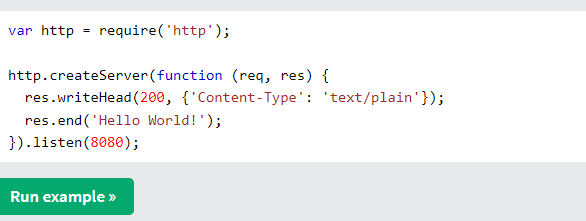




NODE JS

### Unit 4: Development using Node.js

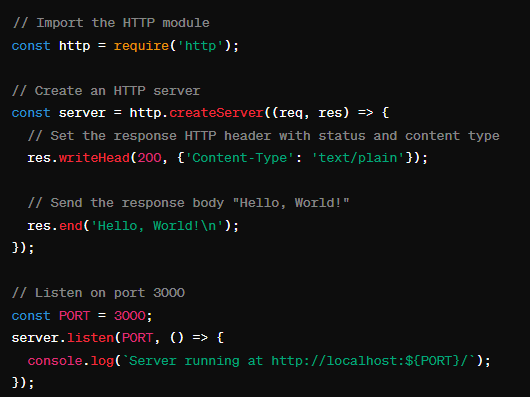
1. **Advantages of Node.js:**
   * Discusses the benefits of using Node.js, such as its non-blocking, event-driven architecture that makes it efficient for handling I/O-heavy operations.
2. **Traditional Web Server Model:**
   * Contrasts Node.js with traditional web server models (like Apache or Nginx) that use a synchronous request/response model.
3. **Node.js Process Model:**
   * Explains the Node.js process model, which includes a single-threaded event loop that handles asynchronous operations efficiently.
4. **Environment Setup:**
   * Guides on setting up the development environment for Node.js, including installing Node.js itself and any necessary dependencies.
5. **Node.js Console:**
   * Introduces the Node.js console, which allows developers to interact with Node.js applications via a command-line interface.
6. **Node.js Modules and Types:**
   * Discusses Node.js modules, which are reusable blocks of code, and the various types of modules, such as core modules, local modules, and third-party modules.
7. **Functions in Node.js:**
   * Covers the creation and use of functions in Node.js, including how to define functions and use them within Node.js applications.
8. **Buffer in Node.js:**
   * Explains the Buffer class in Node.js, which is used to handle binary data, such as reading from files or handling network data streams.
9. **Event-Driven Framework:**
   * Explores Node.js's event-driven architecture, where actions are triggered by events and handled by event listeners.
10. **Event Emitter Class:**
    * Introduces the **EventEmitter** class in Node.js, which is a core module used to handle events.
11. **Events and Event Loop:**
    * Discusses events and the event loop in Node.js, which is the mechanism that allows Node.js to perform non-blocking I/O operations.
12. **Inheriting Events:**
    * Covers how to inherit events in Node.js, allowing for the creation of custom event emitters and listeners.
13. **Node Package Manager (npm):**
    * Introduces npm, the package manager for Node.js, which is used to install and manage Node.js packages and dependencies.



### 1. Advantages of Node.js:

Node.js has gained popularity due to several key advantages it offers:

1. **Non-blocking, Asynchronous Nature:**
   * Node.js uses a non-blocking, event-driven architecture. This means that instead of waiting for an operation to complete (like reading a file or querying a database), Node.js continues to execute the rest of the code. When the operation is finished, it triggers a callback function. This approach is highly efficient for handling I/O-bound tasks.
2. **Single Programming Language:**
   * Node.js allows you to write server-side applications using JavaScript. This means that developers who are already familiar with JavaScript for front-end development can use the same language for the back end. This leads to better code consistency and developer productivity.
3. **Fast Execution:**
   * Node.js is built on the V8 JavaScript engine, which is the same engine that powers Google Chrome. V8 compiles JavaScript directly into machine code, making Node.js applications fast and efficient.
4. **Scalability:**
   * Node.js is designed to handle a large number of simultaneous connections. It uses an event-driven, non-blocking I/O model that makes it ideal for real-time applications where many clients are connected at once, such as chat applications, online gaming, or streaming services.
5. **Rich Ecosystem (npm):**
   * Node.js has a vast ecosystem of open-source libraries and modules available through npm (Node Package Manager). npm allows developers to easily find, install, and manage dependencies for their projects. This extensive library of packages saves developers time by providing pre-built solutions for common tasks.
6. **Community Support:**
   * Node.js has a large and active community of developers. This means that there are plenty of resources available online, such as tutorials, documentation, and forums. Developers can quickly find answers to their questions and stay updated on best practices.



### 2. Traditional Web Server Model:

To understand the traditional web server model, it's helpful to contrast it with the architecture of Node.js. Traditional web servers like Apache HTTP Server or Nginx follow a synchronous request/response model, which can be described as follows:

1. **Synchronous Request/Response:**
   * In a traditional web server model, each incoming request from a client is handled synchronously. This means that when a request arrives, the server dedicates a thread or process to handle that request until it's completed.
2. **Thread/Process Per Request:**
   * For each new request, the server creates a new thread or process. This approach allows the server to handle multiple requests concurrently. However, it can lead to resource inefficiencies, especially when dealing with a large number of concurrent connections.
3. **Blocking I/O Operations:**
   * When the server needs to perform I/O operations (like reading from a file, querying a database, or making an API request), it blocks the current thread/process until the operation completes. During this blocking time, the thread is idle and cannot handle other incoming requests.
4. **Resource Consumption:**
   * Because each request requires a dedicated thread/process, traditional servers can consume a significant amount of system resources (CPU and memory) when handling many concurrent connections. This limits the scalability of the server.
5. **Complexity in Code:**
   * Writing applications for traditional web servers often involves managing threads or processes explicitly. Developers need to handle concurrency, locking, and synchronization, which can lead to complex code.

### 3. Node.js Process Model:

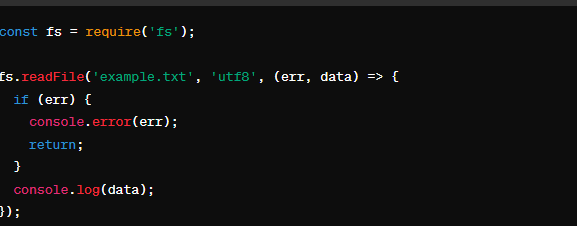
The Node.js process model is fundamental to understanding how Node.js operates. Unlike traditional server-side environments, Node.js uses a single-threaded, event-driven architecture to handle multiple concurrent operations efficiently. Here's an overview of the Node.js process model:

1. **Single-Threaded:**
   * Node.js operates using a single-threaded event loop. This means that it runs all JavaScript code in a single thread, allowing it to handle asynchronous operations without blocking.
2. **Event-Driven:**
   * Node.js is event-driven, meaning it uses events and callbacks to handle I/O operations and other asynchronous tasks. When an asynchronous operation completes or an event occurs, Node.js triggers the corresponding callback function.
3. **Event Loop:**
   * The event loop is at the core of the Node.js runtime. It continuously checks the event queue for new events and executes the associated callback functions. This loop allows Node.js to perform non-blocking I/O operations efficiently.
4. **Concurrency with Asynchronous Operations:**
   * While Node.js is single-threaded, it achieves concurrency through asynchronous operations. When Node.js encounters an asynchronous task (such as reading a file or making a network request), it initiates the operation and moves on to execute other code without waiting for the operation to complete. When the operation finishes, its callback function is added to the event queue for execution.
5. **Blocking vs. Non-Blocking:**
   * Node.js emphasizes non-blocking, asynchronous operations. Blocking operations, where the thread waits for an operation to complete before moving on, can lead to inefficiencies in server performance. Node.js's event loop ensures that the server remains responsive to new requests even while handling I/O operations.
6. **Handling CPU-Intensive Tasks:**
   * Node.js is well-suited for I/O-bound applications (like web servers or APIs) where the main bottleneck is waiting for I/O operations to complete. However, for CPU-intensive tasks that require heavy computations, Node.js may not be the best choice as it uses a single thread for executing JavaScript code.

### 6. Node.js Modules and Types:

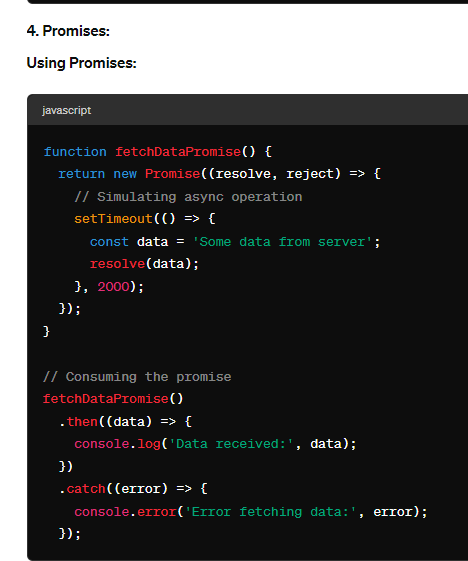
In Node.js, modules are reusable blocks of code that encapsulate related functionality. Understanding modules is essential for building organized and maintainable Node.js applications. Here's an overview of Node.js modules and their types:

#### Core Modules:



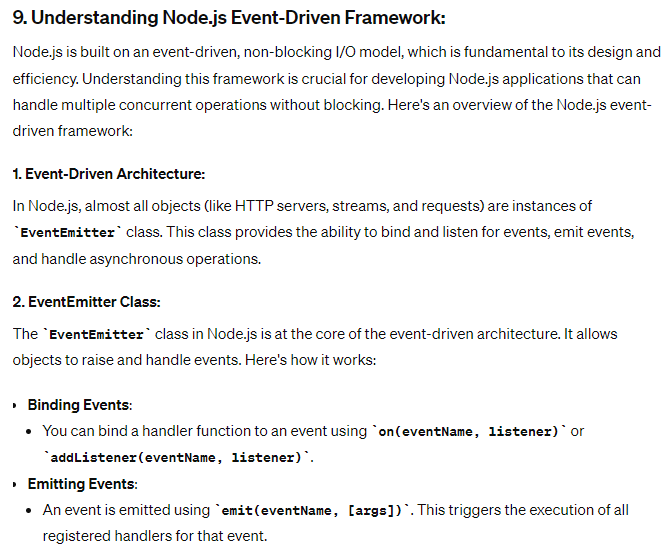


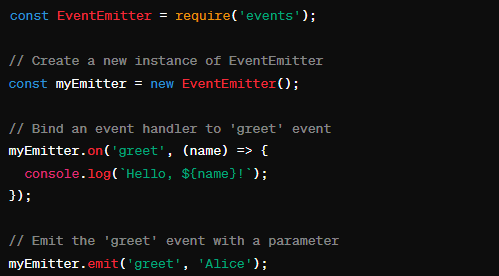




### 8. Buffer in Node.js:

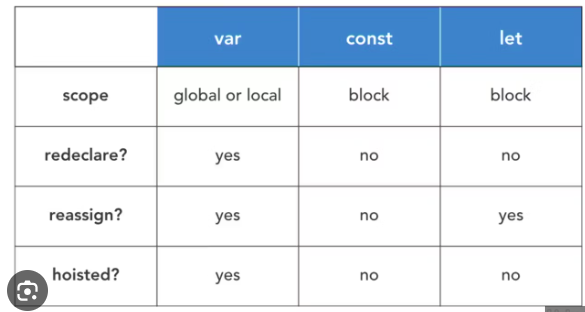
In Node.js, a **Buffer** is a global object that provides a way to work with binary data. It represents a fixed-size chunk of memory allocated outside of the JavaScript engine's garbage-collected heap. Buffers are useful for working with binary streams of data, such as reading from files, interacting with network protocols, or handling raw data.

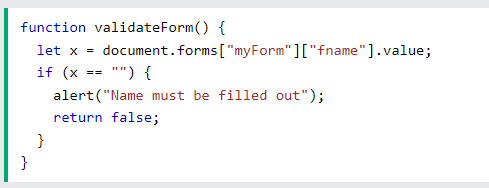




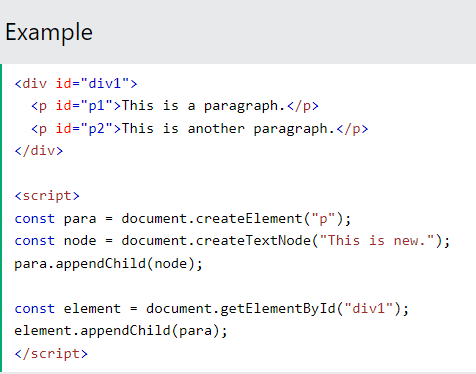
### 10. Node Package Manager (npm):

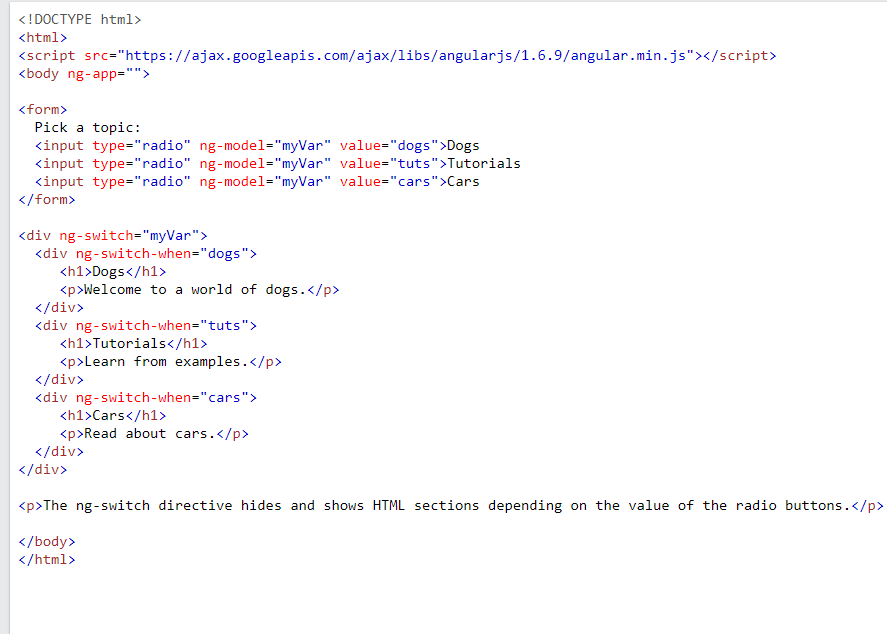
Node Package Manager (npm) is a powerful tool that comes bundled with Node.js. It allows developers to easily manage packages (or libraries) for Node.js applications. npm is used for installing, updating, and removing packages, as well as managing project dependencies. Here's an overview of npm and its common commands:





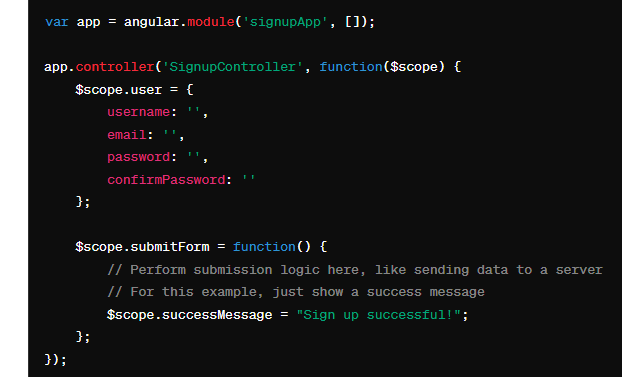


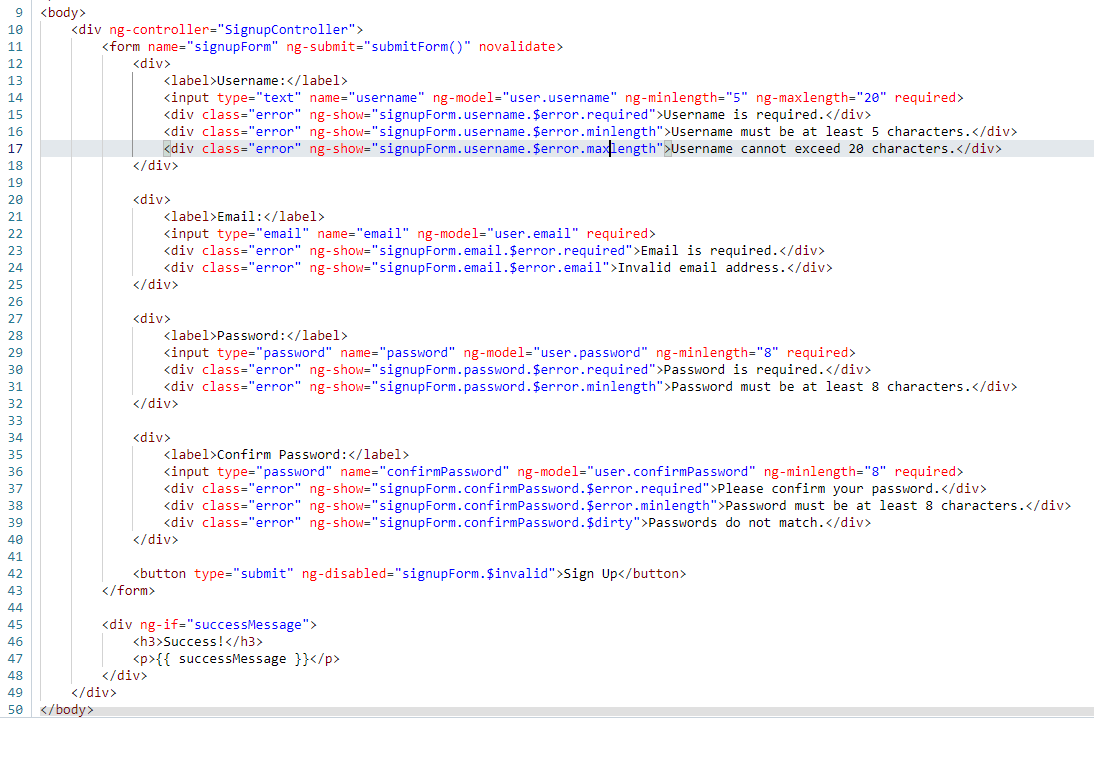




Angular js singup form

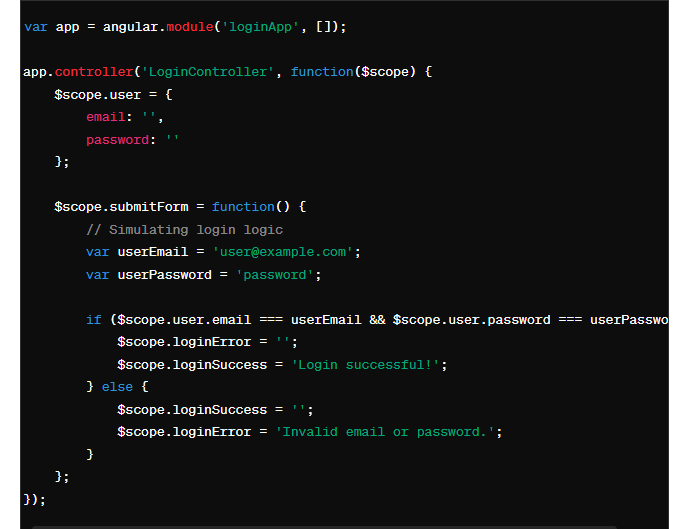






Login form validation





Custom directive in Angular JS



