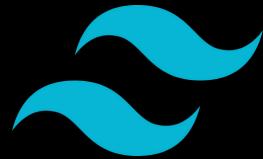
# 11.4 Installing Extension



```
class="bg-tl md:flex md:items-center md:  
.v class="fl bg-teal-50  
<h2 class="t bg-teal-100  
    Back End D bg-teal-200  
</h2>  
</div>  
div class="mt bg-teal-300  
<span class=" bg-teal-400  
    <button ty bg-teal-500  
        Edit bg-teal-600  
        </button> bg-teal-700  
    </span> bg-teal-800  
    <span class=" bg-teal-900  
        <button type="button" class="inline-t  
            Publish bg-transparent  
            </button>  
        </span>
```



# 11.4 Installing Extension



## Tailwind CSS IntelliSense v0.12.10

Tailwind Labs [tailwindcss.com](#) | 7,381,739 ⭐⭐⭐⭐⭐ (99)

Intelligent Tailwind CSS tooling for VS Code

[Install](#)  Auto Update

[DETAILS](#) [FEATURES](#) [CHANGELOG](#)

The screenshot shows a code editor in Visual Studio Code displaying a snippet of HTML with Tailwind CSS classes like "w-full", "flex", "items-center", "justify-between", "block", "p-6", "space-x-6", "mt-1", and "text-gray-500". A tooltip from the extension's UI highlights the "bg-teal-500" class with a list of other teal shades: "bg-teal-50", "bg-teal-100", "bg-teal-200", "bg-teal-300", "bg-teal-400", "bg-teal-500", "bg-teal-600", "bg-teal-700", "bg-teal-800", and "bg-teal-900". The background of the code editor has a faint watermark of a blue wavy icon.

tailwindcss INTELLISENSE

**Categories**

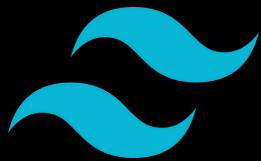
- Linters

**Resources**

- Marketplace
- Issues
- Repository
- License
- Tailwind Labs



# 11.5 Including Tailwind CSS



Installation

## Get started with Tailwind CSS

Tailwind CSS works by scanning all of your HTML files, JavaScript components, and any other templates for class names, generating the corresponding styles and then writing them to a static CSS file.

It's fast, flexible, and reliable — with zero-runtime.

### Installation

[Tailwind CLI](#) [Using PostCSS](#) [Framework Guides](#) [Play CDN](#)

Use the Play CDN to try Tailwind right in the browser without any build step. The Play CDN is designed for development purposes only, and is not the best choice for production.

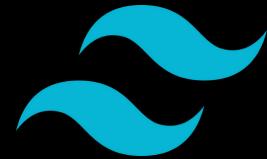
#### 1 Add the Play CDN script to your HTML

Add the Play CDN script tag to the <head> of your HTML file, and start using Tailwind's utility classes to style your content.

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<script src="https://cdn.tailwindcss.com"></script>
</head>
<body>
<h1 class="text-3xl font-bold underline">
Hello world!
</h1>
</body>
</html>
```



# 11.6 Installing Tailwind CSS



## 1. Install:

```
npm init -y
```

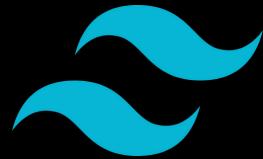
```
npm install -D tailwindcss postcss autoprefixer
```

## 2. Initialize Tailwind CSS Config

```
npx tailwindcss init
```



# 11.6 Installing Tailwind CSS

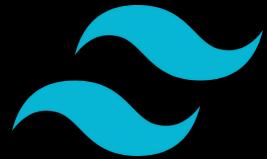


Configure Tailwind in the Configuration Files (tailwind.config.js)

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  ...,
  content: ['./views/*.html'],
  theme: {
    extend: {},
  },
  plugins: [],
}
```



# 11.6 Installing Tailwind CSS

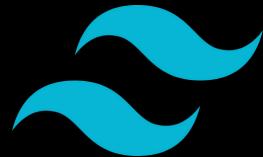


Add Tailwind Directives to views/input.css

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```



# 11.6 Installing Tailwind CSS



5. Include public/output.css into your index.html

6. Run Command

```
npx tailwindcss -i ./views/input.css -o ./public/output.css --watch
```

7. Declare Shortcut:

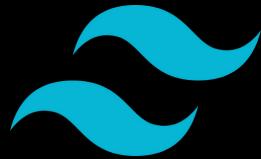
```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "tailwind": "npx tailwindcss -i ./src/input.css -o ./src/output.css --watch"  
},
```

8. Run Command

```
npm run tailwind
```



# 11.7 Using Tailwind CSS



```
<body class="■bg-gray-100 font-sans">  
  <header class="■bg-red-500 ■text-white p-4 shadow-md">  
    <nav class="container mx-auto flex justify-between items-center">  
      <ul class="flex space-x-4">  
        <li>  
          <a href="/" class="■hover:bg-red-400 py-2 px-4 rounded transition duration-300">Go to Home</a>  
        >  
      </li>  
      <li>  
        <a href="/host/add-home" class="■hover:bg-red-400 py-2 px-4 rounded transition duration-300">Add Home</a>  
        >  
      </li>  
    </ul>  
  </nav>  
</header>  
  <main class="container mx-auto ■bg-white shadow-lg rounded-lg p-8 mt-10 max-w-xl">  
    <h2 class="text-2xl ■text-gray-800 font-bold text-center mb-6">  
      | Welcome to Airbnb  
    </h2>  
  </main>  
</body>
```

Complete Coding

# 11.8 Building Tailwind Automatically <%

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "tailwind": "tailwindcss -i ./views/input.css -o ./public/output.css --watch",  
  "start": "nodemon app.js & npm run tailwind"  
},
```



# Practise Set

Style other page using  
Tailwind.

complete

CODING





# Revision

1. Setting up Static Files
  2. Introduction to Tailwind CSS
  3. Utility Classes
  4. Installing Extension
  5. Including Tailwind CSS
  6. Installing Tailwind CSS
  7. Using Tailwind CSS
- A large, semi-transparent watermark reading "COMPLETED CODING" is overlaid across the center of the slide.
- 
- A colorful illustration of a yellow pencil holder containing several pencils of different colors (pink, blue, green, red) and a sharpener.



{ complete  
C O D I N G }



# 12. Dynamic UI using EJS

- 1.Need of Dynamic UI
- 2.Sharing using Global Variables
- 3.What is EJS
- 4.Installing EJS
- 5.Using EJS Templates
- 6.Working with Partials



# 12.1 Need of Dynamic UI

● **Personalized Content:** Tailors responses based on user profiles, preferences, or behaviors to enhance user experience.

● **Dynamic Data Delivery:** Provides real-time information that updates with each request, such as live scores or stock prices.

● **Security and Access Control:** Delivers different content based on user authentication and authorization levels.

● **Localization and Internationalization:** Adjusts responses to accommodate different languages, cultures, or regional settings.

● **API Versatility:** Supports multiple client types (web, mobile, IoT) by providing appropriate data formats and structures.



# node 12.2 Sharing using Global Variables <%

```
const homes = [];  
  
// POST path  
router.post('/add-home', (req, res, next) => {  
  homes.push({houseName: req.body.houseName});  
  res.redirect('/');  
});
```

```
exports.hostRouter = router;  
exports.homes = homes;
```

---

```
const {homes} = require('../routes/host');  
  
userRouter.get('/', (req, res, next) => {  
  console.log('homes', homes);  
  res.sendFile(path.join(rootDir, 'views', 'user.html'));  
});
```

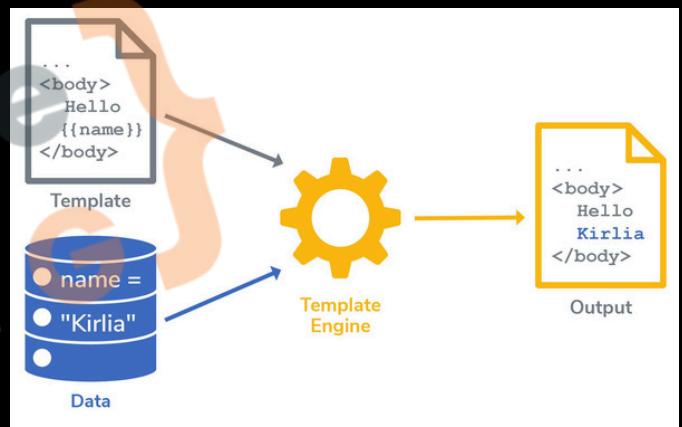
Variables are shared across users



# 12.3 What is EJS

<%

- **HTML with JS:** EJS lets you embed JavaScript code within HTML.
- **Simple Syntax:** Uses `<% %>` for control flow and `<%= %>` for output.
- **Easy to Learn:** Familiar to those who know HTML and JavaScript.
- **Template Reuse:** Supports **partials** for reusing code snippets.
- **Flexible Logic:** Allows full JavaScript expressions in templates.





## 12.4 Installing EJS



```
prashantjain@Prashants-Mac-mini air-bnb % npm install --save ejs  
added 6 packages, and audited 232 packages in 879ms
```

```
const app = express();  
  
app.set('view engine', 'ejs');  
app.set('views', 'views');
```



# 12.4 Installing EJS



## app.set(name, value)

Assigns setting `name` to `value`. You may store any value that you want, but certain names can be used to configure the behavior of the server. These special names are listed in the [app settings table](#).

Calling `app.set('foo', true)` for a Boolean property is the same as calling `app.enable('foo')`. Similarly, calling `app.set('foo', false)` for a Boolean property is the same as calling `app.disable('foo')`.

Retrieve the value of a setting with `app.get()`.

```
app.set('title', 'My Site')
app.get('title') // "My Site"
```

<code>views</code>	String or Array	A directory or an array of directories for the application's views. If an array, the views are looked up in the order they occur in the array.	<code>process.cwd() + '/views'</code>
<code>view cache</code>	Boolean	Enables view template compilation caching. <b>NOTE:</b> Sub-apps will not inherit the value of this setting in production (when 'NODE_ENV' is "production").	<code>true</code> in production, otherwise <code>undefined</code> .
<code>view engine</code>	String	The default engine extension to use when omitted. <b>NOTE:</b> Sub-apps will inherit the value of this setting.	N/A ( <code>undefined</code> )



# 12.5 Using EJS Templates

<%

```
✓ └─ views
    └─ 404.html
    └─ add-home.html
    └─ input.css
    └─ <% user.ejs
```

```
// Local Modules
const {homes} = require('../routes/host');

userRouter.get('/', (req, res, next) => {
  res.render('user', { homes: homes });
});
```

```
<main class="container mx-auto bg-white shadow-lg rounded-lg p-8 mt-10 max-w-xl">
  <h2 class="text-5xl text-gray-800 font-bold text-center mb-6">
    Available Homes
  </h2>
  <ul class="list-disc list-inside space-y-2">
    <% homes.forEach(function(home) { %>
      <li class="text-gray-700 hover:text-red-500 transition-colors duration-300 text-2xl">
        <%= home.houseName %>
      </li>
    <% }); %>
  </ul>
</main>
```



# 12.6 Working with Partials

<%

```
views
  partials
    head.ejs
    nav.ejs
    404.ejs
    add-home.ejs
    input.css
    user.ejs
```

```
1  <!DOCTYPE html>
2  <html lang="en">
3  |   <head>
4  |   |       <meta charset="UTF-8" />
5  |   |       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6  |   |       <title><%= title %></title>
7  |   |       <link rel="stylesheet" href="/output.css" />
8
```



# 12.6 Working with Partials

<%

```
1 <header class="■bg-red-500 ■text-white p-4 shadow-md">
2   <nav class="container mx-auto flex justify-between items-center">
3     <ul class="flex space-x-4">
4       <li>
5         <a
6           href="/"
7           class="■hover:bg-red-400 py-2 px-4 rounded transition duration-300"
8           >Go to Home</a>
9       >
10      <li>
11        <a
12          href="/host/add-home"
13          class="■hover:bg-red-400 py-2 px-4 rounded transition duration-300"
14          >Add Home</a>
15        >
16      </li>
17    </ul>
18  </nav>
19</header>
```

complete CODING

```
views
  partials
    head.ejs
    nav.ejs
    404.ejs
    add-home.ejs
    input.css
    user.ejs
```



# 12.6 Working with Partials

<%

airbnb > views > 404.ejs > ?

```
<%- include('partials/head') %>
</head>
<body class="■bg-gray-100 font-sans">
  <%- include('partials/nav') %>
  <main class="container mx-auto mt-20 text-center">
    <h2 class="text-6xl font-bold ■text-red-500 mb-4">404</h2>
    <p class="text-2xl □text-gray-700 mb-8">Oops! Page Not Found</p>
    <a href="/" class="■bg-red-500 ■text-white py-2 px-6 rounded-lg
      ■hover:bg-red-600 transition duration-300">Go Back Home</a>
  </main>
</body>
</html>
```

```
app.use((req, res, next) => {
  res.status(404).render('404', {title: 'Page Not Found'});
});
```



# 12.6 Working with Partials

<%

```
userRouter.get('/', (req, res, next) => {
  res.render('user', { homes: homes, title: 'Available Homes' });
});

<%- include('partials/head') %>
</head>
<body class="■bg-gray-100 font-sans">
  <%- include('partials/nav') %>
  <main class="container mx-auto ■bg-white shadow-lg rounded-lg p-8 mt-10 max-w-xl">
    <h2 class="text-5xl □text-gray-800 font-bold text-center mb-6">
      Available Homes
    </h2>
```



# Practise Set

Reuse the app from the last assignment

1. Add more fields in the add home page like price per night, location, rating, photo.
2. Design the home card to show all of this information
3. Make the selected tab active on top.





# Revision

1. N e D e d y n o a f m i c U I
  2. Sh a r i n g using Global Variables
  3. What is EJS
  4. Installing EJS
  5. Using EJS Templates
  6. Working with Partials
- A large, semi-transparent watermark reading "COMPLETED CODING" is overlaid across the center of the slide.





{ complete  
C O D I N G }



# 13. Model View Controller

1. Separation of Concerns
2. Adding First Controller
3. Adding All Controllers
4. Adding 404 Controller
5. Adding Models
6. Writing data to Files
7. Nodemon causing problems
8. Reading data from Files





# 13.1 Separation of Concerns

- **MVC stands for Model-View-Controller:** A software architectural pattern for developing user interfaces.

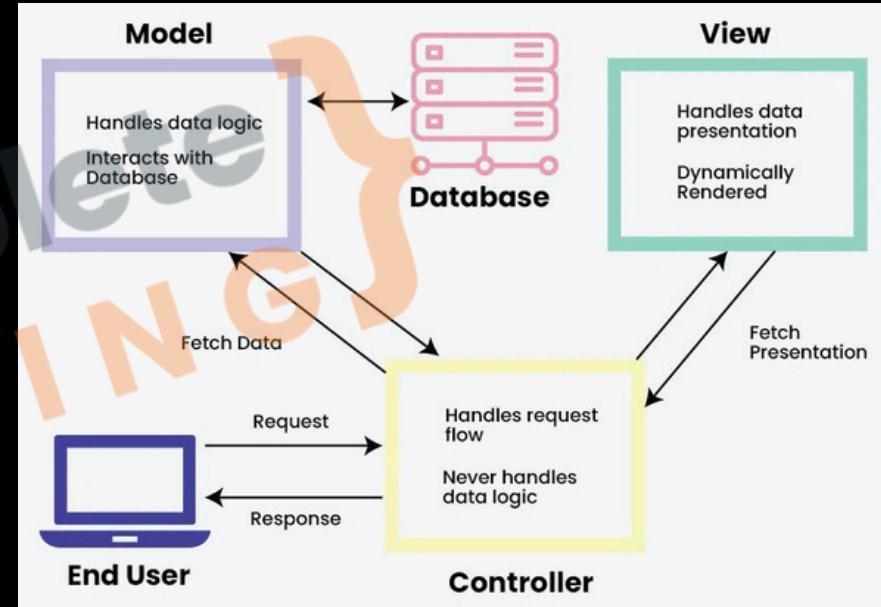
- **Model:** Manages the data and business logic of the application.

- **View:** Handles the display and presentation of data to the user.

- **Controller:** Processes user input, interacts with the Model, and updates the View accordingly.

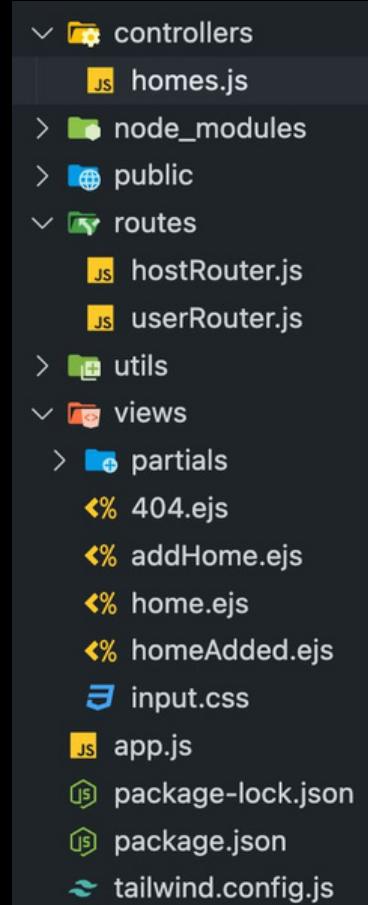
- Routes are a part of Controllers.

- **Purpose:** MVC separates concerns within an application, making it easier to manage and scale.





# 13.2 Adding First Controller



```
js homes.js x
node > Chapter 12 - Dynamic > controllers > homes.js > ...
3   exports.getAddHome = (req, res, next) => {
4     ...
5       res.render("addHome", {
6         pageTitle: "Add Home to airbnb",
7         currentPage: "addHome",
8       });
9   };

  complete } coding }
```

```
js hostRouter.js x
node > Chapter 12 - Dynamic > routes > hostRouter.js > ...
1 // External Module
2 const express = require('express');
3 const homesController = require('../controllers/homes');
4
5 const hostRouter = express.Router();
6
7 hostRouter.get("/add-home", homesController.getAddHome);
```



# 13.3 Adding All Controllers

js homes.js ×

```
node > Chapter 12 - Dynamic > controllers > js homes.js > ...  
1 const registeredHomes = [];  
2  
3 exports.getAddHome = (req, res, next) => {  
4   res.render("addHome", {  
5     pageTitle: "Add Home to airbnb",  
6     currentPage: "addHome",  
7   });  
8 };  
9  
10 exports.postAddHome = (req, res, next) => {  
11   console.log("Home Registration successful for:", req.body);  
12   registeredHomes.push(req.body);  
13   res.render("homeAdded", {  
14     pageTitle: "Home Added Successfully",  
15     currentPage: "homeAdded",  
16   });  
17 };  
18  
19 exports.getHomes = (req, res, next) => {  
20   console.log(registeredHomes);  
21   res.render("home", {  
22     registeredHomes: registeredHomes,  
23     pageTitle: "airbnb Home",  
24     currentPage: "Home",  
25   });  
26 };
```

Complete Coding



# 13.3 Adding All Controllers

JS hostRouter.js X

```
node > Chapter 12 - Dynamic > routes > JS hostRouter.js > ...
1 // External Module
2 const express = require('express');
3 const homesController = require('../controllers/homes');
4
5 const hostRouter = express.Router();
6
7 hostRouter.get("/add-home", homesController.getAddHome);
8
9 hostRouter.post("/add-home", homesController.postAddHome);
10
11 exports.hostRouter = hostRouter;
```



# 13.3 Adding All Controllers

JS userRouter.js X

node > Chapter 12 - Dynamic > routes > JS userRouter.js > ...

```
1 // External Module
2 const express = require('express');
3 const userRouter = express.Router();
4 const homesController = require('../controllers/homes');
5
6 userRouter.get("/", homesController.getHomes);
7
8 module.exports = userRouter;
```



# 13.4 Adding 404 Controller

```
controllers
  error.js
  homes.js
> node_modules
> public
< routes
  hostRouter.js
  userRouter.js
> utils
< views
  partials
    404.ejs
    addHome.ejs
    home.ejs
    homeAdded.ejs
    input.css
  app.js
  package-lock.json
  package.json
  tailwind.config.js
```

```
error.js  x
node > Chapter 12 - Dynamic > controllers > error.js > ...
1  exports.get404 = (req, res, next) => {
2    res.status(404).render('404', {pageTitle: 'Page Not Found', currentPage: '404'});
3  }

//Local Module
const userRouter = require("./routes/userRouter")
const {hostRouter} = require("./routes/hostRouter")
const rootDir = require("./utils/pathUtil");
const errorController = require("./controllers/error");

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');

app.use(express.urlencoded());
app.use(userRouter);
app.use("/host", hostRouter);

app.use(express.static(path.join(rootDir, 'public')))

app.use(errorController.get404);
```



# 13.5 Adding Models

The image shows a file structure on the left and a code editor window on the right.

**File Structure:**

- controllers
- models
- node\_modules
- public
- routes
- utils
- views
- app.js
- package-lock.json
- package.json
- tailwind.config.js

**Code Editor (home.js):**

```
home.js  x
node > Chapter 12 - Dynamic > models > home.js > ...
1 // fake database
2 const registeredHomes = [];
3
4 module.exports = class Home {
5   constructor(houseName, price, location, rating, photoUrl) {
6     this.houseName = houseName;
7     this.price = price;
8     this.location = location;
9     this.rating = rating;
10    this.photoUrl = photoUrl;
11  }
12
13  save() {
14    registeredHomes.push(this);
15  }
16
17  static fetchAll() {
18    return registeredHomes;
19  }
20}
```



# 13.5 Adding Models

node > Chapter 12 - Dynamic > controllers > homes.js > ...

```
1 const Home = require("../models/home");
2
3 exports.getAddHome = (req, res, next) => {
4   res.render("addHome", {
5     pageTitle: "Add Home to airbnb",
6     currentPage: "addHome",
7   });
8 }
9
10 exports.postAddHome = (req, res, next) => {
11   const { houseName, price, location, rating, photoUrl } = req.body;
12   const home = new Home(houseName, price, location, rating, photoUrl);
13   home.save();
14   res.render("homeAdded", {
15     pageTitle: "Home Added Successfully",
16     currentPage: "homeAdded",
17   });
18 }
19
20 exports.getHomes = (req, res, next) => {
21   const homes = Home.fetchAll();
22   res.render("home", {
23     registeredHomes: homes,
24     pageTitle: "airbnb Home",
25     currentPage: "Home",
26   });
27 }
```



# 13.6 Writing data to Files

```
> controllers
└── data
    └── homes.json
    └── models
        └── home.js
    > node_modules
    > public
    > routes
    └── utils
        └── pathUtil.js
    > views
        └── app.js
    └── package-lock.json
    └── package.json
    └── tailwind.config.js
```

```
1 const fs = require('fs');
2 const path = require('path');
3 const rootDir = require('../utils/pathUtil');
4
5 // fake database
6 const registeredHomes = [];
7
8 module.exports = class Home {
9     constructor(houseName, price, location, rating, photoUrl) {
10         this.houseName = houseName;
11         this.price = price;
12         this.location = location;
13         this.rating = rating;
14         this.photoUrl = photoUrl;
15     }
16
17     save() {
18         registeredHomes.push(this);
19         const filePath = path.join(rootDir, 'data', 'homes.json');
20         fs.writeFile(filePath, JSON.stringify(registeredHomes), (err) => {
21             console.log(err);
22         });
23     }
24
25     static fetchAll() {
26         return registeredHomes;
27     }
28 }
```



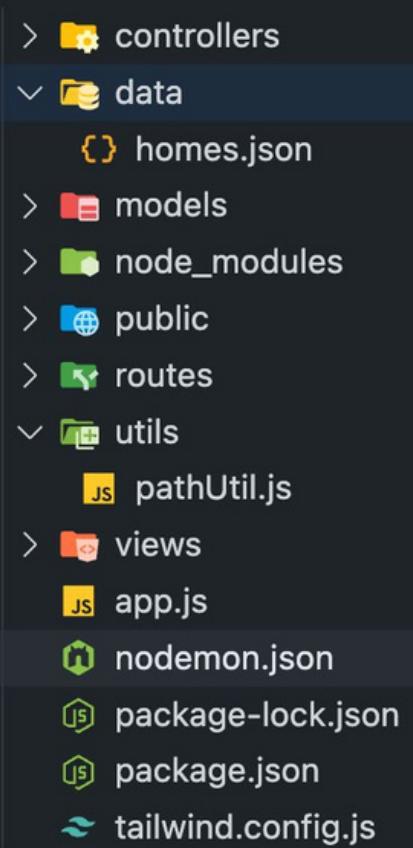
# 13.6 Writing data to Files

homes.json X

node > Chapter 12 - Dynamic > data > homes.json > ...

```
1 [  
2 {  
3   "houseName": "Utsav",  
4   "price": "999",  
5   "location": "Ghaziabad",  
6   "rating": "5",  
7   "photoUrl": "https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.airbnb.com%2Fstays%2Fdesign&psig=A0vVaw259nAPAVlXwRuraoxPS3u7&ust=1729404860631000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTC0iViaXlmYkDFQAAAAAdAAAAABAE"  
8 }]  
9 ]
```

# node 13.7 Nodemon causing problems



A screenshot of a code editor showing the contents of the `nodemon.json` file. The file is a JSON object with the following structure:

```
1  {
2    "watch": ["."],
3    "ext": "js,json,ejs",
4    "ignore": ["node_modules/", "data/"],
5    "exec": "node app.js"
6 }
```

The file is located at `node > Chapter 12 - Dynamic > nodemon.json`. A large watermark reading "COMPLETED" is overlaid across the center of the image.



# 13.8 Reading data from Files

```
static fetchAll() {  
  const filePath = path.join(rootDir, 'data', 'homes.json');  
  const fileContent = fs.readFile(filePath, (err, data) => {  
    if (err) {  
      return [];  
    }  
    return JSON.parse(data);  
  });  
}
```

TypeError: /Users/prashantjain/workspace/Test Project/node/Chapter 12 – Dynamic/views/home.ejs:10  
8| </h2>  
9| <div class="flex flex-wrap justify-center gap-6">  
>> 10| <% registeredHomes.forEach(home => { %>  
11| <div class="bg-white rounded-lg shadow-md overflow-hidden hover:shadow-lg transi  
12| 

Cannot read properties of undefined (reading 'forEach')  
at eval (eval at compile (/Users/prashantjain/workspace/Test Project/node/Chapter 12 – Dynamic  
at home (/Users/prashantjain/workspace/Test Project/node/Chapter 12 – Dynamic/node\_modules/ejs  
at tryHandleCache (/Users/prashantjain/workspace/Test Project/node/Chapter 12 – Dynamic/node\_m  
at events.renderFile [as original] (/Users/prashantjain/workspace/Test Project/node/Chapter 12 – Dynamic/node\_modules/ejs/lib/ejs.js:62:14)



# 13.8 Reading data from Files

```
save() {  
  this.fetchAll((registeredHomes) => {  
    registeredHomes.push(this);  
    const filePath = path.join(rootDir, 'data', 'homes.json');  
    fs.writeFile(filePath, JSON.stringify(registeredHomes), (err) => {  
      console.log(err);  
    });  
  });  
}  
  
static fetchAll(callback) {  
  const filePath = path.join(rootDir, 'data', 'homes.json');  
  fs.readFile(filePath, (err, data) => {  
    let homes = [];  
    if (!err) {  
      homes = JSON.parse(data);  
    }  
    callback(homes);  
  });  
}
```

```
exports.getHomes = (req, res, next) => {  
  Home.fetchAll((homes) => {  
    res.render("home", {  
      registeredHomes: homes,  
      pageTitle: "airbnb Home",  
      currentPage: "Home",  
    });  
  });  
};
```



# Practise Milestone

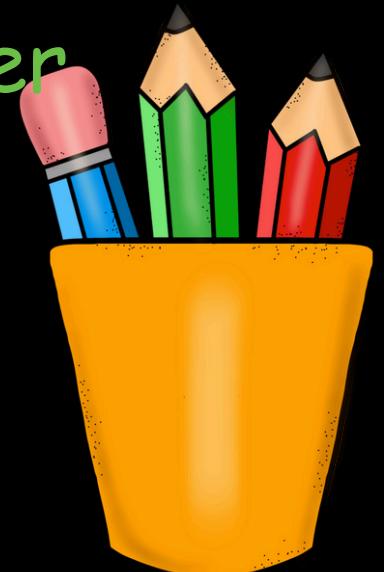
Take your airbnb forward:

1. Structure the views folder into host & store and move the respective view files there.  
Add more views to store like: home-list, home-detail, favourite-list, reserve, bookings. And to host view: edit-home, host-home-list
2. Improve the header with navigation to all pages.
3. Register all the new routes and add dummy views there.
4. Change controllers to store and host setup.
5. Add Edit and Delete buttons to the host-home-list view.
6. Keep the logic for Edit, Delete, Favourite pending.





# Revision

1. Separation of C
  2. Adding First Controller
  3. Adding All Controllers
  4. Adding 404 Controller
  5. Adding Models
  6. Writing data to Files
  7. Nodemon causing problems
  8. Reading data from Files
- 
- A yellow pencil holder containing four colored pencils: blue, pink, green, and red.



{ complete  
C O D I N G }

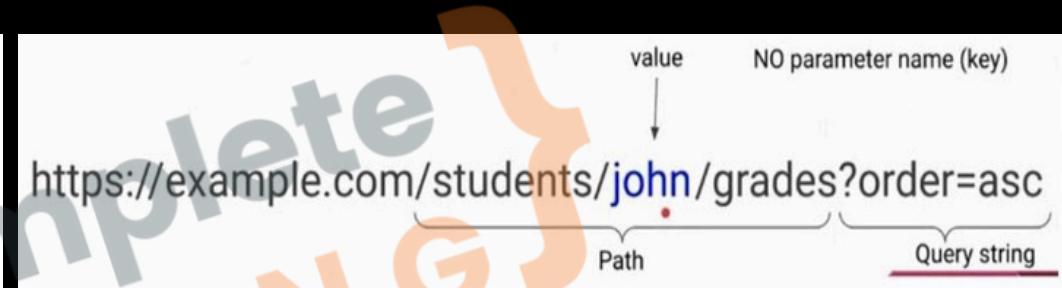
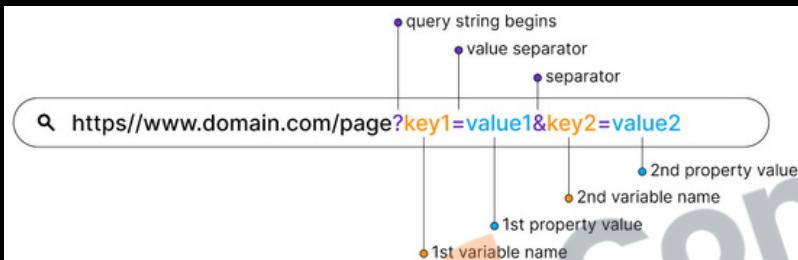
# 14. Dynamic Path & Model deep-dive

1. What are dynamic paths
2. Adding Home Details Page
3. Showing Real Home Data
4. Adding Favourites Feature
5. Adding Favourites Model
6. Edit-Home: Adding Feature Skeleton
7. Edit-Home: Showing Existing Data
8. Edit-Home: Handing Edit Request
9. Adding Delete Feature
10. Removing Home from Favourites





# 14.1 What are dynamic paths



- **Path parameters** are variables embedded directly in the URL

path to capture dynamic values, like `/users/:userId` where `:userId` is replaced with a specific user ID.

- **Query parameters** are key-value pairs appended to the URL after a ?, used to send additional data, like `/search?query=nodejs` where `query=nodejs` specifies the search term.



## 14.2 Adding Home Details Page

1. Add a **details button** in `/homes-list` to go to link path `/homes/:home-id`
2. Add a **random id** to each home in the **data file**.
3. Add a **random id** on home object before **saving** in the **home model**.
4. Add a **route** in the **store router** for `/homes/:home-id`
5. Add a **method** in **store controller** to get the `home-id` using `req.params` and **log it**, before **sending** out a dummy response with `home id`.

# 14.2 Adding Home Details Page

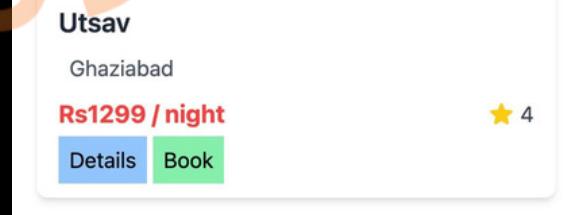
1.

```
<% home-list.ejs > <% homes.json > home.js  
views > store > <% home-list.ejs > ?  
1   <%- include('../partials/head') %>  
3   <body>  
4     <%- include('../partials/nav') %>  
5     <main class="container mx-auto bg-white shadow-lg rounded-lg p-8 mt-10 max-w-6xl">  
9       <div class="flex flex-wrap justify-center gap-6">  
25         <a href="/homes/<%= home.id %>" class="bg-blue-300 p-2">Details</a>  
26         <a href="#" class="bg-green-300 p-2">Book</a>  
27     </div>
```



2.

```
data > homes.json > ...
1  [{"id": "1", "houseName": "Utsav", "price": "1299", "location": "Ghaziabad", "rating": "4",  
  "photoUrl": "asdf"}, {"id": "2", "houseName": "Second House", "price": "999",  
  "location": "Ghaziabad", "rating": "4", "photoUrl": "asdf"}]
```



3.

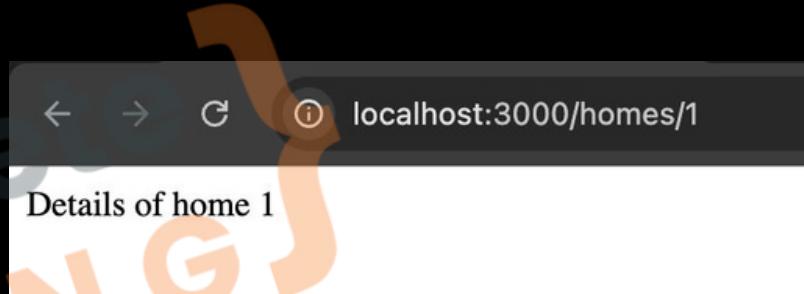
```
models > home.js > ...
  6 module.exports = class Home {
  7   ...
 14
 15   save() {
 16     this.id = Math.random().toString();
 17     Home.fetchAll((registeredHomes) => {
```



# 14.2 Adding Home Details Page

4.

```
storeRouter.js × storeController.js  
routes > storeRouter.js > ...  
8  storeRouter.get("/", homesController.getIndex);  
9  storeRouter.get("/homes", homesController.getHomes);  
10 // homes/1  
11 storeRouter.get("/homes/:homeId", homesController.getHome);  
12 storeRouter.get("/bookings", homesController.getBookings);
```



5.

```
storeRouter.js × storeController.js  
controllers > storeController.js > ...  
23 exports.getHome = (req, res, next) => {  
24   const homeId = req.params.homeId;  
25   console.log(homeId);  
26   res.send("Details of home " + homeId);  
27 };  
28
```



## 14.3 Showing Real Home Data

1. Add a static `findById` method in `Home` model that takes a `callback`.
2. Use this `findById` method in the controller to load home details and log them.
3. Now change the controller:
  - a. Redirect to `/homes` in case the home is not found, logging an error.
  - b. Rendering the `/home-detail` page with home data.
4. Create a `home-detail` page to use the entire page to show all home data.
5. Download from Github:
  - a. The styled `home-detail` file.
  - b. 10 image of homes that you can put locally and use.



# 14.3 Showing Real Home Data

1.

```
storeController.js  home.js  x  
models > home.js > ...
6  module.exports = class Home {
32
33    static findById(id, callback) {
34      this.fetchAll((homes) => {
35        const home = homes.find((home) => home.id === id);
36        callback(home);
37      });
38    }
39  };

```

```
Server running on address http://localhost:3000
{
  id: '1',
  houseName: 'Utsav',
  price: '1299',
  location: 'Ghaziabad',
  rating: '4',
  photoUrl: 'asdf'
}
```

2.

```
exports.getHome = (req, res, next) => {
  const homeId = req.params.homeId;
  Home.findById(homeId, (home) => {
    console.log(home);
  });
};
```



# 14.3 Showing Real Home Data

```
3. exports.getHome = (req, res, next) => {
    const homeId = req.params.homeId;
    Home.findById(homeId, (home) => {
        if (!home) {
            return res.redirect('/homes'); // Redirect if home not found
        }
        res.render("store/home-detail", {
            home: home,
            pageTitle: home.houseName,
            currentPage: "homes",
        });
    });
};
```



# 14.3 Showing Real Home Data

4.

```
views > store > home-detail.ejs > ↗?
1  <%- include('../partials/head') %>
2  </head>
3  <body>
4    <%- include('../partials/nav') %>
5    <main class="container mx-auto bg-white shadow-lg rounded-lg p-8 mt-10 max-w-6xl">
6      <h2 class="text-3xl text-red-500 font-bold text-center mb-6">
7        Details of <%= home.houseName %>
8      </h2>
9
10     <div class="grid grid-cols-1 md:grid-cols-2 gap-8">
11       <div class="rounded-lg overflow-hidden">
12         " class="w-full h-96 object-cover">
13       </div>
14
15       <div class="space-y-6">
16         <div class="border-b pb-4">
17           <h3 class="text-2xl font-semibold mb-2">Location</h3>
18           <p class="text-gray-600"><%= home.location %></p>
19         </div>
20
21         <div class="border-b pb-4">
22           <h3 class="text-2xl font-semibold mb-2">Price</h3>
23           <p class="text-green-600 text-xl font-bold">$<%= home.price %> / night</p>
24         </div>
25
26         <div class="border-b pb-4">
27           <h3 class="text-2xl font-semibold mb-2">Rating</h3>
28           <div class="flex items-center">
29             <span class="text-yellow-400 text-xl">*</span>
30             <span class="ml-2 text-lg"><%= home.rating %> / 5</span>
31           </div>
32         </div>
33
34         <div class="mt-8">
35           <form action="/favourite" method="post">
36             <button type="submit" class="bg-red-500 text-white px-8 py-3 rounded-lg hover:bg-
37               Add to Favorites
38             </button>
39           </form>
40         </div>
41       </div>
42     </div>
43   </main>
44 </body>
45 </html>
```

5.

Complete Coding

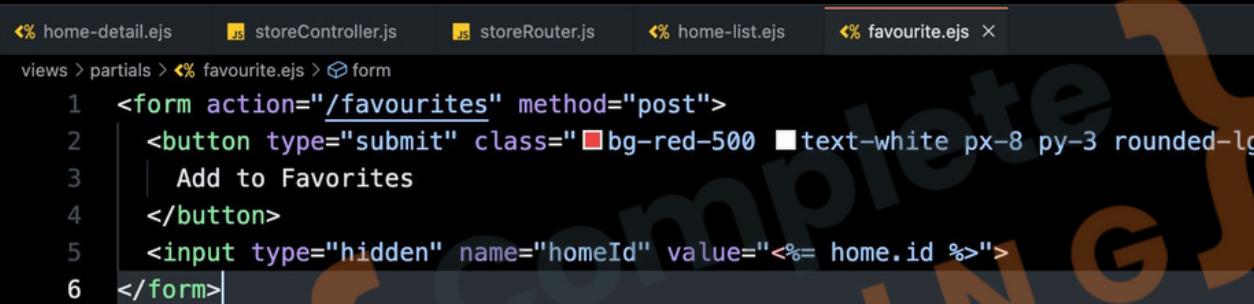
- images
  - house1.png
  - house2.png
  - house3.png
  - house4.png
  - house5.png
  - house6.png
  - house7.png
  - house8.png

# 14.4 Adding Favourites Feature

1. Create a partial with a form and a button that submits to /favourites path with a hidden input having home-id value.
2. Add this partial to home-detail page
3. Add this partial to home-list page.
4. Add router for handling POST request to /favourites path.
5. Add a method in store controller to add a home id as favourites, logging the id for now, before redirecting to /favourites path.

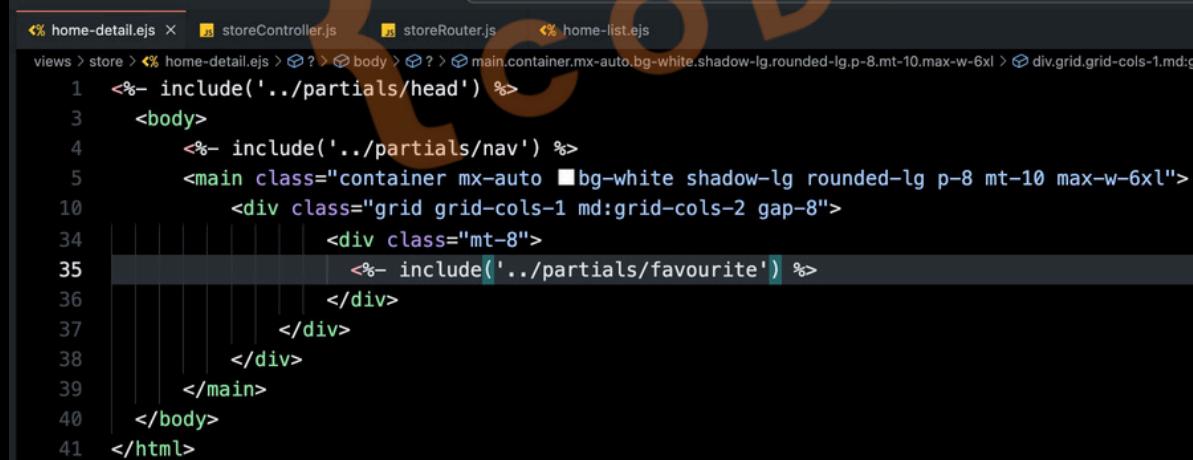
# node 14.4 Adding Favourites Feature

1.



```
<% home-detail.ejs      js storeController.js      js storeRouter.js      <% home-list.ejs      <% favourite.ejs > X
views > partials > <% favourite.ejs > ⚡ form
1   <form action="/favourites" method="post">
2     <button type="submit" class="bg-red-500 text-white px-8 py-3 rounded-lg"
3       Add to Favorites
4     </button>
5     <input type="hidden" name="homeId" value="<%= home.id %>">
6   </form>
```

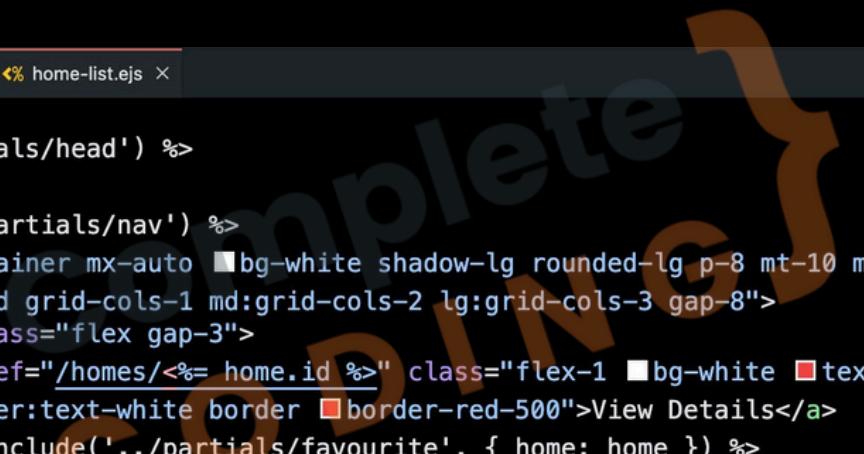
2.



```
<% home-detail.ejs > js storeController.js      js storeRouter.js      <% home-list.ejs
views > store > <% home-detail.ejs > ⚡ ? > ⚡ body > ⚡ ? > ⚡ main.container.mx-auto.bg-white.shadow-lg.rounded-lg.p-8.mt-10.max-w-6xl > ⚡ div.grid.grid-cols-1.md:gr
1   <%- include('../partials/head') %>
2   <body>
3     <%- include('../partials/nav') %>
4     <main class="container mx-auto bg-white shadow-lg rounded-lg p-8 mt-10 max-w-6xl">
5       <div class="grid grid-cols-1 md:grid-cols-2 gap-8">
6         <div class="mt-8">
7           <%- include('../partials/favourite') %>
8         </div>
9       </div>
10      </div>
11    </main>
12  </body>
13 </html>
```

# node 14.4 Adding Favourites Feature

3.

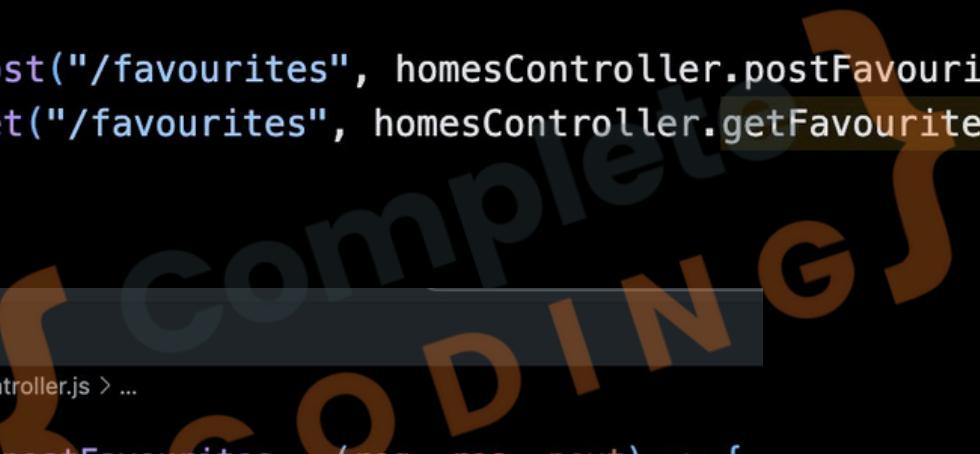


```
views > store > home-list.ejs > ↗ ?
1  <%- include('../partials/head') %>
3  <body>
4    <%- include('../partials/nav') %>
5    <main class="container mx-auto bg-white shadow-lg rounded-lg p-8 mt-10 max-w-6xl">
9      <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-8">
32        <div class="flex gap-3">
33          <a href="/homes/<%= home.id %>" class="flex-1 bg-white text-red-500 py-2 hover:text-white border border-red-500">View Details</a>
34          <%- include('../partials/favourite', { home: home }) %>
35        </div>
36      </div>
37    </div>
38    <% } %>
39  </div>
40 </main>
41 </body>
42 </html>
```

# node 14.4 Adding Favourites Feature

- storeRouter.post("/favourites", homesController.postFavourites);  
storeRouter.get("/favourites", homesController.getFavouriteList);

5.



```
JS storeController.js ×  
controllers > JS storeController.js > ...  
53  
54 exports.postFavourites = (req, res, next) => {  
55   const homeId = req.body.homeId;  
56   console.log(homeId);  
57   res.redirect('/favourites');  
58 };
```



# 14.5 Adding Favourites Model

1. Create a new Model to handle Favourites:
  - a. A static method `getFavourites` that reads the `favourites.json` file and
  - b. return the ids of all homes marked favourite.  
A static method `addFavourites` that adds the `home-id` to `favourites-id` array if not already there, then updates the file.
2. Use this model in `addFavourites` controller.
3. Use this model in `getFavourites` controller while also fetching details of
4. all homes from Homes Model.  
Change the UI of favourites page to show new content.



# 14.5 Adding Favourites Model

1.

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

const rootDir = require('../utils/pathUtil');
const favouritesPath = path.join(rootDir, "data", "favourites.json");

module.exports = class Favourites {
  static addFavourite(homeId) {
    this.getFavourites((favourites) => {
      if (favourites.includes(homeId)) {
        console.log("Home already in favourites");
      } else {
        favourites.push(homeId);
        fs.writeFile(favouritesPath, JSON.stringify(favourites), (err) => {
          console.log("File Writing Concluded", err);
        });
      }
    });
  }

  static getFavourites(callback) {
    fs.readFile(favouritesPath, (err, data) => {
      callback(!err ? JSON.parse(data) : []);
    });
  }
};
```



# 14.5 Adding Favourites Model

2.

```
controllers > JS storeController.js > ...
54
55 exports.postFavourites = (req, res, next) => {
56   const homeId = req.body.homeId;
57   Favourites.addFavourite(homeId);
58   res.redirect('/favourites');
59 };
```

```
data > {} favourites.json > ...
1  ["1", "2"]|
```



# 14.5 Adding Favourites Model

3.

```
exports.getFavouriteList = (req, res, next) => {
  Favourites.getFavourites((favourites) => {
    Home.fetchAll((registeredHomes) => {
      const favouritesWithDetails = favourites.map(homeId => registeredHomes.find(home => home.id === homeId));
      res.render("store/favourite-list", {
        favourites: favouritesWithDetails,
        pageTitle: "My Favourites",
        currentPage: "favourites",
      });
    });
  });
};
```



# 14.5 Adding Favourites Model

4.

```
favourites.js  storeController.js  homes.json  favourites.json  favourite-list.ejs  home-detail.ejs  home-list.ejs
views > store > favourite-list.ejs > ↗
1  <%- include('../partials/head') %>
2  </head>
3  <body>
4    <%= include('../partials/nav') %>
5    <main class="container mx-auto bg-white shadow-lg rounded-lg p-8 mt-10 max-w-6xl">
6      <h2 class="text-3xl text-red-500 font-bold text-center mb-6">
7        Here are your favourites
8      </h2>
9      <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-8">
10        <% favourites.forEach(home => { %>
11          <div class="bg-white rounded-lg shadow-md overflow-hidden hover:shadow-xl transition duration-300">
12            <div class="aspect-square">
13              " class="w-full h-full object-cover">
14            </div>
15            <div class="p-6">
16              <h3 class="text-2xl font-semibold text-gray-800 mb-3"><%- home.houseName %></h3>
17              <p class="text-gray-600 mb-4 flex items-center">
18                <svg xmlns="http://www.w3.org/2000/svg" class="h-5 w-5 mr-2" viewBox="0 0 20 20" fill="currentColor">
19                  <path fill-rule="evenodd" d="M5.05 4.05a7 7 0 11 9.9L10 18.9l-4.95-4.95a7 7 0 01-9.9zM10 11a2 2 0 100-4 2 2 0 000 4z" clip-rule="evenodd" />
20                </svg>
21                <%= home.location %>
22              </p>
23              <div class="flex justify-between items-center mb-4">
24                <span class="text-xl font-bold text-red-500"><%- home.price %> / night</span>
25                <div class="flex items-center bg-yellow-50 px-3 py-1 rounded-full">
26                  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="currentColor" class="w-5 h-5 text-yellow-400 mr-1">
27                    <path fill-rule="evenodd" d="M10.788 3.21c.448-1.077 1.976-1.077 2.424 0l2.082 5.007 5.404.433c1.164.093 1.636 1.545.749 2.305l-4.117 3.527 1.257 5.273c.271 1.136-.964 2.033-1.96 1.425L12 18.354 7.373 21.18c-.996 .608-2.231-.291.96-1.425l1.257-5.273-4.117-3.527c-.887-.76-.415-2.212.749-2.305L5.404-.433 2.082-5.006z" clip-rule="evenodd" />
28                </svg>
29                <span class="text-gray-700 font-medium"><%- home.rating %></span>
30              </div>
31            </div>
32            <div class="flex gap-3">
33              <a href="/homes/<%- home.id %>" class="flex-1 bg-white text-red-500 py-2 px-4 rounded-lg text-center hover:bg-red-500 hover:text-white border border-red-500">View Details</a>
34            </div>
35          </div>
36        </div>
37      <% } %>
38    </div>
39  </main>
40  </body>
41 </html>
```



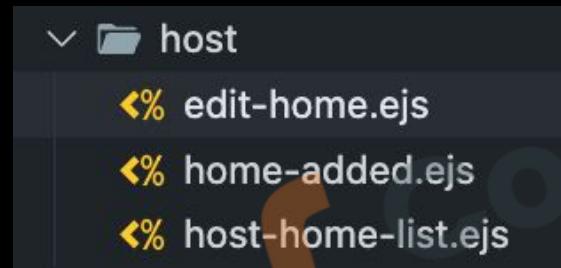
# 14.6 Edit-Home: Adding Feature Skeleton

1. Rename the `add-home.ejs` to `edit-home.ejs`
2. Fix it's usage in the `add-home controller`.
3. Change the path of edit button everywhere.
4. Add a new router for `GET /edit-home/:home-id`
5. Add a new controller method in host controller, optionally logging query param of `editing=true`, while passing editing flag to view.



## 14.6 Edit-Home: Adding Feature Skeleton

1.



2.

```
exports.getAddHome = (req, res, next) => {
  res.render("host/edit-home", {
    pageTitle: "Add Home to airbnb",
    currentPage: "addHome",
  });
};
```



# 14.7 Edit-Home: Showing Existing Data

3.

```
<%- include('../partials/head') %>
<body>
<%- include('../partials/nav') %>
<main class="container mx-auto bg-white shadow-lg rounded-lg p-8 mt-10 max-w-6xl">
  <div class="flex flex-wrap justify-center gap-6">
    </div>
    <a href="/host/edit-home/<%= home.id %>?editing=true" class="bg-blue-500 p-2">Edit</a>
    <button class="bg-red-500 p-2">Delete</button>
  </div>
  <% } ) %>
</div>
</main>
</body>
</html>
```



## 14.6 Edit-Home: Adding Feature Skeleton

```
4. hostRouter.get("/edit-home/:homeId", hostController.getEditHome);
```

```
5. exports.getEditHome = (req, res, next) => {
    const editing = req.query.editing === "true";
    const homeId = req.params.homeId;
    console.log(editing, homeId);
    res.render("host/edit-home", {
        pageTitle: "Edit Home",
        currentPage: "editHome",
        editing: editing,
        homeId: homeId,
    });
};
```

A screenshot of a terminal window. At the top, there's a status bar with a GitHub icon, a user icon, and the URL "localhost:3000/host/edit-home/1?editing=true". Below the status bar, the terminal shows the server's response: "Server running on address http://localhost:3000" followed by two lines of JSON output: "false 1" and "true 1".

```
C ① localhost:3000/host/edit-home/1?editing=true
Server running on address http://localhost:3000
false 1
true 1
```



# 14.7 Edit-Home: Showing Existing Data

1. Change the controller to now fetch the home details, if not found redirect to /host/host-home-list otherwise passing the data to view.
2. Change the edit-home.ejs to show dynamic content based on values:
  - a. Different submit path.
  - b. Different button text.
  - c. Pre-filled values if present.



# 14.7 Edit-Home: Showing Existing Data

```
1. exports.getEditHome = (req, res, next) => {
    const editing = req.query.editing === "true";
    const homeId = req.params.homeId;
    Home.findById(homeId, (home) => {
        if (!home) {
            return res.redirect("/host/host-home-list");
        }
        res.render("host/edit-home", {
            pageTitle: "Edit Home",
            currentPage: "editHome",
            editing: editing,
            home: home,
        });
    });
};
```



# 14.7 Edit-Home: Showing Existing Data

2.

```
<%- include('../partials/head') %>
</head>
<body>
  <%- include('../partials/nav') %>
  <main class="container mx-auto mt-8 p-8 bg-white rounded-lg shadow-md">
    <h1 class="text-3xl font-bold mb-6 text-center text-gray-800">Register Your Home on AirBnB</h1>
    <form action="/host/<%= editing ? 'edit-home' : 'add-home' %>" method="POST" class="max-w-md mx-auto">
      <input
        type="text"
        name="houseName"
        value="<%- home ? home.houseName : '' %>"
        placeholder="Enter your House Name"
        class="w-full px-4 py-2 mb-4 border rounded-md focus:outline-none focus:ring-2 focus:ring-red-500"/>
      <input
        type="text"
        name="price"
        value="<%- home ? home.price : '' %>"
        placeholder="Price Per Night"
        class="w-full px-4 py-2 mb-4 border rounded-md focus:outline-none focus:ring-2 focus:ring-red-500"/>
      <input
        type="text"
        name="location"
        value="<%- home ? home.location : '' %>"
        placeholder="Location"
        class="w-full px-4 py-2 mb-4 border rounded-md focus:outline-none focus:ring-2 focus:ring-red-500"/>
      <input
        type="text"
        name="rating"
        value="<%- home ? home.rating : '' %>"
        placeholder="Rating"
        class="w-full px-4 py-2 mb-4 border rounded-md focus:outline-none focus:ring-2 focus:ring-red-500"/>
      <input
        type="text"
        name="photoUrl"
        value="<%- home ? home.photoUrl : '' %>"
        placeholder="Enter Photo URL"
        class="w-full px-4 py-2 mb-4 border rounded-md focus:outline-none focus:ring-2 focus:ring-red-500"/>
      <input type="submit" value="<%- editing ? 'Update' : 'Add' %> Home" class="w-full bg-red-500 text-white py-2 transition duration-300"/>
    </form>
  </main>
</body>
</html>
```



# 14.8 Edit-Home: Handing Edit Request

1. Add a router for POST /edit-home
2. Add a controller for /edit-home, which creates a home model object
3. and saves it, before redirecting to /host/host-home-list
4. Add hidden id input field in the edit-home.ejs to get the id as part of POST request.

Change the save method in Model to handle creation of new as well as updation of existing Home.



# 14.8 Edit-Home: Handing Edit Request

1. hostRouter.post("/edit-home", hostController.postEditHome);

```
exports.postEditHome = (req, res, next) => {
  const { id, houseName, price, location, rating, photoUrl } = req.body;
  const home = new Home(houseName, price, location, rating, photoUrl);
  home.id = id;
  home.save();
  res.redirect("/host/host-home-list");
};
```

2.

3.

```
<%- include('../partials/head') %>
<body>
  <%- include('../partials/nav') %>
  <main class="container mx-auto mt-8 p-8 bg-white rounded-lg shadow-md">
    <h1 class="text-3xl font-bold mb-6 text-center text-gray-800">Register
    <form action="/host/<%= editing ? 'edit-home' : 'add-home' %>" method="Post"
      <input type="hidden" name="id" value="<%= home ? home.id : '' %>">
      <input
        type="text"
        name="houseName"
```



# 14.8 Edit-Home: Handing Edit Request

4.

```
module.exports = class Home {  
  save() {  
    Home.fetchAll((registeredHomes) => {  
      console.log(registeredHomes, this);  
      if (this.id) {  
        registeredHomes = registeredHomes.map((home) => {  
          console.log(home.id, this.id);  
          if (home.id === this.id) {  
            return this;  
          }  
          return home;  
        });  
      } else {  
        console.log("New Home");  
        this.id = Math.random().toString();  
        registeredHomes.push(this);  
      }  
      const homeDataPath = path.join(rootDir, "data", "homes.json");  
      fs.writeFile(homeDataPath, JSON.stringify(registeredHomes), (error) => {  
        console.log("File Writing Concluded", error);  
      });  
    });  
  }  
};
```



# 14.9 Adding Delete Feature

1. Surround the delete button with a form that submits to path /host/delete-home/:homelid
2. Add a route in the host routes.
3. Add a static delete method to the Home model that takes an id and deletes the home.
4. Add a method in host controller to handle the request, delete the home and redirect to /host/host-homes-list page.
5. Pending: deleted homes might still be in favourites list.



# 14.9 Adding Delete Feature

1. 

```
<a href="/host/edit-home/<%= home.id %>?editing=true" class="■bg-blue-500 p-2 rounded
■text-white ■hover:■bg-blue-600">Edit</a>
<form action="/host/delete-home/<%= home.id %>" method="POST" class="inline">
  <button type="submit" class="■bg-red-500 p-2 rounded ■text-white
■hover:■bg-red-600">Delete</button>
</form>
```
2. 

```
hostRouter.post("/delete-home/:homeId", hostController.postDeleteHome);
```
3. 

```
static deleteById(id, callback) {
  this.fetchAll((homes) => {
    const updatedHomes = homes.filter((home) => home.id !== id);
    fs.writeFile(path.join(rootDir, "data", "homes.json"), JSON.stringify(updatedHomes), callback);
  });
}
```



# 14.9 Adding Delete Feature

4. 

```
exports.postDeleteHome = (req, res, next) => {
  const homeId = req.params.homeId;
  Home.deleteById(homeId, (error) => {
    if (error) {
      console.log("Error Deleting Home", error);
    }
    res.redirect("/host/host-home-list");
  });
};
```

{ complete }

{ CODING }



# 14.10 Removing Home from Favourites

1. Add a static method to the favourites model to delete the home.
2. Add a form to the favourites-list in each home to path  
`/favourites/delete/:homelid`
3. Add a route for POST delete-favourites in the store routes.
4. Add a method in store controller to use the model's static method and then redirect to favourites-list.
5. Use this model method in home-delete to make sure any deleted home is also removed from favourites.

# 14.10 Removing Home from Favourites

1. 

```
static deleteFavourite(homeId, callback) {
  this.getFavourites((favourites) => {
    const updatedFavourites = favourites.filter((id) => id !== homeId);
    fs.writeFile(favouritesPath, JSON.stringify(updatedFavourites), callback);
  });
}
```
2. 

```
<form action="/favourites/delete/<%= home.id %>" method="POST">
  <button type="submit" class="■bg-red-500 p-2 rounded ■text-white
  ■hover:bg-red-600">Delete</button>
</form>
```
3. 

```
storeRouter.post("/favourites/delete/:homeId", storeController.deleteFavourite);
```



# 14.10 Removing Home from Favourites

4.

```
exports.deleteFavourite = (req, res, next) => {
  const homeId = req.params.homeId;
  Favourites.deleteFavourite(homeId, (error) => {
    if (error) {
      console.log("Error Deleting Favourite", error);
    }
    res.redirect('/favourites');
  });
};
```

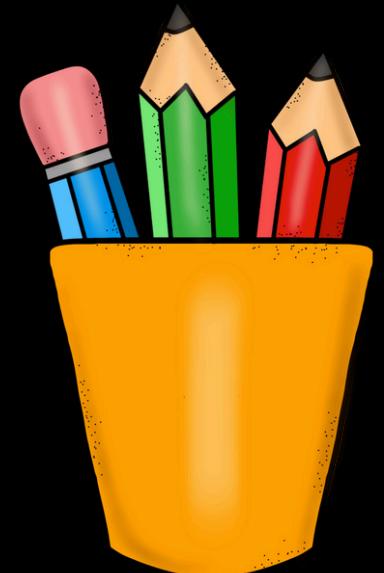
5.

```
static deleteById(id, callback) {
  this.fetchAll((homes) => {
    const updatedHomes = homes.filter((home) => home.id !== id);
    fs.writeFile(path.join(rootDir, "data", "homes.json"), JSON.stringify(updatedHomes), (error) => {
      Favourites.deleteFavourite(id, callback);
    });
  });
}
```



# Revision

1. What are dynamic paths
2. Adding Home Details Page
3. Showing Real Home Data
4. Adding Favourites Feature
5. Adding Favourites Model
6. Edit-Home: Adding Feature Skeleton
7. Edit-Home: Showing Existing Data
8. Edit-Home: Handing Edit Request
9. Adding Delete Feature
10. Removing Home from Favourites





{ complete  
C O D I N G }



# 15. Introduction to SQL

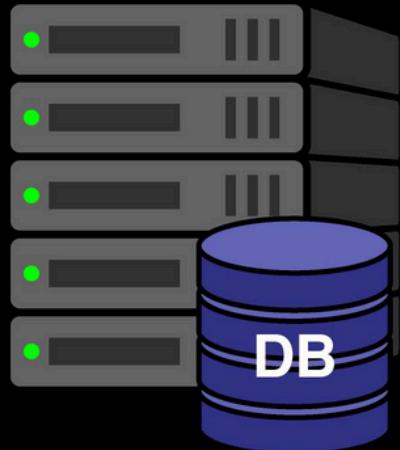
- 1.What is a DB (Database)
- 2.Introduction to SQL DB
- 3.Introduction to NoSQL DB
- 4.SQL vs NoSQL
- 5.Installing MySQL
6. Connecting App to DB
7. Creating homes Table
- 8.Querying homes in App
- 9.Adding DB in Models
- 10.Adding Home in Model
- 11.Implementing Model using Where



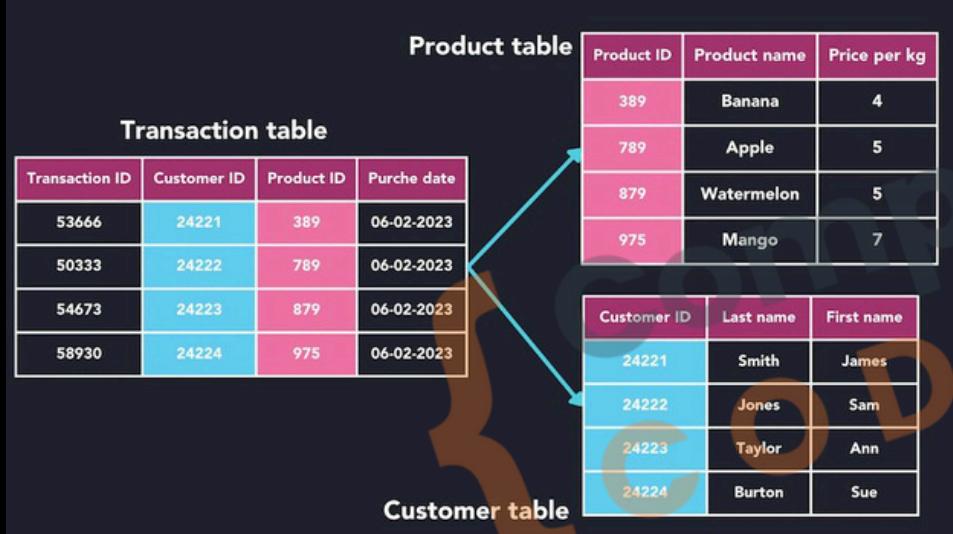


# 15.1 What is a DB (Database)

1. **Store Data:** Keep **large amounts of data** in a **structured format**.
2. **Enable Data Management:** Allow for adding, updating, and deleting data easily.
3. **Facilitate Quick Access:** Provide **fast retrieval** of data through queries.
4. **Ensure Data Integrity:** Maintain accuracy and consistency of data over time.
5. **Support Multiple Users:** Handle **concurrent access** by many users simultaneously.
6. **Secure Data:** Protect information through access controls and authentication.



# 15.2 Introduction to SQL DB



- **Vertical Scalability:** Typically scaled by increasing the resources of a single server (scaling up).
- **Relationships:** Tables can have multiple types of relationships.

- **Relational Model:** Organize data into tables with rows and columns.
- **Fixed Schema:** Require a predefined schema; the structure of the data must be known in advance.

# 15.2 Introduction to SQL DB MySQL™

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT age, country  
FROM Customers  
WHERE country = 'USA';
```

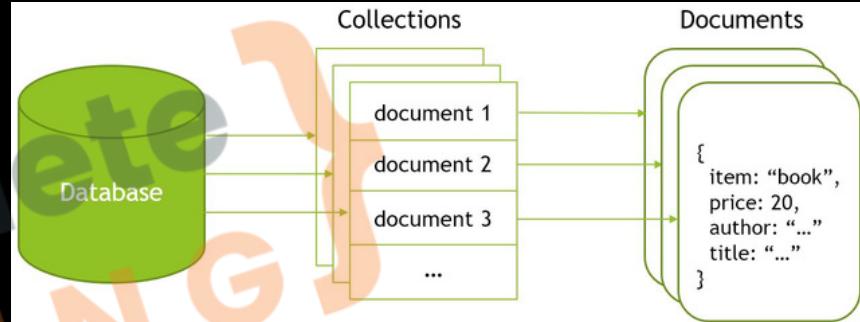
age	country
31	USA
22	USA

- Relational Model Use of SQL: Utilize SQL for querying and managing data, which is a standardized and widely-used language.
- ACID Compliance: Support transactions that are Atomic, Consistent, Isolated, and Durable.
- Complex Queries: Excel at handling complex queries and relationships between data.

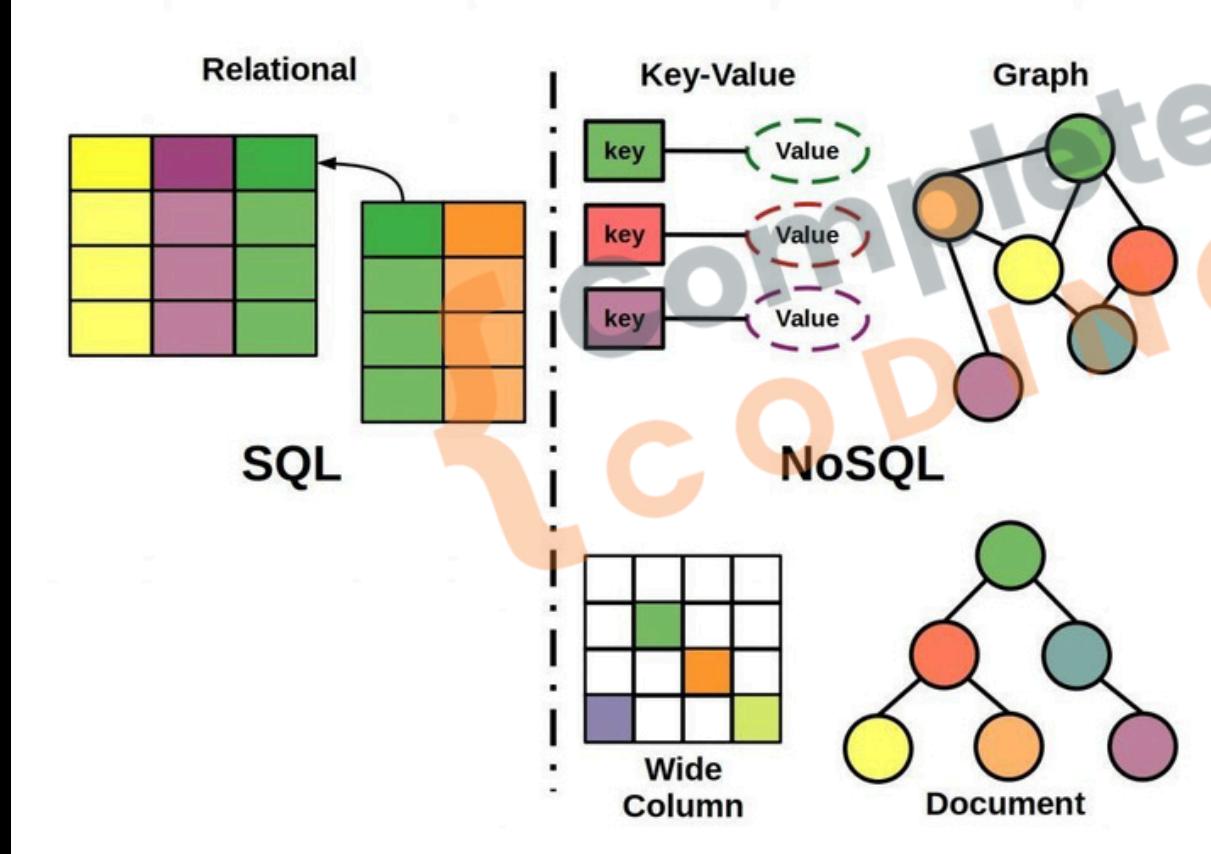


## 15.3 Introduction to NoSQL DB

- Flexible Schema: Allow for dynamic schemas, accommodating unstructured or semi-structured data without predefined structures.
- Duplicacy over Relations: Duplicates data across records
- (denormalization) to enhance performance and scalability, rather than relying on complex relationships and joins as in relational databases.
- Horizontal Scalability: Designed to scale out by adding more servers, handling large volumes of data efficiently.
- Performance: Optimized for high throughput and low latency, suitable for real-time applications.



# 15.4 SQL vs NoSQL





# 15.4 SQL vs NoSQL

Feature	SQL Databases	NoSQL Databases
Data Model	Relational (tables with rows and columns)	Non-relational (document, key-value, graph, etc.)
Schema	Fixed schema (predefined structure)	Flexible schema (dynamic structure)
Scalability	Vertically scalable (scale up)	Horizontally scalable (scale out)
Query Language	SQL (Structured Query Language)	Various query languages and APIs
ACID Compliance	Strong ACID compliance	Varies; often prioritizes performance over ACID
Use Cases	Structured data and complex queries	Unstructured data and real-time applications



# 15.5 Installing MySQL

MySQL™

dev.mysql.com/downloads/mysql/

## MySQL Community Downloads

MySQL Community Server

MySQL Enterprise Edition for Developers  
Free for learning, developing, and prototyping.  
[Download Now »](#)

General Availability (GA) Releases Archives

### MySQL Community Server 9.1.0 Innovation

Select Version:  
**9.1.0 Innovation**

Select Operating System:  
**Microsoft Windows**

Version	File Type	Size	Action
Windows (x86, 64-bit), MSI Installer	9.1.0	118.1M	<a href="#">Download</a>
(mysql-9.1.0-winx64.msi)			MD5: 7a26420bb3446eab56f389dba05a9718   <a href="#">Signature</a>
Windows (x86, 64-bit), ZIP Archive	9.1.0	288.4M	<a href="#">Download</a>
(mysql-9.1.0-winx64.zip)			MD5: 0a2333afc4ef87471bda89232697698e   <a href="#">Signature</a>
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite	9.1.0	822.6M	<a href="#">Download</a>
(mysql-9.1.0-winx64-debug-test.zip)			MD5: cb0327a504fc8d983f98c0cbf451a31d   <a href="#">Signature</a>

We suggest that you use the [MD5 checksums and GnuPG signatures](#) to verify the integrity of the packages you download.

Complete  
Coding



# 15.5 Installing MySQL

MySQL™

dev.mysql.com/downloads/workbench/ Guest (4)

## MySQL Community Downloads

MySQL Workbench

General Availability (GA) Releases Archives

### MySQL Workbench 8.0.38

Select Operating System:  
Microsoft Windows

Recommended Download:

**MySQL Installer** for Windows  
All MySQL Products. For All Windows Platforms. In One Package.

Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI

Go to Download Page >

Other Downloads:

Windows (x86, 64-bit), MSI Installer 8.0.38 41.7M Download  
(mysql-workbench-community-8.0.38-winx64.msi) MD5: 30ea58c9f40816566ac4cccd2f136f1e2 | Signature

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

ORACLE © 2024 Oracle  
Privacy / Do Not Sell My Info | Terms of Use | Trademark Policy | Cookie Preferences

Complete Coding



# 15.5 Installing MySQL

MySQL™

MySQL Workbench

## Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

[Browse Documentation >](#) [Read the Blog >](#) [Discuss on the Forums >](#)

MySQL Connections [+ ⚙](#)

Local instance 3306

- root
- localhost:3306

Filter connections

Ready.



# 15.5 Installing MySQL

MySQL™

A screenshot of the MySQL Workbench application. The title bar reads "Local instance 3306 - Warning - not supported". The main interface shows a toolbar with various icons, a left sidebar with sections for MANAGEMENT, INSTANCE, and PERFORMANCE, and a central query editor window. A large watermark reading "COMPILED CODING" is overlaid across the center. In the top right corner of the query editor, there is a tooltip message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." At the bottom of the application, a status bar displays "SQL Editor Opened".



# 15.5 Installing MySQL

MySQL™

MySQL Workbench

Local Instance 3306 - Warning - not supported

Administration Schemas Query 1 new\_schema - Schema Context Help Snippets

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

Object Info Session

No object selected

Schema Editor

Specify the name of the schema here. You can use any combination of ANSI letters, numbers underscore character for names that don't require quoting. For more flexibility you can use the Unicode Basic Multilingual Plane (BMP), but you will have to quote the name later when you refer to it.

Schema Name: new\_schema

The character set and its collation selected here will be used when no other charset/collation database object (it uses the DEFAULT value then). Setting DEFAULT here will make the schema inherit server defaults.

Character Set: Default Charset

Collation: Default Collation

Action Output

Time	Action	Response	Duration / Fetch Time

SQL Editor Opened.

A large orange watermark reading "CODING" is overlaid across the center of the MySQL Workbench interface.



# 15.6 Connecting App to DB



```
prashantjain@Prashants-Mac-mini Chapter 13 - MVC % npm install --save mysql2
added 12 packages, and audited 222 packages in 586ms
49 packages are looking for funding
  run `npm fund` for details
```

```
utils > database.js > ...
1  const mysql = require("mysql2");
2
3  const pool = mysql.createPool({
4    host: "localhost",
5    user: "root",
6    password: "CompleteCoding@01",
7    database: "airbnb",
8  });
9
10 module.exports = pool.promise();
```



# 15.7 Creating homes Table

Query 1    homes - Table    Schema: airbnb

Name: homes

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
id	INT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<click to edit>				
name	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<click to edit>
price	DOUBLE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<click to edit>
description	LONGTEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<click to edit>
imageUrl	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<click to edit>
location	VARCHAR(45)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<click to edit>
rating	DOUBLE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<click to edit>
<click to edit>										

Column details "

Column Name:

Datatype:

Charset/Collation: Default Charset    Default Collation

Comments:

Expression:

Storage:  VIRTUAL     STORED

Primary Key     Not NULL     Unique

Binary     Unsigned     ZeroFill

Auto Increment     Generated

Columns    Indexes    Foreign Keys    Triggers    Partitioning    Options    Apply    Revert



## 15.7 Creating homes Table

```
- CREATE TABLE `airbnb`.`homes` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  `price` DOUBLE NOT NULL,
  `description` LONGTEXT NOT NULL,
  `imageUrl` VARCHAR(255) NOT NULL,
  `location` VARCHAR(45) NOT NULL,
  `rating` DOUBLE NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE);
```



# 15.7 Creating homes Table

The screenshot shows the MySQL Workbench interface. The left sidebar displays the database schema with the 'airbnb' database selected. Under 'Tables', the 'homes' table is highlighted. The main pane shows a query editor with the SQL command: `SELECT * FROM airbnb.homes;`. Below the query editor is a 'Result Grid' showing the structure of the 'homes' table:

	<b>id</b>	<b>name</b>	<b>price</b>	<b>description</b>	<b>imageUrl</b>	<b>location</b>	<b>rating</b>
1	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	NULL	NULL	NULL	NULL	NULL	NULL	NULL
6	NULL	NULL	NULL	NULL	NULL	NULL	NULL
7	NULL	NULL	NULL	NULL	NULL	NULL	NULL
8	NULL	NULL	NULL	NULL	NULL	NULL	NULL
9	NULL	NULL	NULL	NULL	NULL	NULL	NULL
10	NULL	NULL	NULL	NULL	NULL	NULL	NULL

A large watermark reading 'COMPLETED CODING' is overlaid across the interface.

The dialog box is titled 'Apply SQL Script to Database'. It contains the heading 'Review the SQL Script to be Applied on the Database' and the sub-instruction 'Please review the following SQL script that will be applied to the database.' Below this, it says 'Note that once applied, these statements may not be revertible without losing some of the data.' and 'You can also manually change the SQL statements before execution.'

```
1  INSERT INTO `airbnb`.`homes` (`name`, `price`, `description`, `imageUrl`, `location`) VALUES ('Utsav', '999', 'the best holiday home', '/images/house1.png', 'delhi');
2
```



# 15.8 Querying homes in App

```
const db = require("./utils/database");

db.execute("SELECT * FROM homes").then(([rows, fields]) => {
  console.log(rows);
  console.log(fields);
}).catch((error) => {
  console.log("Error Fetching Homes", error);
});
```

```
Server running on address http://localhost:3000
[
  {
    id: 1,
    name: 'Utsav',
    price: 999,
    description: 'the best holiday home',
    imageUrl: '/images/house1.png',
    location: 'delhi',
    rating: 4.5
  }
]
```
`id` INT UNSIGNED NOT NULL PRIMARY KEY UNIQUE_KEY AUTO_INCREMENT,
`name` VARCHAR(255) NOT NULL,
`price` DOUBLE NOT NULL,
`description` LONGTEXT NOT NULL,
`imageUrl` VARCHAR(255) NOT NULL,
`location` VARCHAR(45) NOT NULL,
`rating` DOUBLE NOT NULL
]
```



# 15.9 Adding DB in Models

1. Remove the **test code** from `app.js`
2. Change the `Home.js` model file to remove all code related to file operations.
3. Import the DB from the `utils`.
4. Change `photoUrl` to `imageUrl` and `houseName` to `name` in the entire project.
5. Implement `fetchAll`:
  - a. Using the query we used while testing.
  - b. `fetchAll` will not take a callback but return a promise.
6. Go to `StoreController` and use the promise to get the data here.
7. Fix all the usages of `fetchAll`.



# 15.9 Adding DB in Models

2,3.

```
const db = require("../utils/database");
const Favourites = require("./favourites");

module.exports = class Home {
  constructor(houseName, price, location, rating, photoUrl) {
    this.houseName = houseName;
    this.price = price;
    this.location = location;
    this.rating = rating;
    this.photoUrl = photoUrl;
  }

  save() { ↵ tab
  }

  static fetchAll() {
  }

  static findById(id) {
  }

  static deleteById(id) {
  }
};
```



# 15.9 Adding DB in Models

```
5. static fetchAll() {  
  |   return db.execute("SELECT * FROM homes");  
}  
  
6. exports.getHomes = (req, res, next) => {  
  Home.fetchAll()  
    .then(([rows, fields]) => {  
      res.render("store/home-list", {  
        registeredHomes: rows,  
        pageTitle: "Homes List",  
        currentPage: "Home",  
      });  
    })  
    .catch((error) => {  
      console.log("Error Fetching Homes", error);  
    });  
};
```



# 15.9 Adding DB in Models

7.

```
exports.getIndex = (req, res, next) => {
  Home.fetchAll()
    .then(([rows, fields]) => {
      res.render("store/index", {
        registeredHomes: rows,
        pageTitle: "airbnb Home",
        currentPage: "index",
      });
    })
    .catch((error) => {
      console.log("Error Fetching Homes", error);
    });
};

exports.getHostHomes = (req, res, next) => {
  Home.fetchAll().then(([rows, fields]) => {
    res.render("host/host-home-list", {
      registeredHomes: rows,
      pageTitle: "Host Homes",
      currentPage: "hostHomes",
    });
  }).catch((error) => {
    console.log("Error Fetching Homes", error);
  });
};
```

```
exports.getFavouriteList = (req, res, next) => {
  Favourites.getFavourites((favourites) => {
    Home.fetchAll().then(([registeredHomes, fields]) => {
      const favouritesWithDetails = favourites.map((homeId) =>
        registeredHomes.find((home) => home.id === homeId)
      );
      res.render("store/favourite-list", {
        favourites: favouritesWithDetails,
        pageTitle: "My Favourites",
        currentPage: "favourites",
      });
    }).catch((error) => {
      console.log("Error Fetching Homes", error);
    });
  });
};
```



# 15.10 Adding Home in Model

1. Add the **description** field in home. Change constructor and usage.
2. Make changes in UI to input and show it everywhere.
3. Implement the **save** method using the **insert** query.
4. Change the usages of save method to use the promise.



# 15.10 Adding Home in Model

1.

```
module.exports = class Home {  
    constructor(name, description, price, location, rating, imageUrl) {  
        this.name = name;  
        this.description = description;  
        this.price = price;  
        this.location = location;  
        this.rating = rating;  
        this.imageUrl = imageUrl;  
    }  
  
    exports.postAddHome = (req, res, next) => {  
        const { name, description, price, location, rating, imageUrl } = req.body;  
        const home = new Home(name, description, price, location, rating, imageUrl);  
  
        exports.postEditHome = (req, res, next) => {  
            const { id, name, description, price, location, rating, imageUrl } = req.body;  
            const home = new Home(name, description, price, location, rating, imageUrl);  
        }  
    }  
}
```



# 15.10 Adding Home in Model

2.

```
<input  
  type="text"  
  name="description"  
  value="<%= home ? home.description : '' %>"  
  placeholder="Enter your House Description"  
  class="w-full px-4 py-2 mb-4 border rounded-md focus:outline-none focus:ring-2  
  focus:ring-red-500"/>  
  
<div class="border-b pb-4">  
  <h3 class="text-2xl font-semibold mb-2">Description</h3>  
  <p class="text-gray-600"><%= home.description %></p>  
</div>  
  
<div class="border-b pb-4">  
  <h3 class="text-2xl font-semibold mb-2">Location</h3>  
  <p class="text-gray-600"><%= home.location %></p>  
</div>
```



# 15.10 Adding Home in Model

3.

```
save() {
  return db.execute(
    "INSERT INTO homes (name, price, location, rating, imageUrl) VALUES (?, ?, ?, ?, ?)",
    [this.name, this.price, this.location, this.rating, this.imageUrl]
  );
}
```

4.

```
exports.postAddHome = (req, res, next) => {
  const { name, description, price, location, rating, imageUrl } = req.body;
  const home = new Home(name, description, price, location, rating, imageUrl);
  home.save().then(() => {
    res.render("host/home-added", {
      pageTitle: "Home Added Successfully",
      currentPage: "homeAdded",
    });
  }).catch((error) => {
    console.log("Error Adding Home", error);
  });
};


```

```
exports.postEditHome = (req, res, next) => {
  const { id, name, description, price, location, rating, imageUrl } = req.body;
  const home = new Home(name, description, price, location, rating, imageUrl);
  home.id = id;
  home.save().then(() => {
    res.redirect("/host/host-home-list");
  }).catch((error) => {
    console.log("Error Editing Home", error);
  });
};
```



# 15.11 Implementing Model using Where

```
static findById(id) {  
  return db.execute("SELECT * FROM homes WHERE id = ?", [id]);  
}  
  
static deleteById(id) {  
  return db.execute("DELETE FROM homes WHERE id = ?", [id]);  
}
```

```
exports.postDeleteHome = (req, res, next) => {  
  const homeId = req.params.homeId;  
  Home.deleteById(homeId).then(() => {  
    res.redirect("/host/host-home-list");  
  }).catch((error) => {  
    console.log("Error Deleting Home", error);  
  });  
};
```

```
exports.getHome = (req, res, next) => {  
  const homeId = req.params.homeId;  
  Home.findById(homeId).then(([rows]) => {  
    const home = rows[0];  
    if (!home) {  
      return res.redirect("/homes");  
    }  
    res.render("store/home-detail", {  
      home: home,  
      pageTitle: home.name,  
      currentPage: "homes",  
    });  
  }).catch((error) => {  
    console.log("Error Fetching Home", error);  
  });  
};
```



# Practise Milestone

Take your **airbnb** forward:

- Change the save method to support both edit and insert functionality.

{ complete  
CODING }





# Practise Milestone (Solution)

```
save() {  
  if (this.id) {  
    // Update existing home  
    return airbnbDb.execute(  
      `UPDATE homes SET houseName=?, price=?, location=?, rating=?, photoUrl=?, description=? WHERE id=?`,  
      [this.houseName, this.price, this.location, this.rating, this.photoUrl, this.description, this.id]  
    );  
  } else {  
    //return airbnbDb.execute(`INSERT INTO homes (houseName, price, location, rating, photoUrl,  
    description) VALUES ('${this.houseName}', ${this.price}, '${this.location}', ${this.rating}, '${this.  
    photoUrl}', '${this.description}')`);  
    // Insert new home  
    return airbnbDb.execute(  
      `INSERT INTO homes (houseName, price, location, rating, photoUrl, description) VALUES  
      (?, ?, ?, ?, ?, ?)`,  
      [this.houseName, this.price, this.location, this.rating, this.photoUrl, this.description]  
    );  
  }  
}
```

Complete Coding





# Practise Milestone (Solution)

```
exports.postEditHome = (req, res, next) => {
  const { id, houseName, price, location, rating, photoUrl, description } =
    req.body;
  const newHome = new Home(
    houseName,
    price,
    location,
    rating,
    photoUrl,
    description
  );
  newHome.id = id;
  newHome
    .save()
    .then(() => {
      res.redirect("/host/host-homes");
    })
    .catch((error) => {
      console.log("Error while updating home", error);
    });
};
```

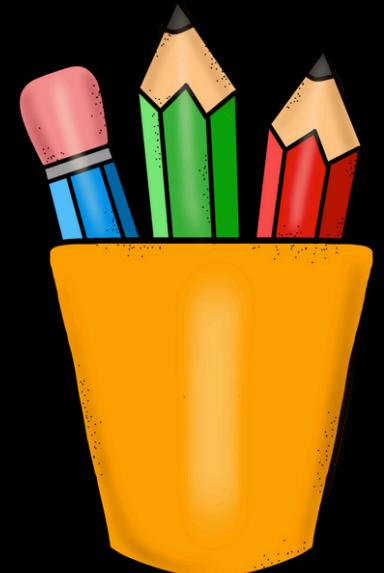
{ Complete Coding }





# Revision

1. What is a DB (Database)
2. Introduction to SQL DB
3. Introduction to NoSQL DB
4. SQL vs NoSQL
5. Installing MySQL
6. Connecting App to DB
7. Creating homes Table
8. Querying homes in App
9. Adding DB in Models
10. Adding Home in Model
11. Implementing Model using Where





{ complete  
C O D I N G }



# 16. Introduction to MongoDB

- 1.What is MongoDB
- 2.Setting up MongoDB
- 3.Installing MongoDB Driver
- 4.Creating MongoDB Connection
- 5.Saving a Home
- 6.Install MongoDB Compass
- 7.Install MongoDB for VSCode
- 8.Fetching all Homes
- 9.Fetching one Home
10. Supporting Edit & Delete
- 11.Adding MongoDB to Favourite

```
{  
  "_id": "4f5b5c85-d8d3-4f58-8acf-3f5e5e4e59ea",  
  "Items": [  
    {  
      "ProductId": 1,  
      "ProductName": "Elden Ring",  
      "Price": "49.97",  
      "Quantity": 1  
    },  
    {  
      "ProductId": 2,  
      "ProductName": "FIFA 23",  
      "Price": "69.97",  
      "Quantity": 1  
    }  
  ]  
}
```





# 16.1 What is MongoDB

1. MongoDB is the product and the company that builds it.
2. The name comes from the work **Humongous**.
3. **NoSQL Document Database:** Stores data in flexible, JSON-like documents.
4. **Dynamic Schema:** Allows fields to vary across documents without predefined schemas.
5. **High Performance:** Optimized for fast read and write operations.
6. **Scalability:** Supports horizontal scaling through sharding.
7. **High Availability:** Provides replication with automatic failover.
8. **Rich Query Capabilities:** Offers powerful querying, indexing, and aggregation.
9. **Geospatial and Text Search:** Includes support for location-based and full-text queries.
10. **Cross-Platform Compatibility:** Works with various operating systems and programming languages.
11. **Easy Integration:** Integrates smoothly with modern development stacks.

```
{  
  "_id": "4f5b5c85-d8d3-4f58-8acf-3f5e5e4e59ea",  
  "Items": [  
    {  
      "ProductId": 1,  
      "ProductName": "Elden Ring",  
      "Price": "49.97",  
      "Quantity": 1  
    },  
    {  
      "ProductId": 2,  
      "ProductName": "FIFA 23",  
      "Price": "69.97",  
      "Quantity": 1  
    }  
  ]  
}
```





# 16.2 Setting up MongoDB

## MongoDB Community Server

### Download

The Community version of our distributed database offers a flexible document data model along with support for ad-hoc queries, secondary indexing, and real-time aggregations to provide powerful ways to access and analyze your data.

The database is also offered as a fully-managed service with [MongoDB Atlas](#). Get access to advanced functionality such as auto-scaling, serverless instances, full-text search, and data distribution across regions and clouds. Deploy in minutes on AWS, Google Cloud, and/or Azure, with no downloads necessary.

Give it a try with a free, highly-available 512 MB cluster, or get started from your terminal with the following two commands:

```
$ brew install mongodb-atlas  
$ atlas setup
```

Version  
8.0.3 (current)

Platform  
Windows x64

Package  
msi

Download

Copy link

More Options •••

VS

## Try MongoDB Atlas

A developer data platform built around a fully managed MongoDB service. Address transactional, search, and analytical workloads.

Explore all our products →

## Create your Atlas Account

The multi-cloud developer data platform.

First Name\*

Last Name\*

Company



# 16.2 Setting up MongoDB

## Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10      \$0.08/hour

Dedicated cluster for development environments and low-traffic applications.

STORAGE 10 GB	RAM 2 GB	vCPU 2 vCPUs
------------------	-------------	-----------------

Serverless

For application development and testing, or workloads with variable traffic.

STORAGE Up to 1TB	RAM Auto-scale	vCPU Auto-scale
----------------------	-------------------	--------------------

M0      Free

For learning and exploring MongoDB in a cloud environment.

STORAGE 512 MB	RAM Shared	vCPU Shared
-------------------	---------------	----------------

**Free forever!** Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

**Configurations**

Name  
You cannot change the name once the cluster is created.

Provider

Region

Tag (optional)  
Create your first tag to categorize and label your resources; more tags can be added later. [Learn more](#).

Select or enter key	:	Select or enter value
---------------------	---	-----------------------

complete  
Coding



# 16.2 Setting up MongoDB

Connect to KGCluster

1 Set up connection security    2 Choose a connection method    3 Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more ↗](#)

**1. Add a connection IP address**

Your current IP address (160.202.37.194) has been added to enable local connectivity. Only an IP address you add to your Access List will be able to connect to your project's clusters. Add more later in [Network Access ↗](#).

**2. Create a database user**

This first user will have [atlasAdmin ↗](#) permissions for this project.

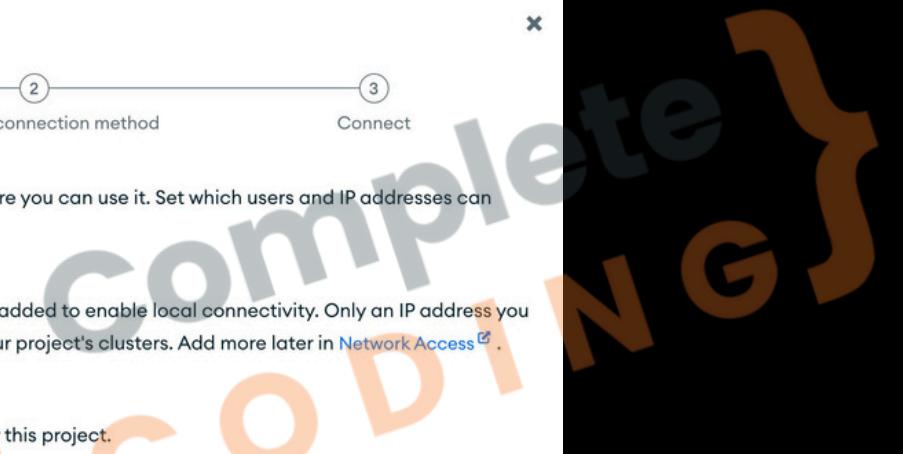
We autogenerated a username and password. You can use this or create your own.

**ⓘ You'll need your database user's credentials in the next step. Copy the database user password.**

**Username**  **Password**  [HIDE](#) [Copy](#)

**Create Database User**

[Close](#) [Choose a connection method](#)





# 16.2 Setting up MongoDB

## Connect to KGCluster

Set up connection security ✓

Choose a connection method 2

Connect 3

### Connect to your application

**Drivers**  
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

**Access your data through tools**

- Compass**  
Explore, modify, and visualize your data with MongoDB's GUI
- Shell**  
Quickly add & update data using MongoDB's Javascript command-line interface
- MongoDB for VS Code**  
Work with your data in MongoDB directly from your VS Code environment
- Atlas SQL**  
Easily connect SQL tools to Atlas for data analysis and visualization

[Go Back](#) [Close](#)

Set up connection security ✓

Choose a connection method 2

Connect 3

### Connecting with MongoDB Driver

**1. Select your driver and version**  
We recommend installing and using the latest driver version.

Driver: Node.js Version: 5.5 or later

**2. Install your driver**  
Run the following on the command line  
`npm install mongodb`

[View MongoDB Node.js Driver installation instructions.](#)

**3. Add your connection string into your application code**

 KGCluster is provisioning...

It takes an average of 10-15 seconds to provision your deployment. Clusters are built with 3 nodes for resiliency.

You will be able to copy your connection string shortly. If you would like to come back to this later, you can go to the [Atlas Home Page](#).

#### RESOURCES

[Get started with the Node.js Driver](#) [Node.js Starter Sample App](#) [Access your Database Users](#) [Troubleshoot Connections](#)

[Go Back](#) [Done](#)



# 16.2 Setting up MongoDB

Project 0 Data Services Charts

Overview We are deploying your changes (current action: creating a plan)

PRASHANT'S ORG - 2024-11-08 > PROJECT 0

## Network Access

IP Access List Peering Private Endpoint

ⓘ You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address

160.202.37.194/32 (includes your current IP address)

Database Access Network Access Advanced

New On Atlas 7 Goto

The screenshot shows the MongoDB Cloud interface for a project. The left sidebar has sections for Overview, DATABASE, Clusters, SERVICES (Atlas Search, Stream Processing, Triggers, Migration, Data Federation), SECURITY, Quickstart, Backup, Database Access (which is highlighted with a red border), Network Access (which is also highlighted with a red border), and Advanced. The main content area is titled 'Network Access' and shows the 'IP Access List' tab selected. It displays a note that only specific IP addresses can connect to the cluster, listing '160.202.37.194/32'. A large, semi-transparent watermark reading 'COMPLETED CODING' is overlaid across the middle of the page.



# 16.3 Installing MongoDB Driver

## TERMINAL

```
● prashantjain@Prashants-Mac-mini Chapter 13 - MVC % npm install mongodb
added 12 packages, and audited 234 packages in 2s
49 packages are looking for funding
  run `npm fund` for details
2 low severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
○ prashantjain@Prashants-Mac-mini Chapter 13 - MVC %
```



# 16.3 Installing MongoDB Driver

JS database.js M X

Test Project > node > Chapter 13 - MVC > utils > JS database.js > ...

```
1 const mongodb = require("mongodb");
2
3 const MongoClient = mongodb.MongoClient;
4
5 const url = "mongodb+srv://root:root@kgcluster.ie6mb.mongodb.net/?retryWrites=true&w=majority&appName=KGCluster";
6
7 const mongoConnect = (callback) => {
8   MongoClient.connect(url)
9     .then((client) => {
10       console.log("Connected to MongoDB");
11       callback(client);
12     })
13     .catch((err) => {
14       console.log(err);
15     });
16   };
17
18 module.exports = mongoConnect;
```



# 16.3 Installing MongoDB Driver

```
Test Project > node > Chapter 13 - MVC > app.js > ...
25
26 | const mongoConnect = require("./utils/database");
27 | const PORT = 3000;
28 | mongoConnect((client) => {
29 |   console.log(client);
30 |   app.listen(PORT, () => {
31 |     console.log(`Server running on address http://localhost:\${PORT}`);
32 |   });
33 | });

Connected to MongoDB
```

```
<ref *1> MongoClient {
  _events: [Object: null prototype] {},
  _eventsCount: 0,
  _maxListeners: undefined,
  mongoLogger: undefined,
  s: {
    url: 'mongodb+srv://root:root@kgcluster.ie6mb.mongodb.net/?retryWrites=true&w=majority&appName=KGCluster',
    bsonOptions: {
      raw: false,
      useBigInt64: false,
      promoteLongs: true,
      promoteValues: true,
      promoteBuffers: false,
      ignoreUndefined: false,
      bsonRegExp: false,
      serializeFunctions: false,
      fieldsAsRaw: {},
      enableUtf8Validation: true
    },
    namespace: MongoDBNamespace { db: 'admin', collection: undefined },
    hasBeenClosed: false,
    sessionPool: ServerSessionPool { client: [Circular *1], sessions: [List] },
    activeSessions: Set(),
    authProviders: MongoClientAuthProviders { existingProviders: [Map] },
    options: [Getter],
    readConcern: [Getter],
    writeConcern: [Getter],
    readPreference: [Getter],
    isMongoClient: [Getter]
  },
  connectionLock: undefined,
  topology: Topology {
    _events: [Object: null prototype] {
      topologyDescriptionChanged: [Array],
      connectionPoolCreated: [Function (anonymous)]
    }
  }
}
```

```
' Server running on address http://localhost:3000
TypeError: db.execute is not a function
  at Home.fetchAll (/Users/prashantjain/workspace/stuff/Test Project/node/Chapter 13 - MVC/controllers/HomeController.js:5:8)
  at Layer.handle [as handle_request] (/Users/prashantjain/workspace/stuff/Test Project/node_modules/express/lib/router/layer.js:95:5)
  at next (/Users/prashantjain/workspace/stuff/Test Project/node/Chapter 13 - MVC/controllers/index.js:149:13)
  at Route.dispatch (/Users/prashantjain/workspace/stuff/Test Project/node_modules/express/lib/router/route.js:119:3)
  at Layer.handle [as handle_request] (/Users/prashantjain/workspace/stuff/Test Project/node_modules/express/lib/router/layer.js:95:5)
  at /Users/prashantjain/workspace/stuff/Test Project/node/Chapter 13 - MVC/controllers/index.js:284:15
  at Function.process_params (/Users/prashantjain/workspace/stuff/Test Project/node_modules/express/lib/router/index.js:346:12)
  at next (/Users/prashantjain/workspace/stuff/Test Project/node/Chapter 13 - MVC/controllers/index.js:280:10)
  at Function.handle (/Users/prashantjain/workspace/stuff/Test Project/node_modules/express/lib/router/index.js:175:3)
```



# 16.3 Installing MongoDB Driver

```
module.exports = class Home {  
  constructor(name, description, price, location, rating,  
  imageUrl) {  
    this.name = name;  
    this.description = description;  
    this.price = price;  
    this.location = location;  
    this.rating = rating;  
    this.imageUrl = imageUrl;  
  }  
  
  save() {  
  }  
  
  static fetchAll() {  
  }  
  
  static findById(id) {  
  }  
  
  static deleteById(id) {  
  }  
};
```

{ Complete Coding }



# 16.4 Creating MongoDB Connection

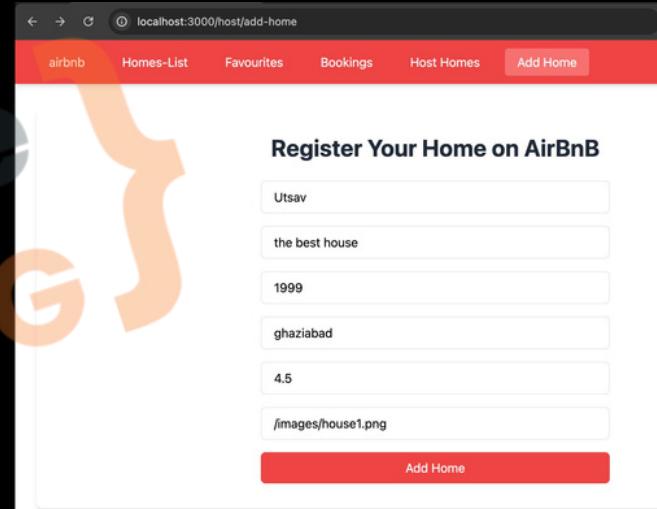
```
database.js M X app.js M home.js M
Test Project > node > Chapter 13 - MVC > utils > database.js > ...
1 const mongodb = require("mongodb");
2
3 const MongoClient = mongodb.MongoClient;
4
5 const url = "mongodb+srv://root:root@kgcluster.ie6mb.mongodb.net/?retryWrites=true&
w=majority&appName=KGCluster";
6
7 let _db;
8
9 const mongoConnect = (callback) => {
10   MongoClient.connect(url)
11     .then((client) => {
12       console.log("Connected to MongoDB");
13       _db = client.db("airbnb");
14       callback();
15     })
16     .catch((err) => {
17       console.log(err);
18       throw err;
19     });
20 };
21
22 const getDb = () => {
23   if (!_db) {
24     throw new Error("Database not connected");
25   }
26   return _db;
27 };
28
29 exports.mongoConnect = mongoConnect;
30 exports.getDb = getDb;
```

Complete Coding



# 16.5 Saving a Home

```
save() {  
  const db = getDb();  
  // insertMany takes an array of objects  
  return db.collection("homes").insertOne(this).then((result) => {  
    console.log(result);  
  });  
}  
  
{  
  acknowledged: true,  
  insertedId: new ObjectId('672df86c3d20696a2b523c79')  
}
```





# 16.6 Install MongoDB Compass

The screenshot shows the official MongoDB website. At the top, there is a navigation bar with links for Products, Resources, Solutions, Company, Pricing, a search bar, language selection (Eng), Support, Sign In, and a prominent green "Try Free" button. The main content area features a large, stylized title: "Compass. The GUI for MongoDB." Below the title, a descriptive paragraph explains what Compass is: "Compass is a free interactive tool for querying, optimizing, and analyzing your MongoDB data. Get key insights, drag and drop to build pipelines, and more." At the bottom, there are two calls-to-action: a green "Download Now" button and a link "Read the docs →".

MongoDB. Products ▾ Resources ▾ Solutions ▾ Company ▾ Pricing

TOOLS

# Compass. The GUI for MongoDB.

Compass is a free interactive tool for querying, optimizing, and analyzing your MongoDB data. Get key insights, drag and drop to build pipelines, and more.

Download Now

Read the docs →



# 16.6 Install MongoDB Compass

Learn more

Version  
1.44.6 (Stable)

Platform  
Windows 64-bit (10+)

Package  
exe

Download ↓

Copy link

More Options ⋮

A screenshot of a download page for MongoDB Compass. The page displays the following information:

- Version:** 1.44.6 (Stable)
- Platform:** Windows 64-bit (10+)
- Package:** exe

At the bottom of the page, there are three buttons: "Download ↓", "Copy link", and "More Options ⋮". A large, semi-transparent watermark reading "Complete CODING" is overlaid diagonally across the page.



# 16.6 Install MongoDB Compass

The image shows the MongoDB Compass application window. On the left is a sidebar with tabs for 'Compass' and 'My Queries'. The main area is titled 'Welcome' and features a large, semi-transparent watermark reading 'Complete' in grey and orange. Below the watermark, the text 'Welcome to MongoDB Compass' is displayed, followed by 'To get started, connect to an existing server or'. A green button labeled '+ Add new connection' is highlighted with a red box. To the right of this button is a light blue callout box containing the text 'New to Compass and don't have a cluster? If you don't already have a cluster, you can create one for free using MongoDB Atlas' and a 'CREATE FREE CLUSTER' button. The bottom of the screen shows a Windows taskbar with various icons and a system tray.



# 16.6 Install MongoDB Compass

**New Connection**  
Manage your connection settings

**URI** i Edit Connection String

`mongodb+srv://root:*****@kgcluster.ie6mb.mongodb.net/`

**Name**  **Color**  Orange

**Favorite this connection**  
Favoriting a connection will pin it to the top of your list of connections

Advanced Connection Options

**Save** **Save & Connect**

**complete**

**How do I find my connection string in Atlas?**  
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.  
[See example ↗](#)

**How do I format my connection string?**  
[See example ↗](#)



# 16.6 Install MongoDB Compass

The screenshot shows the MongoDB Compass interface. On the left, the 'Connections' sidebar lists 'kgcluster.ie6mb.mongodb.net' and 'airbnb'. The 'airbnb' connection is selected, highlighted with a blue border. The main pane displays the 'airbnb' database with a single collection named 'homes'. The collection details are as follows:

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	1	159.00 B	1	20.48 kB

A large, semi-transparent watermark reading 'COMPLETE CODING' is overlaid across the center of the screen.



# 16.6 Install MongoDB Compass

The screenshot shows the MongoDB Compass interface. At the top, there's a navigation bar with 'Welcome' and a selected 'homes' database. A '+' button is available for creating new databases. Below the navigation is a breadcrumb trail: 'kgcluster.ie6mb.mongodb.net > airbnb > homes'. On the right, there's a link to 'Open MongoDB shell'.

The main area has tabs for 'Documents' (1), 'Aggregations', 'Schema', 'Indexes' (1), and 'Validation'. The 'Documents' tab is active. A search bar contains the placeholder 'Type a query: { field: 'value' } or [Generate query](#)'. To the right of the search bar are buttons for 'Explain', 'Reset', 'Find' (which is highlighted in green), and 'Options'.

Below the search bar are action buttons: '+ ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. To the right are pagination controls: '25', '1-1 of 1', and arrows for navigating through results. There are also icons for filtering and expanding the view.

The document details section shows a single document with the following fields:

```
_id: ObjectId('672df86c3d20696a2b523c79')
name : "Utsav"
description : "the best house"
price : "1999"
location : "ghaziabad"
rating : "4.5"
imageUrl : "/images/house1.png"
```



# 16.7 Install MongoDB for VSCode

The screenshot shows the VS Code Marketplace search results for 'mongodb'. The search bar at the top has 'mongodb' typed in. Below it, there's a 'MARKETPLACE' section with a count of 39 items. The first item listed is 'MongoDB for VS Code' by 'mongodb', which has 1.9M installs and a 4.5 rating. It includes a green leaf icon and a 'Install' button. Other items listed include 'ES7 JavaScript/Node.js', 'Auto MongoDB', 'Azure Databases', 'Mongodb-dly', 'MongoDB ID Generator', 'MySQL', 'Mongo Snippets', and 'Azure Tools'.

The screenshot shows the details page for the 'MongoDB for VS Code' extension. At the top, it displays the extension name 'MongoDB for VS Code' with version v1.9.3, 1,900,000 installs, and a 5-star rating from 39 reviews. There are 'Install' and 'Auto Update' buttons. Below this, there are tabs for 'DETAILS' and 'FEATURES'. The 'DETAILS' tab contains a large image of a hand pointing at a computer screen displaying a MongoDB interface, with the text 'Get Started Watch the demo video'. The 'FEATURES' tab lists 'Navigate your MongoDB Data'.

**MongoDB for VS Code** v1.9.3

mongodb | 1,900,000 | ★★★★★ (39)

Connect to MongoDB and Atlas directly from your VS Code environment, navigate your databases...

**Install**  Auto Update

**DETAILS** **FEATURES**

## MongoDB for VS Code

**Test and Build** passing

MongoDB for VS Code makes it easy to work with your data in MongoDB directly from your VS Code environment. MongoDB for VS Code is the perfect companion for MongoDB Atlas, but you can also use it with your self-managed MongoDB instances.

### Get Started

Watch the demo video

### Features

Navigate your MongoDB Data

**Categories**

- Programming Languages
- Snippets
- Data Science
- AI
- Chat

**Resources**

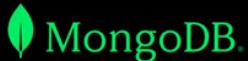
- Marketplace
- Issues
- Repository
- License
- mongodb

**More Info**

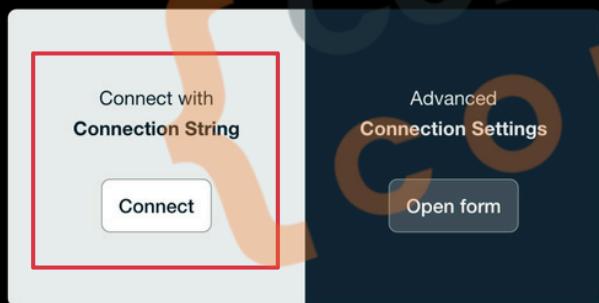
Published	2020-05-12, 01:46:46
Last released	2024-10-24, 18:53:28
Identifier	mongodb.mongodb-vscode



# 16.7 Install MongoDB for VSCode



Navigate your databases and collections, use playgrounds for exploring and transforming your data



Cmd + Shift + P for all MongoDB Command Palette options



New to MongoDB and don't have a cluster?

Create one for free using [MongoDB Atlas](#).

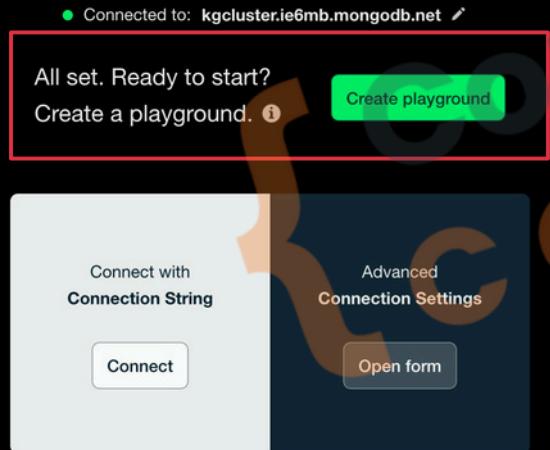
Create free cluster



# 16.7 Install MongoDB for VSCode

MongoDB.

Navigate your databases and collections, use playgrounds for exploring and transforming your data



Cmd + Shift + P for all MongoDB Command Palette options



New to MongoDB and don't have a cluster?  
Create one for free using [MongoDB Atlas](#).

Create free cluster



# 16.7 Install MongoDB for VSCode

Currently connected to kgcluster.ie6mb.mongodb.net. Click here to change connection.

```
1 /* global use, db */
2 // MongoDB Playground
3 // To disable this template go to Settings | MongoDB | Use Default Template For Playground.
4 // Make sure you are connected to enable completions and to be able to run a playground.
5 // Use Ctrl+Space inside a snippet or a string literal to trigger completions.
6 // The result of the last command run in a playground is shown on the results panel.
7 // By default the first 20 documents will be returned with a cursor.
8 // Use 'console.log()' to print to the debug output.
9 // For more documentation on playgrounds please refer to
10 // https://www.mongodb.com/docs/mongodb-vscode/playgrounds/
11
12 // Select the database to use.
13 use('airbnb');
14
15 // Get all documents from homes collection
16 const homes = db.getCollection('homes').find({}).toArray();
17
18 // Print the results
19 console.log('All homes in collection:');
20 console.log(JSON.stringify(homes, null, 2));
21
22 // Print total count
23 const totalHomes = db.getCollection('homes').countDocuments();
24 console.log(`Total number of homes: ${totalHomes}`);
```



# 16.7 Install MongoDB for VSCode

Currently connected to kgcluster.ie6mb.mongodb.net. Click here to change connection.

```
1 /* global use, db */
2 // MongoDB Playground
3 // To disable this template go to Settings | MongoDB | Use Default Template For Playground.
4 // Make sure you are connected to enable completions and to be able to run a playground.
5 // Use Ctrl+Space inside a snippet or a string literal to trigger completions.
6 // The result of the last command run in a playground is shown on the results panel.
7 // By default the first 20 documents will be returned with a cursor.
8 // Use 'console.log()' to print to the debug output.
9 // For more documentation on playgrounds please refer to
10 // https://www.mongodb.com/docs/mongodb-vscode/playgrounds/
11
12 // Select the database to use.
13 use('airbnb');
14
15 // Get all documents from homes collection
16 const homes = db.getCollection('homes').find({}).toArray();
17
18 // Print the results
19 console.log('All homes in collection:');
20 console.log(JSON.stringify(homes, null, 2));
21
22 // Print total count
23 const totalHomes = db.getCollection('homes').countDocuments();
24 console.log(`Total number of homes: ${totalHomes}`);
```

```
All homes in collection:
[
  {
    "_id": "672df86c3d20696a2b523c79",
    "name": "Utsav",
    "description": "the best house",
    "price": "1999",
    "location": "ghaziabad",
    "rating": "4.5",
    "imageUrl": "/images/house1.png"
  }
]
Total number of homes: 1
```



# 16.8 Fetching all Homes

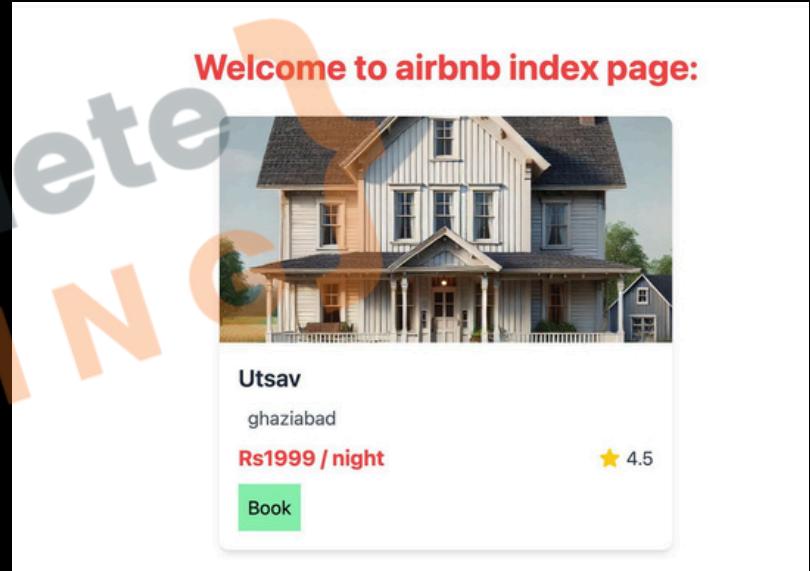
```
static fetchAll() {  
  const db = getDb();  
  return db.collection("homes")  
    .find()  
    .toArray()  
    .then((homes) => {  
      console.log(homes);  
      return homes;  
    }).catch((err) => {  
      console.log(err);  
    });  
}
```

1. Basic Usage: `collection.find(query)` retrieves documents matching the query criteria.
2. Returns a Cursor: The method returns a cursor, an iterator over the result set.



# 16.8 Fetching all Homes

```
exports.getHomes = (req, res, next) => {
  Home.fetchAll()
    .then(rows => {
      res.render("store/home-list", {
        registeredHomes: rows,
        pageTitle: "Homes List",
        currentPage: "Home",
      });
    })
    .catch((error) => {
      console.log("Error Fetching Homes", error);
    });
};
```





# 16.9 Fetching one Home

```
static findById(homeId) {  
  const db = getDb();  
  return db.collection("homes")  
    .find({ _id: homeId })  
    .next()  
    .then((home) => {  
      console.log(home);  
      return home;  
    }).catch((err) => {  
      console.log(err);  
    });  
}
```

```
exports.getHome = (req, res, next) => {  
  const homeId = req.params.homeId;  
  Home.findById(homeId).then(home => {  
    if (!home) {  
      return res.redirect("/homes");  
    }  
    res.render("store/home-detail", {  
      home: home,  
      pageTitle: home.name,  
      currentPage: "homes",  
    });  
  }).catch((error) => {  
    console.log("Error Fetching Home", error);  
  });  
};
```



# 16.9 Fetching one Home

SEARCH

home.id

home.\_id

files to include

files to exclude

9 results in 7 files - Open in editor

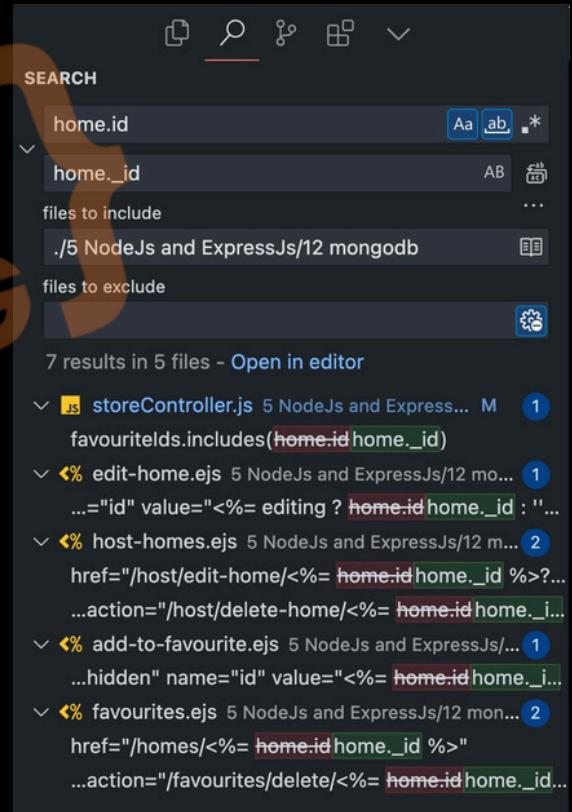
- hostController.js Test Project/node/Chapter 13 - MVC/controllers
  - home.id home.\_id = id;
- storeController.js Test Project/node/Chapter 13 - MVC/controllers
  - registeredHomes.find((home) => home.id === homeld)
- edit-home.ejs Test Project/node/Chapter 13 - MVC/views/host
  - ...name="id" value="<%= home ? home.id : '' %>">
- host-home-list.ejs Test Project/node/Chapter 13 - MVC/views/host
  - ...href="/host/edit-home/<%= home.id %>?editing=true" class="bg...
  - ...action="/host/delete-home/<%= home.id %>" method="POST" cl...
- favourite.ejs Test Project/node/Chapter 13 - MVC/views/partials
  - ... name="homeld" value="<%= home.id %>">
- favourite-list.ejs Test Project/node/Chapter 13 - MVC/views/store
  - <a href="/homes/<%= home.id %>" class="flex-1 bg-white text-re..."
  - ...action="/favourites/delete/<%= home.id %>" method="POST">
- home-list.ejs Test Project/node/Chapter 13 - MVC/views/store
  - <a href="/homes/<%= home.id %>" class="flex-1 bg-white text-re..."

```
[  
  {  
    _id: new ObjectId('672df86c3d20696a2b523c79'),  
    name: 'Utsav',  
    description: 'the best house',  
    price: '1999',  
    location: 'ghaziabad',  
    rating: '4.5',  
    imageUrl: '/images/house1.png'  
  }]
```



# 16.9 Fetching one Home

```
static findById(homeId) {  
  const db = getDb();  
  return db.collection("homes")  
    .find({ _id: new ObjectId(String(homeId)) })  
    .next()  
    .then((home) => {  
      console.log(home);  
      return home;  
    }).catch((err) => {  
      console.log(err);  
    });  
}
```





# 16.9 Fetching one Home

**Details of Utsav**



**Description**  
the best house

**Location**  
ghaziabad

**Price**  
\$1999 / night

**Rating**  
★ 4.5 / 5

Add to Favorites

complete  
CODING}



# 16.10 Supporting Edit & Delete

```
save() {  
  const db = getDb();  
  if (this.id) {  
    // Update existing home  
    return db  
      .collection("homes")  
      .updateOne(  
        { _id: new ObjectId(String(this.id)) },  
        { $set: this }  
      );  
  } else {  
    // Insert new home  
    return db  
      .collection("homes")  
      .insertOne(this)  
      .then((result) => {  
        console.log(result);  
      });  
  }  
}
```

```
static deleteById(homeId) {  
  const db = getDb();  
  return db  
    .collection("homes")  
    .deleteOne({ _id: new ObjectId(String(homeId)) });  
}
```

# 16.11 Adding MongoDB to Favourite

1. Remove all the file handling related code from Favourite Model.

2. Delete the data folder.

3. Change the following methods in Favourite model to use mongo:

- a. fetchAll
- b. Change addToFavourites to save method
- c. deleteById

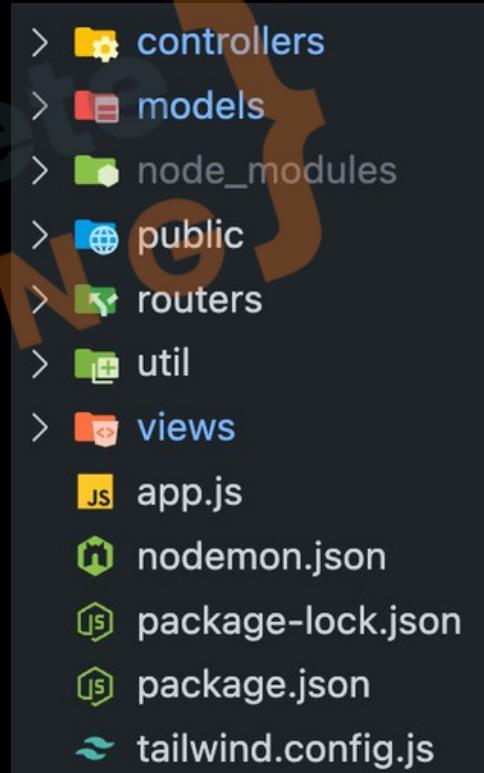
4. Change the usages of Favourite model in StoreController to use the promise syntax

- a. getFavourites
- b. postAddFavourites
- c. postRemoveFavourite

# 16.11 Adding MongoDB to Favourite

```
1. module.exports = class Favourite {  
    constructor(homeId) {  
        this.homeId = homeId;  
    }  
  
    save() {}  
  
    static fetchAll() {}  
  
    static deleteById(homeId) {}  
}
```

2.





# 16.11 Adding MongoDB to Favourite

```
3. save() {
    const db = getDb();
    return db.collection("favourites").insertOne(this);
}

static fetchAll() {
    const db = getDb();
    return db.collection('favourites').find().toArray();
}

static deleteById(homeId) {
    const db = getDb();
    return db.collection('favourites').deleteOne({homeId});
}
```



# 16.11 Adding MongoDB to Favourite

```
4.a. exports.getFavourites = (req, res, next) => {
  Favourite.fetchAll().then((favouriteIds) => {
    Home.fetchAll().then((registeredHomes) => {
      console.log(favouriteIds);
      console.log(registeredHomes);
      const favouriteHomes = registeredHomes.filter((home) =>
        favouriteIds.includes(home._id.toString())
      );
      res.render("store/favourites", {
        homes: favouriteHomes,
        pageTitle: "Favourites",
      });
    });
  });
};
```



# 16.11 Adding MongoDB to Favourite

```
4.b. exports.postAddFavourites = (req, res, next) => {
  const homeId = req.body.id;
  const fav = new Favourite(homeId);
  fav
    .save()
    .then(() => {
      res.redirect("/favourites");
    })
    .catch((err) => {
      console.log("Error while adding to favourites", err);
      res.redirect("/favourites");
    });
};
```

# 16.11 Adding MongoDB to Favourite

4.c.

```
exports.postRemoveFavourite = (req, res, next) => {
  const homeId = req.params.homeId;
  Favourite.deleteById(homeId)
    .then(() => {
      res.redirect("/favourites");
    })
    .catch((err) => {
      console.log("Error while remove from favourites ", err);
    });
};
```



# Practise Milestone

Take your **airbnb** forward:

- Change the favourite model to fix the problem of having duplicate favourite records.

{ Complete Coding }





# Practise Milestone (Solution)

```
save() {  
  const db = getDb();  
  return db.collection('favourites').findOne({homeId: this.homeId})  
    .then(existingFav => {  
      if (!existingFav) {  
        return db.collection("favourites").insertOne(this);  
      }  
      return Promise.resolve();  
    });  
}
```

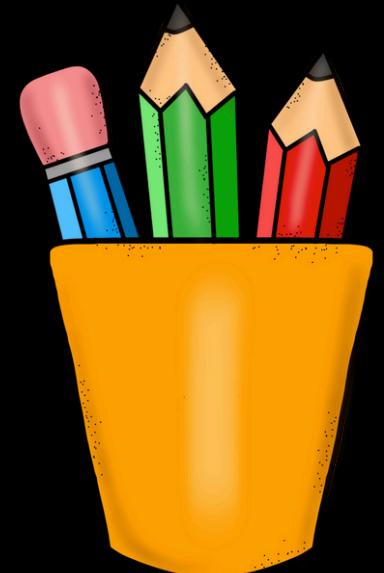
COMPLETED CODING





# Revision

1. What is MongoDB
2. Setting up MongoDB
3. Installing MongoDB Driver
4. Creating MongoDB Connection
5. Saving a Home
6. Install MongoDB Compass
7. Install MongoDB for VSCode
8. Fetching all Homes
9. Fetching one Home
10. Supporting Edit & Delete
11. Adding MongoDB to Favourite





{ complete  
C O D I N G }



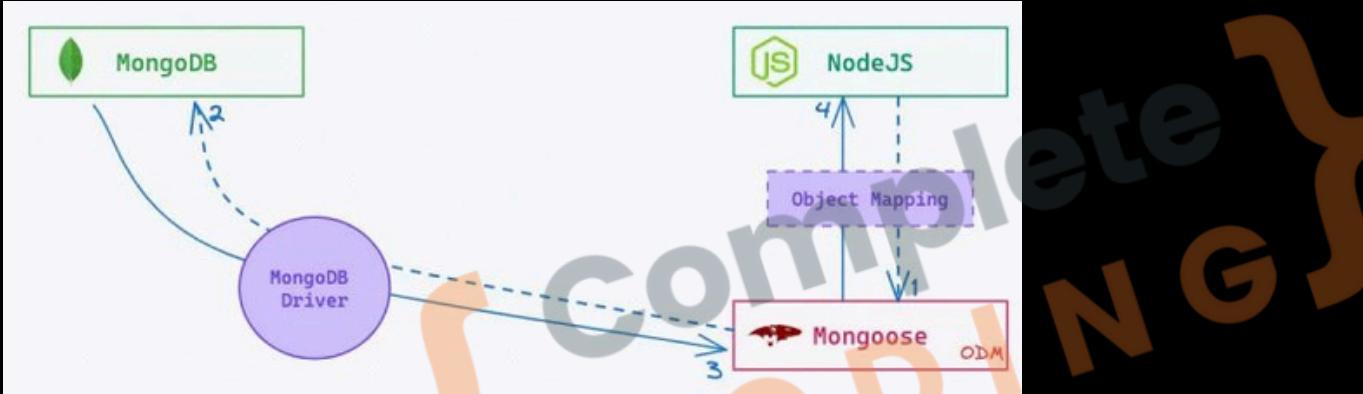
# 17. Introduction to Mongoose

- 1.What is Mongoose
- 2.Setting up Mongoose
- 3.Create Home Schema
- 4.Saving Home using Mongoose
5. Fetching Homes
- 6.Fetching one Home
- 7.Editing a Home
- 8.Deleting a Home
- 9.Using Mongoose for Favourite
10. Fetching Relations





# 17.1 What is Mongoose



1. Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js.
2. Provides a schema-based solution to model application data.
3. Simplifies data validation and type casting in Node.js applications.
4. Enables easy interaction with MongoDB through intuitive methods.
5. Supports middleware for pre and post-processing of data.
6. Helps manage relationships between data with built-in functions.



# 17.2 Setting up Mongoose

# mongoose

elegant **mongodb** object modeling for **node.js**

[read the docs](#)

[discover plugins](#)



Version 8.8.0



Let's face it, writing MongoDB validation, casting and business logic boilerplate is a drag. That's why we wrote Mongoose.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/test');

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.



[Get Professionally Supported Mongoose](#)

complete  
coding}



## 17.2 Setting up Mongoose

1. Install Mongoose package.
2. Import and use mongoose in app.js
3. Delete the database-util file.
4. Remove the usage of db-util from everywhere.

1.

```
● prashantjain@Prashants-Mac-mini:12 mongodb % npm install mongoose
  added 8 packages, and audited 258 packages in 1s
  48 packages are looking for funding
    run `npm fund` for details
  found 0 vulnerabilities
```



## 17.2 Setting up Mongoose

2. const PORT = 3001;  
mongoose.connect("mongodb+srv://root:root@kgcluster.ie6mb.mongodb.net/airbnb?  
retryWrites=true&w=majority&appName=KGCluster").then(() => {  
 console.log("Connected to MongoDB");  
 app.listen(PORT, () => {  
 console.log(`Server running at: http://localhost:\${PORT}`);  
 });  
}).catch((err) => {  
 console.log("Error while connecting to MongoDB", err);  
});



## 17.3 Create Home Schema

1. Delete complete airbnb db from mongo.
2. Delete the existing Home Model code.
3. Create the new Home Schema in the Home Model File.

```
const mongoose = require("mongoose");

// _id is automatically added by mongoose
const homeSchema = new mongoose.Schema({
  houseName: {type: String, required: true},
  price: {type: Number, required: true},
  location: {type: String, required: true},
  rating: {type: Number, required: true},
  photoUrl: String,
  description: String
});
```



# 17.4 Saving Home using Mongoose

```
module.exports = mongoose.model("Home", homeSchema);

exports.postAddHome = (req, res, next) => {
  const { houseName, price, location, rating, photoUrl, description } =
    req.body;
  const newHome = new Home({
    houseName,
    price,
    location,
    rating,
    photoUrl,
    description
  });

  newHome.save().then((rows) => {
    res.render("host/home-added", { pageTitle: "Home Hosted" });
  });
};
```



# 17.5 Fetching Homes

The screenshot shows a code editor's search interface. The search bar at the top contains the query "fetchAll()". Below the search bar, there are several filters: "Aa" (case sensitive), "ab" (case insensitive), and a dropdown menu with an asterisk (\*). There are also buttons for "AB" (case insensitive) and a file icon. Below these are sections for "files to include" and "files to exclude", both currently empty. A "Recent files" section lists ".5 NodeJs and ExpressJs/12 mongodb" and "files to exclude". At the bottom, it says "4 results in 4 files - Open in editor". A list of files is shown below:

- hostController.js: Home.fetchAll().find().then((registeredHomes) => {
- storeController.js: Favourite.fetchAll().find().then((favouritelds) => {
- Favourite.js: static fetchAll() find() {
- Home.js: // static fetchAll() find() {

```
exports.getIndex = (req, res, next) => {
  Home.find().then((registeredHomes) => {
    res.render("store/index", {
      homes: registeredHomes,
      pageTitle: "Tumahara airbnb",
    });
  });
};
```



## 17.6 Fetching one Home

It already works!!!

{ complete }  
{ CODING }



## 17.7 Editing a Home

```
exports.postEditHome = (req, res, next) => {
  const { id, houseName, price, location, rating, photoUrl, description } = req.body;
  Home.findById(id).then((home) => {
    if (!home) {
      console.log("Home not found for editing");
      return res.redirect("/host/host-homes");
    }
    home.houseName = houseName;
    home.price = price;
    home.location = location;
    home.rating = rating;
    home.photoUrl = photoUrl;
    home.description = description;
    return home.save();
  }).then(() => {
    res.redirect("/host/host-homes");
  })
  .catch((err) => {
    console.log("Error while updating home", err);
  });
};
```



## 17.8 Deleting a Home

```
exports.postDeleteHome = (req, res, next) => {
  const homeId = req.params.homeId;
  console.log("Came to delete ", homeId);
  Home.findByIdAndDelete(homeId).then(() => {
    res.redirect("/host/host-homes");
  });
};
```



# 17.9 Using Mongoose for Favourite

1. Delete the existing Favourite Model code.
2. Create the new Favourite Schema in the Favourite Model File.
3. Fix the following functionalities:
  - a. Getting all Favourite
  - b. Adding A Favourite
  - c. Deleting a Favourite
4. Removing Favourite while removing home.



# 17.9 Using Mongoose for Favourite

1, 2.

```
const mongoose = require('mongoose');
...
const favouriteSchema = new mongoose.Schema({
  homeId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Home',
    required: true,
    unique: true
  }
});

module.exports = mongoose.model('Favourite', favouriteSchema);
```

A large, semi-transparent watermark reading "Complete CODING" is overlaid across the slide, with the letters slightly slanted and overlapping each other.



# 17.9 Using Mongoose for Favourite

```
3.a. exports.getFavourites = (req, res, next) => {
  Favourite.find().then((favouriteIds) => {
    favouriteIds = favouriteIds.map((favourite) => favourite.homeId.toString());
    Home.find().then((registeredHomes) => {
      console.log(favouriteIds);
      console.log(registeredHomes);
      const favouriteHomes = registeredHomes.filter((home) =>
        favouriteIds.includes(home._id.toString())
      );
      res.render("store/favourites", {
        homes: favouriteHomes,
        pageTitle: "Favourites",
      });
    });
  });
};
```



# 17.9 Using Mongoose for Favourite

3.b.

```
exports.postAddFavourites = (req, res, next) => {
  const homeId = req.body.id;
  Favourite.findOne({ homeId: homeId })
    .then(existingFav => {
      if (existingFav) {
        return res.redirect("/favourites");
      }
      const fav = new Favourite({ homeId: homeId });
      return fav.save();
    })
    .then(() => {
      res.redirect("/favourites");
    })
    .catch((err) => {
      console.log("Error while adding to favourites", err);
    });
};
```



# 17.9 Using Mongoose for Favourite

3.c.

```
exports.postRemoveFavourite = (req, res, next) => {
  const homeId = req.params.homeId;
  Favourite.findOneAndDelete({ homeId: homeId })
    .then(() => {
      res.redirect("/favourites");
    })
    .catch((err) => {
      console.log("Error while remove from favourites ", err);
    });
};
```



# 17.9 Using Mongoose for Favourite

4.

```
homeSchema.pre('findOneAndDelete', async function(next) {  
  const homeId = this.getQuery()["_id"];  
  await Favourite.deleteMany({ homeId: homeId });  
  next();  
});
```



# 17.10 Fetching Relations

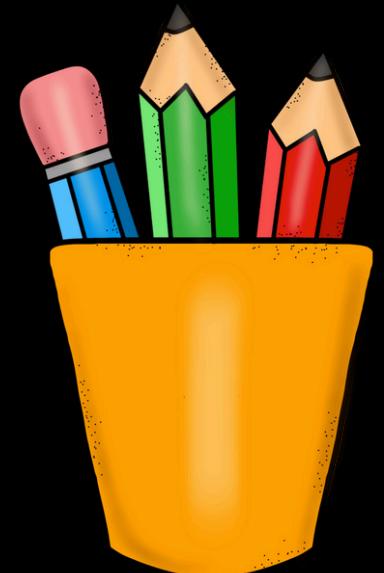
```
exports.getFavourites = (req, res, next) => {
  Favourite.find()
    .populate("homeId")
    .then((favourites) => {
      const favouriteHomes = favourites.map((favourite) => favourite.homeId);
      res.render("store/favourites", {
        homes: favouriteHomes,
        pageTitle: "Favourites",
      });
    });
};
```

A large, semi-transparent watermark is angled across the slide. It contains the word "complete" in a light blue sans-serif font, followed by "CODING" in a larger, bold brown sans-serif font.



# Revision

1. What is Mongoose
2. Setting Mongoose
3. Create Home Schema
4. Saving Home using Mongoose
5. Fetching Homes
6. Fetching one Home
7. Editing a Home
8. Deleting a Home
9. Using Mongoose for Favourite
10. Fetching Relations





{ complete  
C O D I N G }



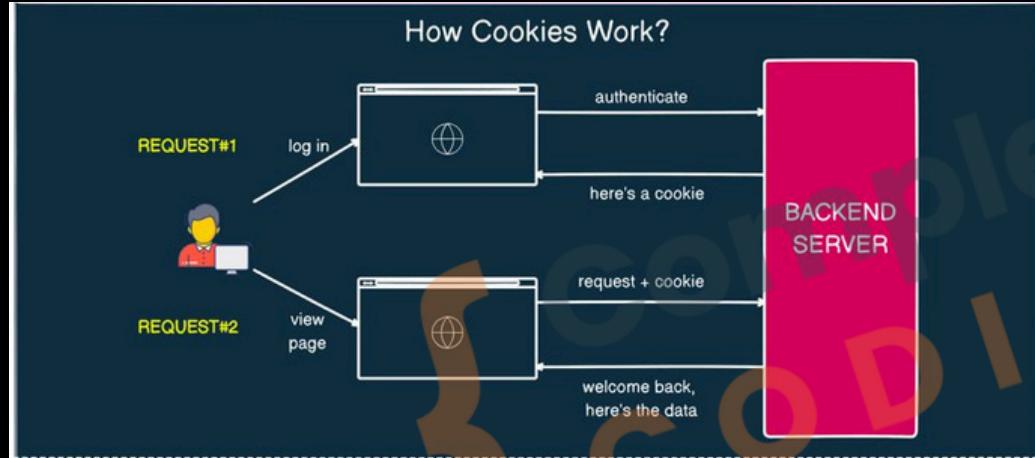
# 18. Cookies & Sessions

1. What are Cookies
2. Adding Login Functionality
3. Checking Login State
4. Using a Cookie
5. Define the Logout Feature
6. Problem with Cookies
7. What are Sessions
8. Installing Session Package
9. Creating a Sessions
10. Saving Session in DB





# 18.1 What are Cookies



1. Cookies are small pieces of data stored in the user's browser by server.
2. They help websites remember user information and preferences between page loads or visits.
3. Cookies can manage user sessions and store data for personalized experiences.



# 18.1 What are Cookies

Screenshot of the Chrome DevTools Network tab showing cookies for learn.completecoding.in.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly
MUID	0644EB8141A266AA049EFEB74050676E	.clarity.ms	/	2025-12-08T09:00:21.314Z	36	
_clk	1sddxv4%7C2%7Cfw%7C0%7C1778	.completecoding.in	/	2025-11-15T03:08:42.000Z	33	
_clsk	10b84hl%7C1731668781029%7C2%7C0%7Ck.clarity....	.completecoding.in	/	2024-11-16T11:06:21.000Z	61	
_ga	GA1.1.585233942.1731488419	.completecoding.in	/	2025-12-20T03:08:42.237Z	29	
_ga_8JBLENZJ2Z	GS1.1.1731640107.6.1.1731640122.0.0	.completecoding.in	/	2025-12-20T03:08:42.171Z	51	
_ga_QBNBN7VB0P	GS1.1.1731640107.6.1.1731640122.0.0	.completecoding.in	/	2025-12-20T03:08:42.242Z	51	
_gcl_au	1.1.1428947795.1731488419	.completecoding.in	/	2025-02-11T09:00:19.000Z	32	
amp_e56929	Rk-Al4gjFrnePO5Vi2d1kq.NjUxMjc4MzIINGlwYzdKMTU...	.completecoding.in	/	2025-11-15T03:08:42.000Z	93	
amp_e56929_completecoding.in	Rk-Al4gjFrnePO5Vi2d1kq.NjUxMjc4MzIINGlwYzdKMTU...	.completecoding.in	/	2025-11-13T15:33:29.000Z	110	
SESSIONID	C2A93FEDBFC031164BE4A45FA51E74F2	learn.completecoding.in	/	Session	41	✓
c_login_token	1695797312295	learn.completecoding.in	/	2025-12-18T09:01:00.498Z	26	✓
id	5a3e5c2c-939d-4b02-a1d9-a8a793efcf50	learn.completecoding.in	/	2025-12-18T09:00:18.028Z	38	✓
org.springframework.web.servlet.i18n.CookieLocaleResolver.LOC...	en	learn.completecoding.in	/	Session	66	



## 18.2 Adding Login Functionality

1. Add a **login button** to the **nav bar** pointing to **/login**
2. Create a **auth router** to handle login related **routes**, register the new router in **app.js**
3. Create a **auth controller** to handle **GET** request for **/login** and return a UI that does the following:
  - a. Accepts **email** and **password**
  - b. Has a **Login** button that submits the form to **/login** with a **POST** request.
4. Add a **POST** login path in **router** and **handler** in controller.
5. Assume the person logged in and redirect them to the home page.



# 18.2 Adding Login Functionality

1.

```
<div>
  <a href="/login" class="bg-blue-600 hover:bg-blue-700 text-white font-semibold py-2.5
    px-6 rounded-lg transition duration-300 ease-in-out transform hover:scale-105 shadow-md">
    Login
  </a>
</div>
```

{ CODING }



## 18.2 Adding Login Functionality

2.

```
const express = require("express");
const authRouter = express.Router();
const authController = require("../controllers/authController");

authRouter.get("/login", authController.getLogin);

exports.authRouter = authRouter;
```

```
app.use("/host", hostRouter);
app.use(authRouter);
```



## 18.2 Adding Login Functionality

3.

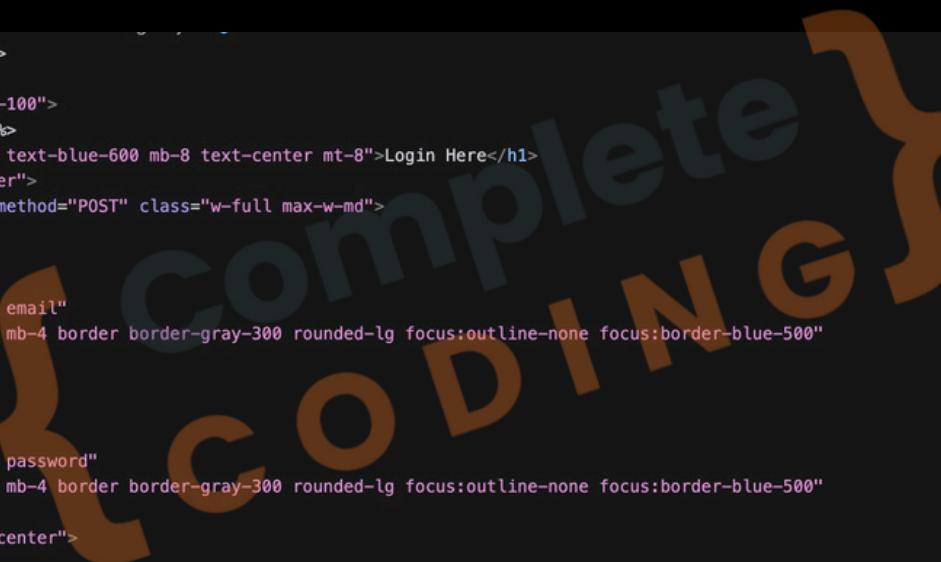
```
exports.getLogin = (req, res, next) => {
  ...
  res.render("auth/login", { pageTitle: "Login" });
};
```



# 18.2 Adding Login Functionality

3.

```
<%- include('../partials/head') %>
</head>
<body class="min-h-screen bg-gray-100">
<%- include('../partials/nav') %>
<h1 class="text-4xl font-bold text-blue-600 mb-8 text-center mt-8">Login Here</h1>
<div class="flex justify-center">
  <form action="/auth/login" method="POST" class="w-full max-w-md">
    <input
      type="email"
      name="email"
      placeholder="Enter your email"
      class="w-full px-4 py-2 mb-4 border border-gray-300 rounded-lg focus:outline-none focus:border-blue-500"
    />
    <input
      type="password"
      name="password"
      placeholder="Enter your password"
      class="w-full px-4 py-2 mb-4 border border-gray-300 rounded-lg focus:outline-none focus:border-blue-500"
    />
    <div class="flex justify-center">
      <input
        type="submit"
        value="Login"
        class="bg-blue-500 hover:bg-blue-600 text-white font-semibold py-2 px-4 rounded-lg transition duration-300 ease-in-out transform hover:scale-105 cursor-pointer"
      />
    </div>
  </form>
</div>
</main>
</body>
</html>
```





## 18.2 Adding Login Functionality

4, 5.

```
authRouter.get("/login", authController.getLogin);
authRouter.post("/login", authController.postLogin);
```

```
exports.postLogin = (req, res, next) => {
  console.log(req.body);
  res.redirect("/");
};
```



## 18.3 Checking Login State

1. Add a `isLoggedIn` field in the `req` object and use it everywhere else.
2. Add a condition in the `navigation.ejs` file that no path other than `home` and `login` should be visible until a user has logged in.
3. Fix all the `render` calls to send the flag
4. Also add a middleware for host routes that if the user is not logged in they should be redirected to the login page.



## 18.3 Checking Login State

1.

```
exports.postLogin = (req, res, next) => {
  console.log(req.body);
  req.isLoggedIn = true;
  res.redirect("/");
};
```



# 18.3 Checking Login State

2.

```
<% if (isLoggedIn) { %>
  <a href="/favourites" class="bg-blue-600 hover:bg-blue-700 text-white">
    Favourites
  </a>
  <a href="/host/host-homes" class="bg-blue-600 hover:bg-blue-700 text-white">
    Host Homes
  </a>
  <a href="/host/add-home" class="bg-blue-600 hover:bg-blue-700 text-white">
    Add Home
  </a>
<% } %>
```



## 18.3 Checking Login State

3.

```
res.render("host/edit-home", {  
  home: home,  
  editing: editing,  
  pageTitle: "Edit Your Home",  
  isLoggedIn: req.isLoggedIn,  
});
```



## 18.3 Checking Login State

4.

```
app.use(storeRouter);
app.use("/host", (req, res, next) => {
  if (!req.isLoggedIn) {
    return res.redirect("/login");
  }
  next();
});
app.use("/host", hostRouter);
```



## 18.4 Using a Cookie

1. Understand why a global variable would not work.
2. Set a cookie on successful login. See it in storage, also on the next request.
3. Read the cookie using syntax and Define a middleware to set this value to the request object.

```
console.log(req.get('Cookie').split('=')[1]);
```

4. Change the Cookie from the browser and see the result.



## 18.4 Using a Cookie

2.

```
exports.postLogin = (req, res, next) => {
  console.log(req.body);
  res.cookie("isLoggedIn", true);
  res.redirect("/");
};
```

3.

```
app.use((req, res, next) => {
  req.isLoggedIn = req.get('Cookie')?.split('=')[1] || false;
  console.log(req.isLoggedIn);
  next();
});
```



## 18.4 Using a Cookie

4.

The screenshot shows the Chrome DevTools Application tab open. On the left, the Storage section is expanded, showing Local storage, Session storage, IndexedDB, and Cookies. The Cookies section is also expanded, showing a cookie for the URL `http://localhost:3001`. The main table in the Application tab lists a single cookie:

Name	Value
isLoggedIn	false



## 18.5 Define the Logout Feature

1. Define a **logout button** in **nav bar** that should come only when user is logged in. Button should be a **form** that submits to link /logout.
2. Hide the **login** button in case user is logged in.
3. Handle the **logout path** and set the **isLoggedIn** cookie to false.



# 18.5 Define the Logout Feature

1, 2.

```
<div class="flex space-x-4">
  <% if (!isLoggedIn) { %>
    <a href="/login" class="bg-blue-600 hover:bg-blue-700 text-white p-2 rounded">
      Login
    </a>
  <% } else { %>
    <form action="/logout" method="POST">
      <button type="submit" class="bg-red-600 hover:bg-red-700 text-white p-2 rounded">
        Logout
      </button>
    </form>
  <% } %>
</div>
```



## 18.5 Define the Logout Feature

3.

```
exports.postLogout = (req, res, next) => {
  res.cookie("isLoggedIn", false);
  res.redirect("/login");
};
```



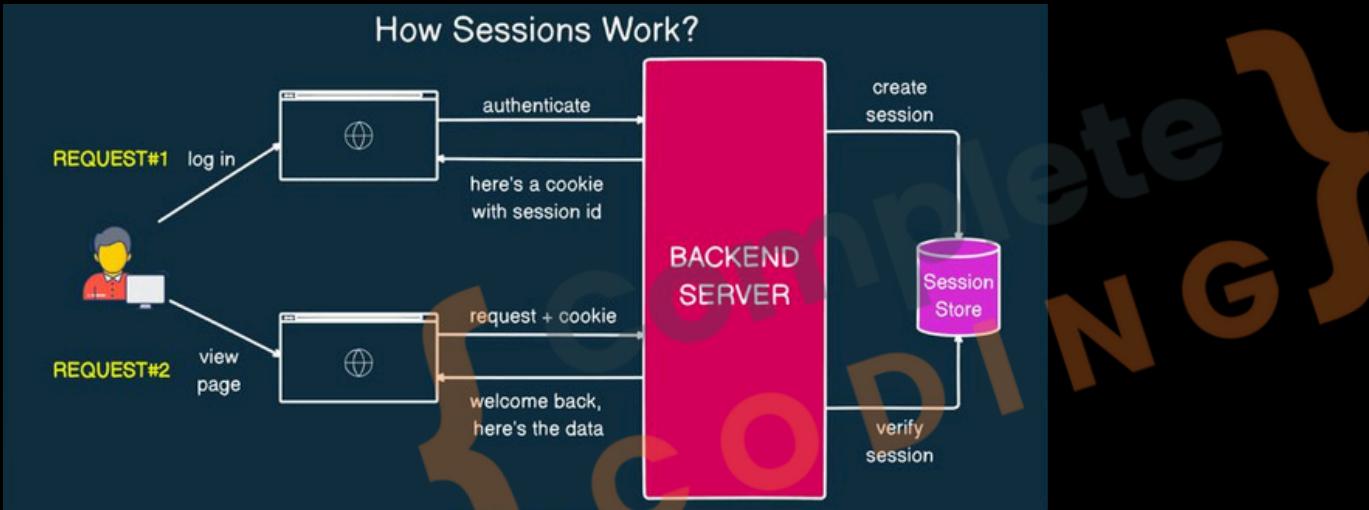
## 18.6 Problem with Cookies

1. Cookies can be intercepted or stolen, posing security risks.
2. They have limited storage capacity (about 4KB).
3. Users can delete or modify cookies, leading to data loss or tampering.
4. Data in cookies is not encrypted, making sensitive information vulnerable.
5. Storing important info in cookies exposes it to client-side attacks.





## 18.7 What are Sessions



1. Sessions are **server-side storage** mechanisms that track user interactions with a website.
2. They maintain user state and data across multiple requests in a web application.
3. Sessions enable **persistent user experiences** by maintaining state between the client and server over **stateless HTTP**.



# 18.8 Installing Session Package

Pro Teams Pricing Documentation

npm Search packages Sign Up Sign In

**express-session** 1.18.1 • Public • Published a month ago

Readme Code Beta 8 Dependencies 5,041 Dependents 65 Versions

**express-session**

npm v1.18.1 downloads 7.8M/month CI success coverage 100%

## Installation

This is a **Node.js** module available through the [npm registry](#). Installation is done using the `npm install` command:

```
$ npm install express-session
```

## API

```
var session = require('express-session')
```

Install `npm i express-session`

Repository [github.com/expressjs/session](https://github.com/expressjs/session)

Homepage [github.com/expressjs/session#readme](https://github.com/expressjs/session#readme)

Weekly Downloads 1,829,979

Version 1.18.1 License MIT

The page features a large watermark reading "COMPLETED CODING" in grey and orange.



# 18.8 Installing Session Package

```
prashantjain@Prashants-MacBook-Pro:~/Desktop/mongoose$ npm install express-session
added 6 packages, and audited 264 packages in 2s
48 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

```
const session = require('express-session');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(session({
  // Secret key used to sign the session ID cookie and encrypt session data
  secret: 'Complete Coding Secret',
  // Forces session to be saved back to the session store, even if not modified
  resave: false,
  // Forces a session that is "uninitialized" to be saved to the store
  saveUninitialized: true,
}));
```



## 18.9 Creating a Session

1. Sensitive info is stored on server.
2. Same session is valid for all requests from one user, using cookies.
3. Remove setting the cookie and now save the flag in session.
4. Check the browser for cookie changes.
5. Log Session in some get request.
6. Try to use a different browser and show that session is different.
7. Sessions are stored in memory so they reset when server restarts.



# 18.9 Creating a Sessions

3.

```
exports.postLogin = (req, res, next) => {
  console.log(req.body);
  req.session.isLoggedIn = true;
  res.redirect("/");
};
```

4.

The screenshot shows the Chrome DevTools Application tab. On the left, there are three sections: Manifest, Service workers, and Storage. The Storage section is expanded, showing a table of session data. The table has columns for Name, Value, and Do... (with a dropdown menu). There are two rows: one for 'connect.sid' with a long value starting 's%3Ao1svY1dKn7Pltaijb0U...' and another for 'isLoggedIn' with the value 'false'.

Name	Value	Do...
connect.sid	s%3Ao1svY1dKn7Pltaijb0U...	loc...
isLoggedIn	false	loc...



## 18.9 Creating a Sessions

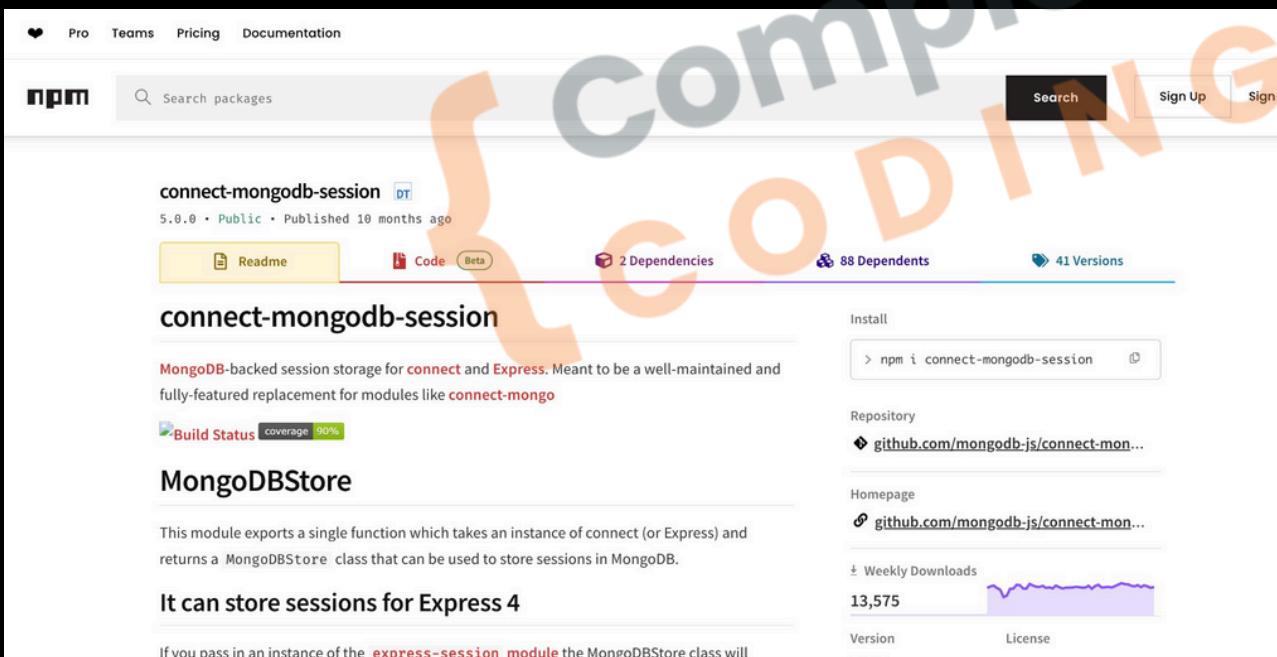
5. `exports.getIndex = (req, res, next) => {  
 console.log(req.session, req.session.isLoggedIn);  
 Home.find().then((registeredHomes) => {`

```
Session {  
  cookie: { path: '/', _expires: null, originalMaxAge: null, httpOnly: true },  
  isLoggedIn: true  
} true
```



# 18.10 Saving Session in DB

★ 176 **connect-mongodb-session** Lightweight MongoDB-based session store built and maintained by MongoDB.

A screenshot of the npm package page for `connect-mongodb-session`. The page includes the package name, version (5.0.0), license (MIT), and repository URL (`github.com/mongodb-js/connect-mongodb-session`). A large, semi-transparent watermark reading "Complete Coding" is overlaid across the center of the page. The page also displays the number of dependencies (2 and 88) and versions (41). Below the header, there's a brief description of the package: "MongoDB-backed session storage for `connect` and `Express`. Meant to be a well-maintained and fully-featured replacement for modules like `connect-mongo`". At the bottom, there's a section titled "MongoDBStore" with a description of its functionality and a note that it can store sessions for Express 4.

connect-mongodb-session

5.0.0 • Public • Published 10 months ago

Readme Code Beta 2 Dependencies 88 Dependents 41 Versions

## connect-mongodb-session

MongoDB-backed session storage for `connect` and `Express`. Meant to be a well-maintained and fully-featured replacement for modules like `connect-mongo`.

 coverage 90%

### MongoDBStore

This module exports a single function which takes an instance of `connect` (or `Express`) and returns a `MongoDBStore` class that can be used to store sessions in MongoDB.

### It can store sessions for Express 4

If you pass in an instance of the `express-session` module the `MongoDBStore` class will



# 18.10 Saving Session in DB

```
prashantjain@Prashants-MacBook-Pro ~ % npm install connect-mongodb-session  
added 3 packages, and audited 267 packages in 2s  
  
48 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

```
const MongoDBStore = require('connect-mongodb-session')(session);  
  
const MONGO_DB_URL = "mongodb+srv://root:root@kgcluster.ie6mb.mongodb.net/  
  
const store = new MongoDBStore({  
  uri: MONGO_DB_URL,  
  collection: 'sessions'  
});  
  
app.use(session({  
  // Secret key used to sign the session ID cookie and encrypt session data  
  secret: 'Complete Coding Secret',  
  // Forces session to be saved back to the session store, even if not modified  
  resave: false,  
  // Forces a session that is "uninitialized" to be saved to the store  
  saveUninitialized: true,  
  store: store,  
}));
```



# Practise Milestone

Take your airbnb forward:

- Cleanup cookie code to use session everywhere.
- Remove the cookie middleware.
- Destroy the session on logout.
- Cleanup Logs.
- Understand why leaving the cookie on logout is fine.





# Practise Milestone (Solution)

SEARCH

req.isLoggedIn Aa ab \*

req.session.isLoggedIn AB ...

files to include  
./5 NodeJs and ExpressJs/13 mongoose

files to exclude

11 results in 5 files - Open in editor

- ✓ JS app.js 5 NodeJs and ExpressJs/13 mo... M 1  
if (!req.isLoggedIn || req.session.isLoggedIn) {
- ✓ JS authController.js 5 NodeJs and Expre... U 1  
req.isLoggedIn || req.session.isLoggedIn = true;
- ✓ JS errorController.js 5 NodeJs and Expr... M 1  
...Page Not Found', isLoggedIn: req.isLoggedIn r...
- ✓ JS hostController.js 5 NodeJs and Expre... M 3  
const isLoggedIn = req.isLoggedIn || req.session.i...  
isLoggedIn: req.isLoggedIn || req.session.isLogge...  
isLoggedIn: req.isLoggedIn || req.session.isLogge...
- ✓ JS storeController.js 5 NodeJs and Expr... M 5  
console.log(req.session, req.isLoggedIn || req.ses...  
isLoggedIn: req.isLoggedIn || req.session.isLogge...  
isLoggedIn: req.isLoggedIn || req.session.isLogge...  
isLoggedIn: req.isLoggedIn || req.session.isLogge...  
isLoggedIn: req.isLoggedIn || req.session.isLogge...

```
exports.postLogout = (req, res, next) => {
  req.session.destroy(() => {
    res.redirect("/login");
  });
};
```

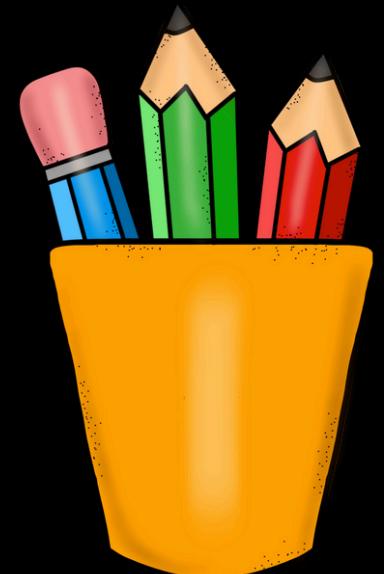
Complete CODING





# Revision

1. What are Cookies
2. Adding Login Functionality
3. Checking Login State
4. Using a Cookie
5. Define the Logout Feature
6. Problem with Cookies
7. What are Sessions
8. Installing Session Package
9. Creating a Sessions
10. Saving Session in DB





{ complete  
C O D I N G }

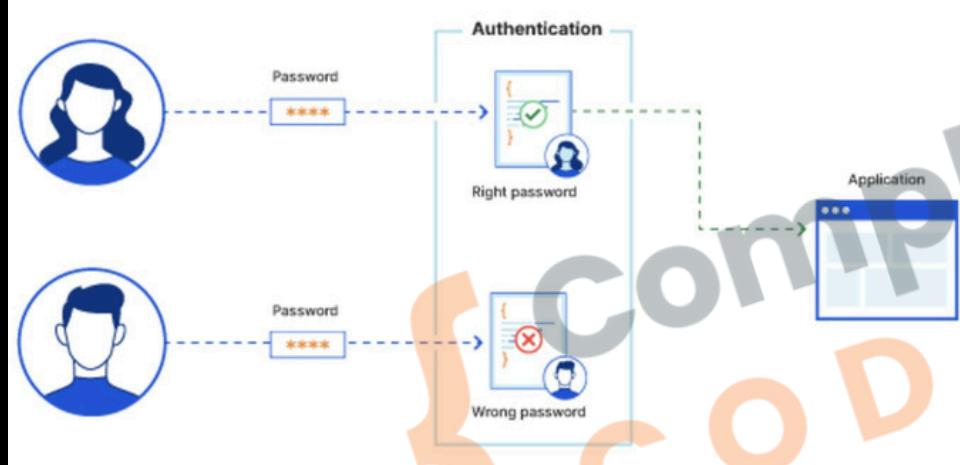


# 19. Authentication & Authorization

- 1.What is Authentication
- 2.What is Authorization
- 3.Session based Authentication
- 4.Authentication vs Authorization
- 5.Signup UI
- 6.Using Express Validator
- 7.Adding User Model
8. Encrypting Password
- 9.Implementing Login
- 10.Adding User Functions



# 19.1 What is Authentication

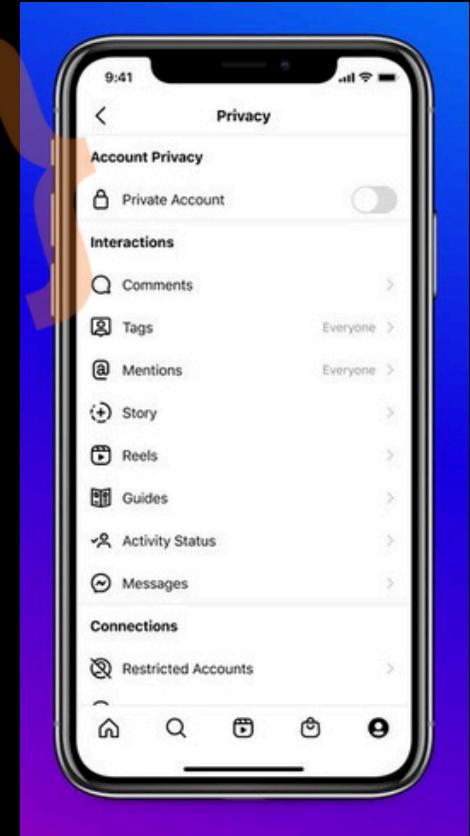


1. Authentication is the process of verifying the identity of a user or system accessing an application.
2. It ensures that only authorized users can access protected resources and features.
3. Authentication is crucial for security, protecting data, and providing personalized experiences in web applications.



## 19.2 What is Authorization

1. Authorization is the process of determining what actions a user is permitted to perform within an application.
2. It ensures that users can access only the resources and functionalities they have permission for.
3. Authorization enhances security by restricting access to sensitive data and operations, complementing the authentication process.

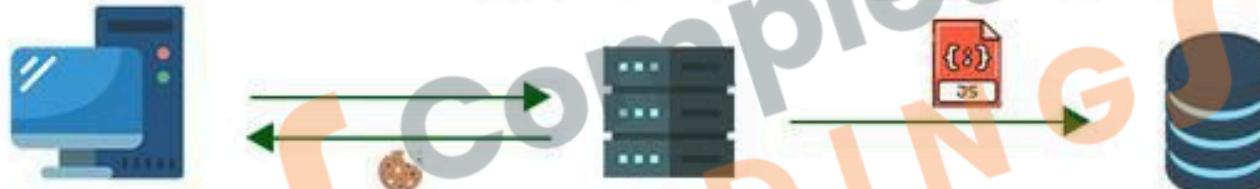




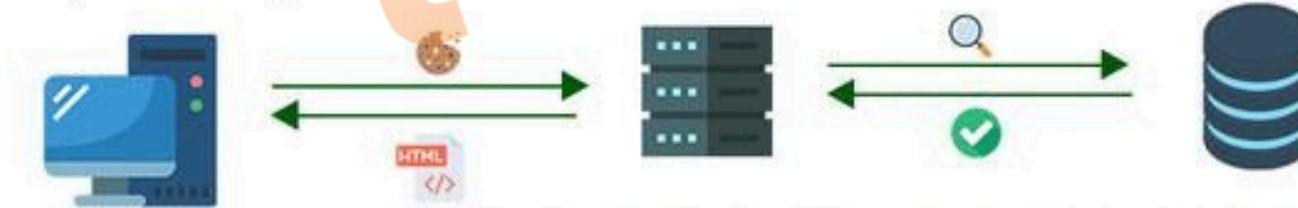
# 19.3 Session based Authentication

The user sends login request

The server authorizes the login, sends a session to the database, and returns a cookie containing the session ID to the user



The user sends new request  
(with a cookie)

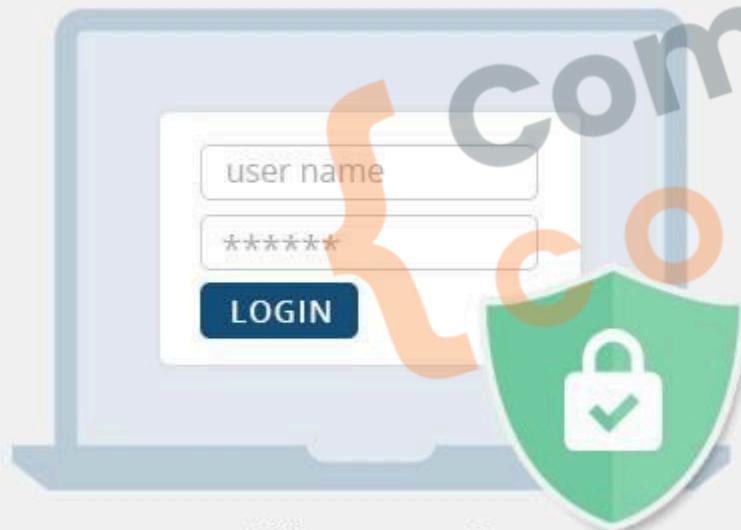


The server looks up in the database for the ID Found in the cookie, if the ID is found it sends the requested pages to the user



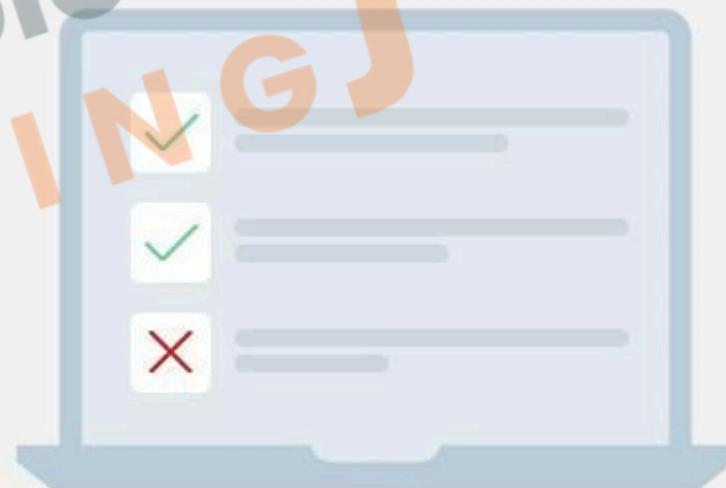
# 19.4 Authentication vs Authorization

## Authentication



Who are you?  
Validate a system is accessing by the right person

## Authorization



Are you allowed to do that?  
Check users' permissions to access data

completo  
CODING

# 19.4 Authentication vs Authorization

Aspect	Authentication	Authorization
Definition	Verifies the identity of a user or system	Determines what resources a user can access
Purpose	Ensures users are who they claim to be	Grants or denies permissions to resources and actions
Process	Validates credentials like usernames and passwords	Checks user privileges and access levels
Occurs When	At the start of a session or when accessing secured areas	After authentication, during resource access
User Interaction	Requires user input (e.g., logging in)	Usually transparent unless access is denied
Managed By	Handled by both frontend and backend systems	Mainly enforced by backend servers
Example	User logs into an account with a password	User accesses settings page only if they have admin rights



## 19.5 Signup UI

1. Define a **signup button** in navigation bar **along with sign-in**. It should point to a link `/signup`
2. Define a `auth/signup.ejs` file that has the following fields and submits **POST** request to `/signup`:
  - a. `firstName`, `lastName`,
  - b. `email`
  - c. `password`, `confirmPassword`
  - d. `userType` with possible values 'guest', 'host'
  - e. Terms and conditions checkbox.
3. Define routes in `authRouter` and behaviour in `authController`.
4. Fix the UI of the app to look pretty.



# 19.5 Signup UI

```
1.  <% if (!isLoggedIn) { %>
    <a href="/signup" class="bg-green-600 hover:bg-green-700 text-white font-semibold py-2.5 px-6
    rounded-lg transition duration-300 ease-in-out transform hover:scale-105 shadow-md">
        | Sign Up
    </a>
    <a href="/login" class="bg-blue-600 hover:bg-blue-700 text-white font-semibold py-2.5 px-6
    rounded-lg transition duration-300 ease-in-out transform hover:scale-105 shadow-md">
        | Login
    </a>
<% } else { %>
```



# 19.5 Signup UI

2.

```
<form action="/signup" method="POST" class="w-full max-w-md">
  <input type="text" name="firstName" placeholder="First Name" class="w-full px-4 py-2 mb-4 border
    border-gray-300 rounded-lg focus:outline-none focus:border-blue-500"/>
  <input type="text" name="lastName" placeholder="Last Name" class="w-full px-4 py-2 mb-4 border
    border-gray-300 rounded-lg focus:outline-none focus:border-blue-500"/>
  <input type="email" name="email" placeholder="Enter your Email" class="w-full px-4 py-2 mb-4 border
    border-gray-300 rounded-lg focus:outline-none focus:border-blue-500"/>
  <input type="password" name="password" placeholder="Password" class="w-full px-4 py-2 mb-4 border
    border-gray-300 rounded-lg focus:outline-none focus:border-blue-500"/>
  <input type="password" name="confirm_password" placeholder="Confirm Password" class="w-full px-4 py-2
    mb-4 border border-gray-300 rounded-lg focus:outline-none focus:border-blue-500"/>

  <div class="mb-4">
    <p class="text-gray-700 mb-2">User Type:</p>
    <div class="flex space-x-4">
      <label class="flex items-center"><input type="radio" name="userType" value="guest" class="mr-2"/>
        <span class="text-gray-700">Guest</span></label>
      <label class="flex items-center"><input type="radio" name="userType" value="host" class="mr-2"/>
        <span class="text-gray-700">Host</span></label>
    </div>
  </div>

  <div class="mb-4">
    <label class="flex items-center"><input type="checkbox" name="termsAccepted" class="mr-2"/><span
      class="text-gray-700">I accept the terms and conditions</span></label>
  </div>

  <div class="flex justify-center">
    <input type="submit" value="Sign me up" class="bg-green-500 hover:bg-green-600 text-white
      font-semibold py-2 px-4 rounded-lg transition duration-300 ease-in-out transform hover:scale-105
      cursor-pointer">
  </div>
</form>
```



## 19.5 Signup UI

3, 4.

```
authRouter.post("/logout", authController.postLogout);
authRouter.get("/signup", authController.getSignup);

exports.getSignup = (req, res, next) => {
  res.render("auth/signup", { pageTitle: "Sign Up", isLoggedIn: false });
};
```



# 19.6 Using Express Validator

1. Add handling for POST /signup in auth controller and router.
2. Install Express Validator.
3. Use the email and password validations in the post handler.
4. Change the signup.ejs to show the errors. And accept the old values.

Version: 7.2.0

## express-validator

### Overview

express-validator is a set of express.js middlewares that wraps the extensive collection of validators and sanitizers offered by validator.js.

It allows you to combine them in many ways so that you can validate and sanitize your express requests, and offers tools to determine if the request is valid or not, which data was matched according to your validators, and so on.



# 19.6 Using Express Validator

1.

```
authRouter.get("/signup", authController.getSignup);
authRouter.post("/signup", authController.postSignup);
```

```
exports.postSignup = (req, res, next) => {
  console.log(req.body);
  res.redirect("/login");
};
```

2.

```
prashantjain@Prashants-MacBook-Pro:~/Desktop$ npm install express-validator
added 2 packages, and audited 269 packages in 553ms
48 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```



# 19.6 Using Express Validator

3.

```
exports.postSignup = [
  // First Name validation
  check('firstName')
    .notEmpty()
    .withMessage('First name is required')
    .trim()
    .isLength({ min: 2 })
    .withMessage('First name must be at least 2 characters long')
    .matches(/^[a-zA-Z\s]+$/)
    .withMessage('First name can only contain letters'),

  // Last Name validation
  check('lastName')
    .notEmpty()
    .withMessage('Last name is required')
    .trim()
    .isLength({ min: 2 })
    .withMessage('Last name must be at least 2 characters long')
    .matches(/^[a-zA-Z\s]+$/)
    .withMessage('Last name can only contain letters'),

  // Email validation
  check('email')
    .isEmail()
    .withMessage('Please enter a valid email')
    .normalizeEmail(),

  // Password validation
  check('password')
    .isLength({ min: 8 })
    .withMessage('Password must be at least 8 characters long')
    .matches(/[^a-z]/)
    .withMessage('Password must contain at least one lowercase letter')
    .matches(/[^A-Z]/)
    .withMessage('Password must contain at least one uppercase letter')
    .matches(/[^@#$%^&*(.,?":{}|<>)/])
    .withMessage('Password must contain at least one special character')
    .trim(),

  // Confirm password validation
  check('confirm_password')
    .trim()
    .custom((value, { req }) => {
      if (value !== req.body.password) {
        throw new Error('Passwords do not match');
      }
      return true;
    }),

  // User Type validation
  check('userType')
    .notEmpty()
    .withMessage('User type is required')
    .isIn(['guest', 'host'])
    .withMessage('Invalid user type'),

  // Terms Accepted validation
  check('termsAccepted')
    .notEmpty()
    .withMessage('You must accept the terms and conditions')
    .custom((value) => {
      if (value !== 'on') {
        throw new Error('You must accept the terms and conditions');
      }
      return true;
    })
];
```



# 19.6 Using Express Validator

```
3. // Final handler middleware
(req, res, next) => {
  const { firstName, lastName, email, password, userType } = req.body;
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(422).render('auth/signup', {
      pageTitle: 'Sign Up',
      isLoggedIn: false,
      errorMessages: errors.array().map(error => error.msg),
      oldInput: {
        firstName,
        lastName,
        email,
        password,
        userType
      }
    });
  }

  res.redirect('/login');
}
```



# 19.6 Using Express Validator

```
4. <% if (typeof errorMessages !== 'undefined' && errorMessages && errorMessages.length > 0) { %>
  <div class="■bg-red-100 border ■border-red-400 ■text-red-700 px-4 py-3 rounded relative mb-4"
    role="alert">
    <ul class="list-disc list-inside space-y-1">
      <% errorMessages.forEach(error => { %>
        <li><%= error %></li>
      <% }); %>
    </ul>
  </div>
<% } %>

<input
  type="text"
  name="firstName"
  placeholder="First Name"
  value="<%= typeof oldInput !== 'undefined' ? oldInput.firstName : '' %>"
  class="w-full px-4 py-2 mb-4 border ■border-gray-300 rounded-lg focus:outline-none
  ■focus:border-blue-500"
/>
```



## 19.7 Adding User Model

1. Define a new **User Model** with following fields:
  - a. `firstName & lastName` (required)
  - b. `email` (required, unique)
  - c. `Password` (required)
  - d. `userType` (possible values 'guest', 'host')
2. In the **POST** signup handler, create a new user with the fields from request and redirect to **login** after saving the user.



# 19.7 Adding User Model

1.

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

const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true
  },
  lastName: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  userType: {
    type: String,
    required: true,
    enum: ['guest', 'host']
  }
});

module.exports = mongoose.model('User', userSchema);
```

{ Complete Coding }



## 19.7 Adding User Model

2.

```
const user = new User({
  firstName: firstName,
  lastName: lastName,
  email: email,
  password: password,
  userType: userType
});

user.save()
  .then(() => {
    res.redirect('/login');
})
  .catch(err => {
    console.log("Error while creating user: ", err);
});
```

The slide features a large, semi-transparent watermark in the center-right area. It contains the word "complete" in a light blue, sans-serif font, followed by "CODING" in a larger, bold brown font. A large brown curly brace is positioned behind the word "complete", spanning most of the width of the text area.



# 19.8 Encrypting Password

- 1.Understand the problem with plain text passwords.
- 2.Install a package named: bcryptjs
- 3.Hash the password before saving password.
- 4.Understand that even server does not have the password.



## 19.8 Encrypting Password

2. prashantjain@Prashants-MacBook-Pro 13 mongoose % npm install bcryptjs  
added 1 package, and audited 270 packages in 513ms  
48 packages are looking for funding  
run `npm fund` for details  
found 0 vulnerabilities



# 19.8 Encrypting Password

```
3. bcrypt.hash(password, 12).then(hashedPassword => {
  const user = new User({
    firstName: firstName,
    lastName: lastName,
    email: email,
    password: hashedPassword,
    userType: userType
  });

  user.save()
    .then(() => {
      res.redirect('/login');
    })
    .catch(err => {
      console.log("Error while creating user: ", err);
    });
});
```



## 19.9 Implementing Login

1. Read the **email** and **password** from the request body and find the user with the **email** from the **Users** collection.
2. If the user **is not found** send an error and re-render the login page, also make changes to the **login** page to show errors.
3. If the user **is found**, then **use the bcrypt compare function** to match the entered password, If password does not match **send another error**, otherwise **create a login session** and **redirect user to the home**.



# 19.9 Implementing Login

1.

```
exports.postLogin = async (req, res, next) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email: email });

    if (!user) {
      return res.render('auth/login', {
        pageTitle: 'Login',
        isLoggedIn: false,
        errorMessage: 'Invalid Email'
      });
    }

    res.redirect("/");
  } catch (err) {
    console.log("Error while logging in: ", err);
  }
};
```



# 19.9 Implementing Login

2.

```
<% if (typeof errorMessage !== 'undefined' && errorMessage) { %>
  <div class="■bg-red-100 border ■border-red-400 ■text-red-700 px-4 py-3 rounded relative mb-4"
    role="alert">
    <span class="block sm:inline"><%= errorMessage %></span>
  </div>
<% } %>
```

A large, semi-transparent watermark is positioned diagonally across the slide. It contains the words "Complete" and "CODING" in a stylized, blocky font. The "C" and "O" in "CODING" are particularly prominent and slanted upwards towards the top right corner of the slide.



# 19.9 Implementing Login

3.

```
const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) {
  return res.render('auth/login', {
    pageTitle: 'Login',
    isLoggedIn: false,
    errorMessage: 'Invalid Password'
  });
}

req.session.isLoggedIn = true;
req.session.user = user;
await req.session.save();

res.redirect("/");
} catch (err) {
  console.log("Error while logging in: ", err);
}
```



# 19.10 Adding User Functions

1. Make the navigation bar items display only on the basis of userType by passing the user object to all views.
2. Add a field in User Model names favouriteHomes, which is an array of home ids.
3. Remove the Favourite Model and the Pre hook from the Home Model.
4. Make the favourite user specific and change the following methods:
  - a. postAddFavourites
  - b. getFavourites
  - c. postRemoveFavourite



# 19.10 Adding User Functions

1.

```
<% if (isLoggedIn) { %>
<% if (user.userType === 'guest') { %>
  <a href="/homes" class="■bg-blue-600 ■hover:bg-blue-700 ■text-white
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
    | Homes
  </a>
  <a href="/favourites" class="■bg-blue-600 ■hover:bg-blue-700 ■text-white
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
    | Favourites
  </a>
<% } %>
<% if (user.userType === 'host') { %>
  <a href="/host/host-homes" class="■bg-blue-600 ■hover:bg-blue-700 ■text-wh
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
    | Host Homes
  </a>
  <a href="/host/add-home" class="■bg-blue-600 ■hover:bg-blue-700 ■text-whit
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
    | Add Home
  </a>
<% } %>
```

isLoggedIn: req.session.isLoggedIn,  
user: req.session.user,



# 19.10 Adding User Functions

```
2. userType: {  
    type: String,  
    required: true,  
    enum: ['guest', 'host']  
},  
favouriteHomes: [  
    type: mongoose.Schema.Types.ObjectId,  
    ref: 'Home'  
}]
```

The slide features a large, semi-transparent watermark in the center-right area. It contains the word "Coding" in a stylized, rounded font, where each letter is partially enclosed by a brown bracket-like shape. Above "Coding", the word "Complete" is written in a smaller, lighter gray font.



# 19.10 Adding User Functions

4.a.

```
exports.postAddFavourites = (req, res, next) => {
  const homeId = req.body.id;
  const userId = req.session.user._id;

  User.findById(userId)
    .then(user => {
      if (!user.favouriteHomes.includes(homeId)) {
        user.favouriteHomes.push(homeId);
        return user.save();
      }
      return user;
    })
    .then(() => {
      res.redirect("/favourites");
    })
    .catch((err) => {
      console.log("Error while adding to favourites", err);
      res.redirect("/favourites");
    });
};
```



# 19.10 Adding User Functions

```
4.b. exports.getFavourites = (req, res, next) => {
    const userId = req.session.user._id;
    User.findById(userId)
        .populate('favouriteHomes')
        .then((user) => {
            res.render("store/favourites", {
                homes: user.favouriteHomes,
                pageTitle: "Favourites",
                isLoggedIn: req.session.isLoggedIn,
                user: req.session.user,
            });
        });
};
```



# 19.10 Adding User Functions

4.c.

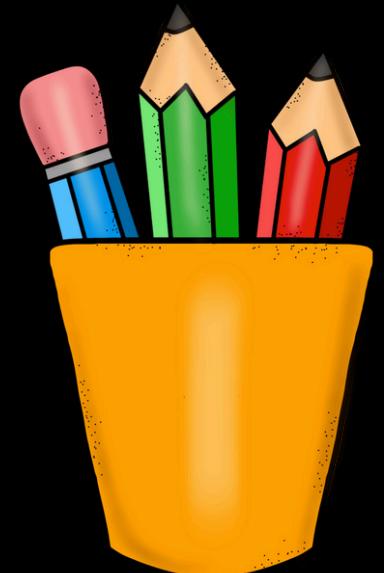
```
exports.postRemoveFavourite = (req, res, next) => {
  const homeId = req.params.homeId;
  const userId = req.session.user._id;

  User.findById(userId)
    .then(user => {
      user.favouriteHomes = user.favouriteHomes.filter(id => id.toString()
        !== homeId);
      return user.save();
    })
    .then(() => {
      res.redirect("/favourites");
    })
    .catch(error => {
      console.log("Error while remove from favourites ", error);
      res.redirect("/favourites");
    });
};
```



# Revision

1. What is Authentication
2. What is Authorization
3. Session based Authentication
4. Authentication vs Authorization
5. Signup UI
6. Using Express Validator
7. Adding User Model
8. Encrypting Password
9. Implementing Login
10. Adding User Functions





{ complete  
C O D I N G }



# 20. File Upload & Download

1. Adding a File Picker
2. Creating Multipart Form
3. Handling Multipart Form Data
4. Saving Image Files
5. Custom File Names
6. Restricting Upload File Types
7. Handling Edits
8. Serving Saved Data
9. Serving Files after Auth
10. Deleting Files





# 20.1 Adding a File Picker

● **Input Element:** Use `<input type="file">` to create a file picker.

● **Multiple Files:** Add `multiple` attribute to allow selecting multiple files.

● **File Types:** Use `accept` attribute to restrict file types (e.g., `accept=".jpg, .png"`).

```
<input type="file"  
      name="photo"  
      class="w-full px-4 py-2 mb-4 border border-gray-300  
      rounded-lg focus:outline-none focus:border-blue-500"  
/>
```

The screenshot shows a user interface for adding a home listing. At the top, the title "Add your Home" is displayed. Below it are four input fields with placeholder text: "Name of your house", "Daily rent of your home", "Where is your home", and "Rating of the house". A fifth input field is labeled "Choose File" with the message "No file chosen". Below these is a larger text area labeled "Describe your house". At the bottom right is a blue button labeled "Add Home". A large, semi-transparent watermark reading "COMPLETED CODING" is overlaid across the center of the form.



# 20.2 Creating Multipart Form

```
exports.postAddHome = (req, res, next) => {
  const { houseName, price, location, rating, photoUrl, description } = req.body;
  console.log(req.body);
```

```
Server running at: http://localhost:3001
{
  id: '',
  houseName: 'New House',
  price: '1299',
  location: 'kashmir',
  rating: '3.9',
  photo: 'IMG_4705.HEIC',
  description: 'asdfasdfs'
}
```

Request URL: http://localhost:3001/host/edit-home  
Request Method: POST  
Status Code: 302 Found  
Remote Address: [:1]:3001  
Referrer Policy: strict-origin-when-cross-origin

Response Headers (8)

Header	Value
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding	gzip, deflate, br, zstd
Accept-Language	en-GB,en;q=0.9
Cache-Control	max-age=0
Connection	keep-alive
Content-Length	114
Content-Type	application/x-www-form-urlencoded
Cookie	connect.sid=s%3AJHZ1M5GAnP9QuJSwuUTfVrbHSq6HBC7
Host	localhost:3001
Origin	http://localhost:3001



# 20.3 Handling Multipart Form Data

npm Search packages Sign Up Sign In

**Multer** DT  
1.4.5-lts.1 • Public • Published 2 years ago

[Readme](#) [Code](#) Beta [7 Dependencies](#) [4,749 Dependents](#) [46 Versions](#)

Multer is a node.js middleware for handling `multipart/form-data`, which is primarily used for uploading files. It is written on top of `busboy` for maximum efficiency.

**NOTE:** Multer will not process any form which is not `multipart/multipart/form-data`.

### Translations

This README is also available in other languages:

- [Español](#) (Spanish)
- [简体中文](#) (Chinese)
- [한국어](#) (Korean)
- [Русский язык](#) (Russian)
- [Việt Nam](#) (Vietnam)
- [Português](#) (Portuguese Brazil)

### Installation

```
$ npm install --save multer
```

Install  
`> npm i multer`

Repository [github.com/expressjs/multer](https://github.com/expressjs/multer)

Homepage [github.com/expressjs/multer#readme](https://github.com/expressjs/multer#readme)

Weekly Downloads 6,313,138

Version 1.4.5-lts.1 License MIT

Unpacked Size 27.6 kB Total Files 11

Issues 195 Pull Requests 70

prashantjain@Prashants-MacBook-Pro:~/16\_email\_and\_advance\_auth % `npm install multer`  
added 17 packages, and audited 298 packages in 719ms  
50 packages are looking for funding  
run `'npm fund'` for details  
1 **high** severity vulnerability  
To address all issues, run:  
`npm audit fix`  
Run `'npm audit'` for details.



# 20.3 Handling Multipart Form Data

```
<form action="/host/<%= editing ? "edit-home" : "add-home"%>" method="POST" enctype="multipart/form-data"
class="w-full max-w-md">

const multer = require('multer');

app.use(express.urlencoded());
app.use(multer().single('photo'));
app.use(express.static(path.join(rootDir, 'public')))

|   console.log(req.file);

{
 fieldname: 'photo',
originalname: 'IMG_4705.HEIC',
encoding: '7bit',
mimetype: 'image/heic',
buffer: <Buffer 00 00 00 1c 66 74 79 70 68 65 69 63 00 00 00 00 74 6d
00 00 21 68 64 6c 72 00 00 ... 2246669 more bytes>,
size: 2246719
}
```



# 20.4 Saving Image Files

```
app.use(multer({ dest: 'uploads/' }).single('photo'));  
  
{  
 fieldname: 'photo',  
originalname: 'house1.png',  
encoding: '7bit',  
mimetype: 'image/png',  
destination: 'uploads/',  
filename: '0c710a9b2903e323fc38e7926650d159',  
path: 'uploads/0c710a9b2903e323fc38e7926650d159',  
size: 387102  
}
```

```
> 🌐 routers  
└─ uploads  
    └─ 0c710a9b2903e323fc38e7926650d159  
└─ util
```



# 20.5 Custom File Names

```
// This defines where uploaded files will be stored and how they'll be named
const storage = multer.diskStorage({
  // Set the destination folder for uploaded files
  destination: (req, file, cb) => {
    | cb(null, 'uploads/'); // Files will be saved in the 'uploads' directory
  },
  // Set the filename for uploaded files
  filename: (req, file, cb) => {
    | cb(null, new Date().toISOString() + '-' + file.originalname);
  }
});

app.use(multer({ storage }).single('photo'));
```

```
> routers
└─ uploads
  └─ 0c710a9b2903e323fc38e7926650d159
    └─ 2024-11-25T11:26:17.582Z-house1.png
└─ util
```



# 20.6 Restricting Upload File Types

```
const fileFilter = (req, file, cb) => {
  if ([ 'image/jpeg', 'image/png', 'image/jpg' ].includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(null, false);
  }
};
```

```
app.use(multer({ storage, fileFilter }).single('photo'));
```

```
exports.postEditHome = (req, res, next) => {
  const { id, houseName, price, location, rating, photoUrl, description } =
    req.body;

  if (!req.file) {
    return res.status(400).send('No image provided');
  }
```

Adding this filter, now multer would not have the file if it does not match the type and req.file will just be undefined.



# 20.6 Restricting Upload File Types

```
if (!req.file) {  
  return res.status(400).send('No image provided');  
}  
  
const photoUrl = req.file.path;
```

\_id: ObjectId('67445aa6e25a19de45a1b82e')  
houseName : "New House"  
price : 1299  
location : "kashmir"  
rating : 3.9  
description : "asdfasdf"  
host : ObjectId('673cacd23e8557da8e22d9b1')  
\_\_v : 0  
photoUrl : "uploads/2024-11-25T11:35:47.086Z-house1.png"



# 20.7 Handling Edits

```
// business logic outside model
Home.findById(id)
  .then((existingHome) => {
    if (!existingHome) {
      console.log("Home not found for editing");
      return res.redirect("/host/host-homes");
    }
    existingHome.houseName = houseName;
    existingHome.price = price;
    existingHome.location = location;
    existingHome.rating = rating;
    if (req.file) {
      existingHome.photoUrl = req.file.path;
    }
    existingHome.description = description;
    return existingHome.save();
  })
```

While editing we can make sure that we only overwrite the photoUrl in case a new valid image was uploaded. And don't force the user to upload images each time they edit.



## 20.8 Serving Saved Data

```
app.use(express.static(path.join(rootDir, "public")));
app.use('/uploads', express.static(path.join(rootDir, "uploads")));
app.use(bodyParser.urlencoded({ extended: true }));
```

This is done as like in case of public, things inside public will be served like they are in root folder and the url does not have public in it. Here also either remove uploads/ from the image path, or add the qualifier.



# 20.9 Serving Files after Auth



## Property Details

### Description

Experience luxury living in this beautiful property. This home offers the perfect blend of comfort and style, situated in the prime location of goa. With its exceptional rating of 4.5/5, this property stands out as one of our finest offerings.

[Download House Rules](#)

[Add to Favourite](#)

### Features

- ✓ Prime Location
- ✓ Highly Rated
- ✓ Modern Amenities

```
<div class="mt-4">
  <a href="/rules/<%= home._id %>" class="inline-flex items-center text-blue-600 hover:text-blue-800">
    <svg class="w-5 h-5 mr-2" fill="none" stroke="currentColor" viewBox="0 0 24 24">
      <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M12 10v6m0 0l-3-3m3 3l3-3m2 8H7a2 2 0 01-2-2V5a2 2 0 012-2h5.586a1 1 0 01.707.293l5.414 5.414a1 1 0 01.293.707V19a2 2 0 01-2 2z"/>
    </svg>
    Download House Rules
  </a>
</div>
```



# 20.9 Serving Files after Auth

```
storeRouter.get("/rules/:homeId", storeController.getHouseRules);
```

```
const path = require('path');
const rootDir = require('../util/path-util');

exports.getHouseRules = [(req, res, next) => {
  if (!req.session.isLoggedIn) {
    return res.redirect("/login");
  }
  next();
},
(req, res, next) => {
  const homeId = req.params.homeId;
  // We can make it house specific after have homeId in file name like this: `House Rules-${homeId}.pdf`
  const rulesFileName = `House Rules.pdf`;
  const filePath = path.join(rootDir, 'rules', rulesFileName);
  // res.sendFile(filePath);
  // res.download(filePath, 'Rules.pdf');
}
];
```





# 20.10 Deleting Files

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

exports.deleteFile = (filePath) => {
  fs.unlink(filePath, (err) => {
    if (err) throw err;
  });
};

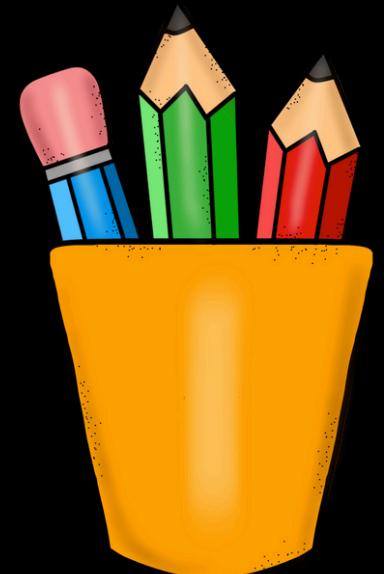
existingHome.location = location;
existingHome.rating = rating;
if (req.file) {
  deleteFile(existingHome.photoUrl);
  existingHome.photoUrl = req.file.path;
}
```





# Revision

1. Adding a File Picker
2. Creating Multipart Form
3. Handling multipart Data
4. Saving Image Files
5. Custom File Names
6. Restricting Upload File Types
7. Handling Edits
8. Serving Saved Data
9. Serving Files after Auth
10. Deleting Files





{ complete  
C O D I N G }

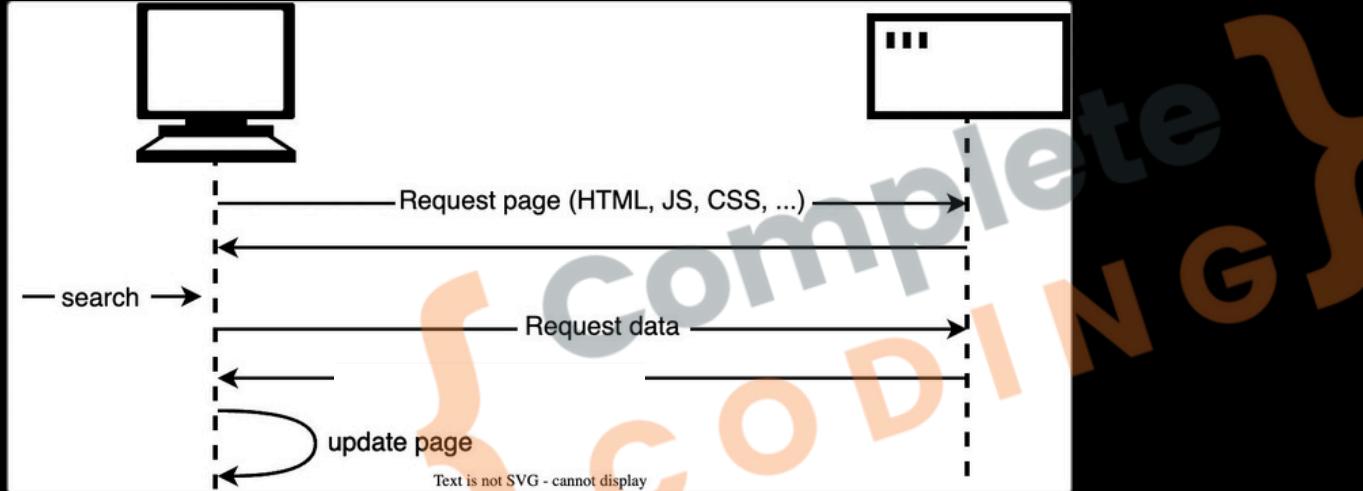


# 21. REST API / JSON Requests

1. What are **Async Requests**
2. What are **REST APIs**
3. Decoupling Frontend & Backend
4. Routes & HTTP Methods
5. REST Core Concepts
6. First API Todo App
7. API for Fetch Items
8. API for Deleting Items
9. Improving UI Elements
10. Adding Complete Item functionality
11. Deploying React App



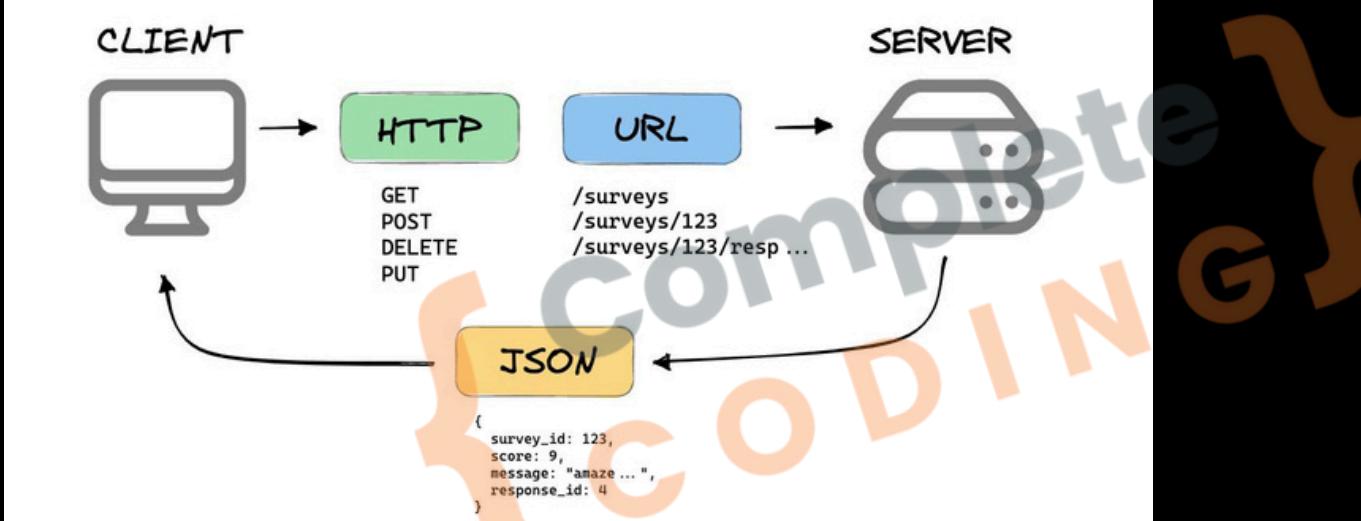
# 21.1 What are Async Requests



- Async network requests enable web pages to communicate with servers without reloading.
- The client sends JSON requests to the server asynchronously in single-page apps.
- The server processes the request and returns a JSON response.
- The page updates itself dynamically using the received data.



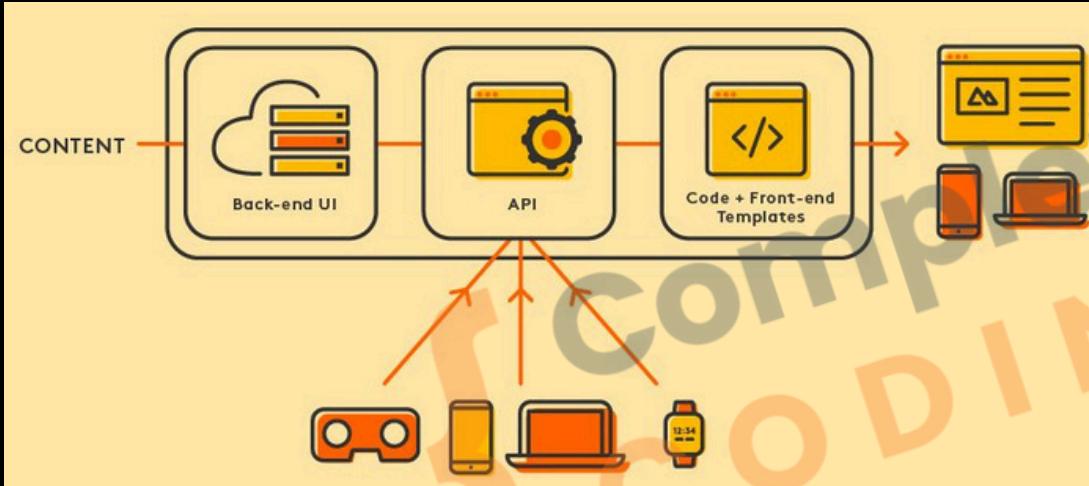
## 21.2 What are REST APIs



- REST APIs enable communication between clients and servers using HTTP.
- They are mainly identified by a URI.
- They use standard **HTTP** methods like **GET**, **POST**, **PUT**, and **DELETE**.
- Data is exchanged in formats like **JSON** or **XML**.
- REST APIs are stateless.
- REST APIs allow clients to access and manipulate web resources.



# 21.3 Decoupling Frontend & Backend



- Separating front-end and back-end allows independent development and scaling.
- REST APIs serve as a communication layer between them.
- Front-end interacts with back-end through standardized RESTful calls.
- Decoupling enhances flexibility and simplifies maintenance.
- REST APIs enable front-end updates without altering back-end code.



# 21.4 Routes & HTTP Methods

- REST API routes define the endpoints (URLs) where resources can be accessed by clients.
- GET: Retrieves data from the server at the specified route.
- POST: Sends new data to the server to create a resource.
- PUT: Updates or replaces an existing resource at a given route.
- DELETE: Removes a resource from the server at the specified route.
- PATCH: Partially updates an existing resource with new data.

superheroes		
GET	/api/dc/superheroes	Retrieves the collection of superheroes resources.
POST	/api/dc/superheroes	Creates a superheroes resource.
GET	/api/dc/superheroes/{id}	Retrieves a superheroes resource.
PUT	/api/dc/superheroes/{id}	Replaces the superheroes resource.
DELETE	/api/dc/superheroes/{id}	Removes the superheroes resource.
PATCH	/api/dc/superheroes/{id}	Updates the superheroes resource.



# 21.5 REST Core Concepts



- **Statelessness:** Each request contains all necessary information; the server maintains no client session.
- **Uniform Interface:** Standardized communication using HTTP methods like GET, POST, PUT, DELETE.
- **Client-Server Separation:** Independent development of front-end and back-end components.
- **Cacheability:** Responses indicate if they can be cached to improve performance.
- **Layered System:** Architecture allows for multiple layers between client and server.
- **Code on Demand (Optional):** Servers can extend client functionality by sending executable code.



# 21.6 First API Todo App

1. Copy the following from the previous app:

- a. Env files
- b. Package dependencies
- c. app.js

2. Changes in env files

- a. Remove the email data.
- b. Change DB name to todo-app

3. Create empty controllers, models, and routes folders.

4. Remove excess code from app.js.

5. Create the error controller to give out a 404 error message and status.

6. Create a Mongoose model for Todolitem.

7. Create itemsRouter and itemsController for POST /todos request from frontend.

8. Install and setup CORS package in backend.

9. Add express.json() to app.



# 21.6 First API Todo App

1.

```
.env.development  
.env.example  
.env.production  
app.js  
package.json
```

2.

```
backend > .env.development  
1 MONGO_DB_USERNAME=root  
2 MONGO_DB_PASSWORD=root  
3 MONGO_DB_DATABASE=todo-app-test|
```

```
backend > .env.production
```

```
1 MONGO_DB_USERNAME=root  
2 MONGO_DB_PASSWORD=root  
3 MONGO_DB_DATABASE=todo-app|
```

3.

```
> controllers  
> models  
> node_modules  
> routers
```



# 21.6 First API Todo App

4.

```
const ENV = process.env.NODE_ENV || 'production'
require('dotenv').config({
  path: `.${ENV}`
});

// External Module
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");

// Local Module
const errorController = require("./controllers/errorController");
const itemsRouter = require("./routers/itemsRouter");

const MONGO_DB_URL =
  `mongodb+srv://${process.env.MONGO_DB_USERNAME}:${process.env.MONGO_DB_PASSWORD}@kgcluster.ie6mb.mongodb.net/${process.env.MONGO_DB_DATABASE}`;

const app = express();

app.use(bodyParser.urlencoded({ extended: true }));
app.use(itemsRouter);
app.use(errorController.get404);

const PORT = process.env.PORT || 3000;
mongoose.connect(MONGO_DB_URL).then(() => {
  app.listen(PORT, () => {
    console.log(`Server running at: http://localhost:\${PORT}`);
  });
});
```



# 21.6 First API Todo App

backend > models > `TodoItem.js` > ...

```
6. 1 const mongoose = require("mongoose");
2
3 const todoItemSchema = new mongoose.Schema({
4   task: {
5     type: String,
6     required: true
7   },
8   date: {
9     type: Date,
10    required: true
11 },
12   completed: {
13     type: Boolean,
14     default: false
15 },
16   createdAt: {
17     type: Date,
18     default: Date.now
19 }
20});
```

5. backend > controllers > `errorController.js` > ...

```
1 exports.get404 = (req, res, next) => {
2   res.status(404).json({ message: 'Page Not Found' });
3 };
```

{ COMPLETED CODING }



# 21.6 First API Todo App

7.

```
const express = require("express");
const itemsController = require("../controllers/itemsController");
const itemsRouter = express.Router();

itemsRouter.post("/todos", itemsController.postItem);

module.exports = itemsRouter;

exports.postItem = async (req, res, next) => {
  try {
    console.log(req.body);
    const todoItem = new TodoItem(req.body);
    const item = await todoItem.save();
    res.json(item);
  } catch (err) {
    next(err);
  }
};
```



# 21.6 First API Todo App

8,9.

```
const bodyParser = require("body-parser");
const cors = require("cors");

app.use(bodyParser.urlencoded({ extended: true }));
app.use(cors());
app.use(express.json());
app.use(itemsRouter);
app.use(errorController.get404);
```



# 21.7 API for Fetch Items

```
itemsRouter.get("/todos", itemsController.getItems);
itemsRouter.post("/todos", itemsController.postItem);
```

```
exports.getItems = async (req, res, next) => {
  try {
    const items = await TodoItem.find();
    res.json(items);
  } catch (err) {
    next(err);
  }
};
```

```
export const todoItemToClientModel = (serverItem) => {
  return {
    id: serverItem._id,
    todoText: serverItem.task,
    todoDate: serverItem.date
  }
}
```

```
const formattedDate = new Date(todoDate).toLocaleDateString('en-US', {
  year: 'numeric',
  month: 'long',
  day: 'numeric'
});
```



# 21.8 API for Deleting Items

```
itemsRouter.get("/todos", itemsController.getItems);
itemsRouter.post("/todos", itemsController.postItem);
itemsRouter.delete("/todos/:id", itemsController.deleteItem);
```

```
exports.deleteItem = async (req, res, next) => {
  try {
    const { id } = req.params;
    await TodoItem.findByIdAndDelete(id);
    res.json({ id, message: "Item deleted" });
  } catch (err) {
    next(err);
  }
};
```



## 21.9 Improving UI Elements

1. Remove bootstrap
2. Add tailwind to the project.
3. Create the whole app using tailwind classes.

{ complete }  
CODING



# 21.10 Adding Complete Item functionality

1. Add a UI element to mark the item complete.
2. Add a component state based on which items will be striked through, default value should come from server.
3. Add a toggleComplete handler in TodolItem that sends a PATCH request to update the completed flag and expects the updated item in return. Also the value of state must be updated based on the final value.
4. Add a route and API in the backend to update the todolitem.



# 21.10 Adding Complete Item functionality

1,2.

```
<input
  type="checkbox"
  checked={isComplete}
  onChange={toggleComplete}
  className="w-5 h-5 rounded border-gray-300 text-blue-600 focus:ring-blue-500"
/>
```

```
const TodoItem = ({ id, todoText, todoDate, completed }) => {
  const { deleteTodoItem } = useContext(TodoItemsContext);
  const [isComplete, setIsComplete] = useState(completed);
```

```
<div className={`flex flex-col ${isComplete ? 'text-gray-400 line-through' : 'text-gray-700'}`}>
  <span className="font-medium">{todoText}</span>
  <span className="text-sm text-gray-500">{formattedDate}</span>
</div>
```



# 21.10 Adding Complete Item functionality

```
3. const toggleComplete = () => {
    fetch(`http://localhost:3000/todos/${id}`, {
        method: 'PATCH',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ completed: !isComplete })
    })
    .then(res => res.json())
    .then(data => {
        setIsComplete(data.completed);
    })
    .catch(err => console.log(err));
}
```



# 21.10 Adding Complete Item functionality

4.

```
itemsRouter.get("/todos", itemsController.getItems);
itemsRouter.post("/todos", itemsController.postItem);
itemsRouter.delete("/todos/:id", itemsController.deleteItem);
itemsRouter.patch("/todos/:id", itemsController.updateItem);

exports.updateItem = async (req, res, next) => {
  try {
    const { id } = req.params;
    const { completed } = req.body;
    const updatedItem = await TodoItem.findByIdAndUpdate(
      id,
      { completed: completed },
      { new: true }
    );

    res.json(updatedItem);
  } catch (err) {
    next(err);
  }
};
```



# Revision

1. What are Async Requests
2. What are REST APIs
3. Decoupling Frontend & Backend
4. Routes & HTTP Methods
5. REST Core Concepts
6. First API Todo App
7. API for Fetch Items
8. API for Deleting Items
9. Improving UI Elements
10. Adding Complete Item functionality

