UNIT I

Introduction to FullStack Development: Understanding the Basic Web Development Framework- User, Browser, Webserver, Backend Services, Full Stack Components - Node.js, MongoDB, Express, React, Angular. Java Script Fundamentals, NodeJS- Understanding Node.js, Installing Node.js, Working with Node Packages, creating a Node.js Application, Understanding the Node.js Event Model, Adding Work to the Event Queue, Implementing Callbacks

Full stack development:

Full Stack Development refers to the development of both front end (client side) and Backend (server side) portions of web applications.

Full Stack Web Developers:

Full Stack web developers have the ability to design complete web applications and websites. They work on the frontend, backend, database, and debugging of web applications or websites.

Technology Related to Full Stack Development:

Front-end Development

It is the visible part of website or web application which is responsible for user experience. The user directly interacts with the front end portion of the web application or website.

Front-end Technologies

The front end portion is built by using some languages which are discussed below:

HTML:

HTML stands for Hyper Text Markup Language. It is used to design the front endportion of web pages using markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. The markup languageis used to define the text documentation within tag which defines the structure of web pages.

CSS:

Cascading Style Sheets, fondly referred to as CSS, is a simply designed languageintended to simplify the process of making web pages presentable. CSS allows you to applystyles to web pages. More importantly, CSS enables you to do this independent of the HTMLthat makes up each web page.

JavaScript:

JavaScript is a famous scripting language used to create the magic on the sitesto make the site interactive for the user. It is used to enhancing the functionality of a websiteto running cool games and web-based software.

Front End Libraries and Frameworks

AngularJS:

AngularJs is a JavaScript open source front-end framework that is mainly used to develop single page web applications(SPAs). It is a continuously growing and expanding framework which provides better ways for developing web applications. It changes the static HTML to dynamic HTML. It is an open source project which can be freely used and changed by anyone. It extends HTML attributes with Directives, and data is bound with HTML.

React.js:

React is a declarative, efficient, and flexible JavaScript library for building userinterfaces. ReactJS is an open-source, component-based front end library responsible onlyfor the view layer of the application. It is maintained by Facebook.

Bootstrap:

Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScriptframe work for developing responsive, mobile-first web site

Understanding the Basic Web Development Framework

The main components of any given web framework are the user, browser, Webserver, and backend services. Although websites vary greatly on appearance and behavior, all have these basic components in one form or another.

User expectations define the requirements for developing a good website, and these ex

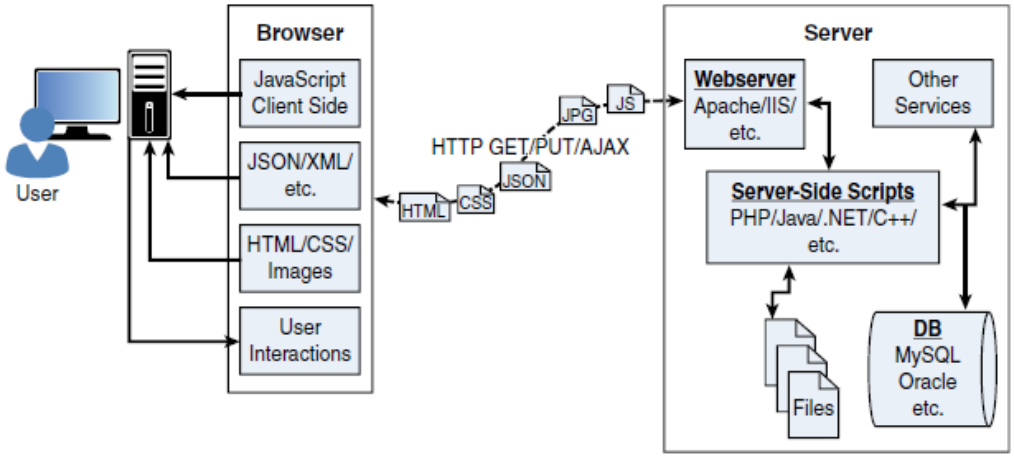


Figure 1.1 Diagram showing the components of a basic website/web application

User

Users are a fundamental part of all websites; they are, after all, the reason websites exist in the first place. pectations have changed a lot over the years. Users used to accept the slow, cumbersome experience of the "world-wide-wait," but no longer. They expect websites to behave closer to applications installed on their computers and mobile devices.

Browser:

The browser plays three roles in the web framework.

- First, it provides communication to and from the webserver.
- Second, it interprets the data from the server and renders it into the view that the user actually sees.
- Finally, the browser handles user interaction through the keyboard, mouse, touchscreen, or other input device and takes the appropriate action.

Browser-to-Webserver Communication

Browser-to-webserver communication consists of a series of requests, using the HTTP and HTTPS protocols. Hypertext Transfer Protocol (HTTP) is used to define communication between the browser and the webserver. HTTP defines what types of requests can be made as well as the format of those requests and the HTTP response.

HTTPS adds an additional security layer, SSL/TLS, to ensure secure connections by requiring the webserver to provide a certificate to the browser. The user can then determine whether to accept the certificate before allowing the connection.
There are three main types of requests that a browser will make to a webserver:

GET:
The GET request is typically used to retrieve data from the server, such as .html files, images, or JSON data.
POST:
POST requests are used when sending data to the server, such as adding an item to a shopping cart or submitting a web form.
AJAX:
Asynchronous JavaScript and XML (AJAX) is actually just a GET or POSTrequest that is done directly by JavaScript running in the browser. Despite the name, anAJAX request can receive XML, JSON, or raw data in the response.

Rendering the Browser View

The screen that the user actually views and interacts with is often made up of several different pieces of data retrieved from the webserver. The browser reads data from the initial URL and then renders the HTML document to build a Document Object Model (DOM). The DOM is a tree structure object with the HTML document as the root. The structure of the tree basically matches the structure of the HTML document.

For example, document will have html as a child,and html will have head and body as c hildren, and body may have div, p, or other elements aschildren, like this:

document

+ html
+ head
+ body
+ div
+ p

The browser interprets each DOM element and renders it to the user's screen to build the webpage view.
The browser often ends up getting various types of data from multiple webserver requests to build the webpage. The following are the most common types of data the browser uses to render the final user view as well as define the webpage behavior.

■ HTML files: These provide the fundamental structure of the DOM.

■ CSS files: These define how each of the elements on the page is to be styled; for example, font, color, borders, and spacing.

■ Client-side scripts: These are typically JavaScript files. They can provide added functionality to the webpage, manipulate the DOM to change the look of the webpage, and provide any necessary logic required to display the page and provide functionality.

■ Media files: Image, video, and sound files are rendered as part of the webpage.

■ Data: Any data, such as XML, JSON, or raw text, can be provided by the webserver as a response to an AJAX request. Rather than sending a request back to the server to rebuild the webpage, new data can be retrieved via AJAX and inserted into the webpage via JavaScript.

Full Stack Components:

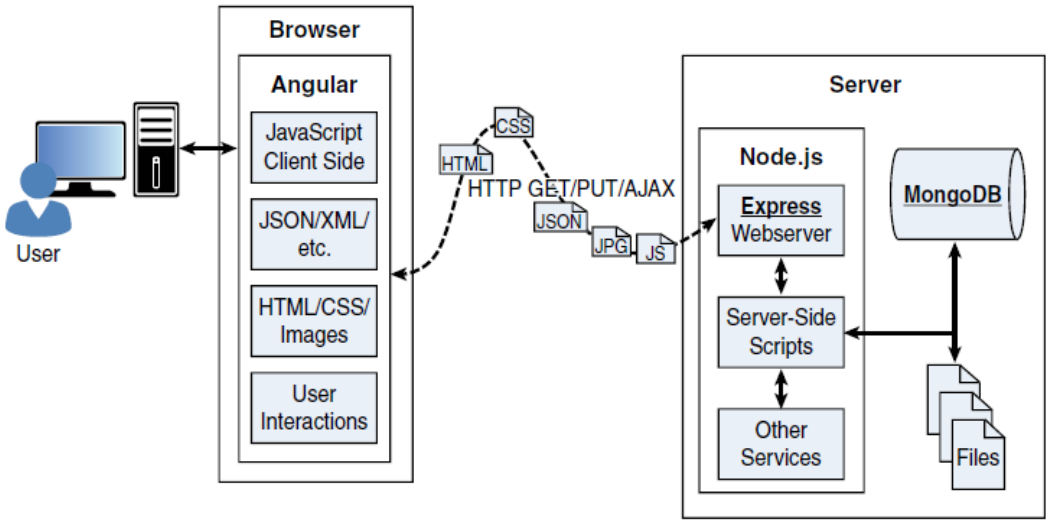


Figure 1.2 Basic diagram showing where Node.js, Express, MongoDB, and Angular fit in the web paradigm

---

I. Node.js

- Node.js is a development framework based on Google's V8 JavaScript engine. Therefore, Node.js code is written in JavaScript and then compiled into machine code by V8 to be executed.
- Many of your backend services can be written in Node.js, as can the server-side scripts and any supporting web application functionality.

The following are just a few reasons why Node.js is a great framework to start from:

JavaScript end-to-end:
One of the biggest advantages of Node.js is that it allows you to write both server- and client-side scripts in JavaScript. There have always been difficulties in deciding whether to put logic in client-side scripts or server-side scripts. With Node.js you can take JavaScript written on the client and easily adapt it for the server and vice versa. An added plus is that client developers and server developers are speaking the same language.

Event-driven scalability:
Node.js applies a unique logic to handling web requests. Rather than having multiple threads waiting to process web requests, with Node.js they are processed on the same thread, using a basic event model. This allows Node.js webservers to scale in ways that traditional webservers can't.

Extensibility:
Node.js has a great following and very active development community. People are providing new modules to extend Node.js functionality all the time. Also, it is very simple to install and include new modules in Node.js; you can extend a Node.js project to include new functionality in minutes.

Fast implementation:
Setting up Node.js and developing in it are super easy. In only a few minutes you can install Node.js and have a working webserver.

II. MongoDB

MongoDB is an agile and very scalable NoSQL database. It is based on the NoSQL document store model, which means data is stored in the database as basically JSON objects rather than as the traditional columns and rows of a relational database.

MongoDB provides great website backend storage for high-traffic websites that need to store data such as user comments, blogs, or other items because it is quickly scalable and easy to implement. This book covers using the MongoDB driver library to access MongoDB from Node.js.

Node.js supports a variety of database access drivers, so the data store can easily be MySQL or some other database. However, the following are some of the reasons that MongoDB really fits in the Node.js stack well:

Document orientation:
Because MongoDB is document oriented, data is stored in the database in a format that is very close to what you deal with in both server-side and client-side scripts. This eliminates the need to transfer data from rows to objects and back.

High performance:
MongoDB is one of the highest-performing databases available. Especially today, with more and more people interacting with websites, it is important to have a backend that can support heavy traffic.

High availability:
MongoDB's replication model makes it very easy to maintainscalability while keeping high performance.

High scalability:
MongoDB's structure makes it easy to scale horizontally by sharding the data across multiple servers.

III. Express

The Express module acts as the webserver in the Node.js-to-AngularJS stack. Because it runs inNode.js, it is easy to configure, implement, and control. The Express module extends Node.js to provide several key components for handling web requests. It allows you to implement a running webserver in Node.js with only a few lines of code.

For example, the Express module provides the ability to easily set up destination routes (URLs) for users to connect to. It also provides great functionality in terms of working with HTTP request and response objects, including things like cookies and HTTP headers.The following is a partial list of the valuable features of Express:

Route management:
Express makes it easy to define routes (URL endpoints) that tie directly to the Node.js script functionality on the server.

Error handling:
Express provides built-in error handling for "document not found" and other errors.

Easy integration:
An Express server can easily be implemented behind an existing reverse proxy system, such as Nginx or Varnish. This allows you to easily integrate it into your existing secured system.

Cookies:
Express provides easy cookie management.

Session and cache management:
Express also enables session management and cache management.

IV. AngularJS

AngularJS is a client-side framework developed by Google. It provides all the functionality needed to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. It is written in JavaScript, with a reduced jQuery library. The theory behind Angular JS is to provide a framework that makes it easy to implement web applications using the MVC framework.

Other JavaScript frameworks could be used with the Node.js platform, such as Backbone, Ember, and Meteor. However, AngularJS has the best design, feature set, and trajectory at this writing. Here are some of the benefits AngularJS provides:

Data binding:
AngularJS has a very clean method for binding data to HTML elements, using its powerful scope mechanism.

Extensibility:
The AngularJS architecture allows you to easily extend almost every aspect of the language to provide your own custom implementations.

Clean:
AngularJS forces you to write clean, logical code.

Reusable code:
The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. In fact, the language often forces you to do so when creating custom services.

Support:
Google is investing a lot into this project, which gives it an advantage over similar initiatives that have failed.

Compatibility:
AngularJS is based on JavaScript and has a close relationship with jQuery. This makes it easier to begin integrating AngularJS into your environment andr euse pieces of your existing code within the structure of the AngularJS framework.

JavaScript Fundamentals:

Syntax and Basics

Variables - Used to store data values.

```
var name = "John";
let age = 30;
const isStudent = true;
```

Data Types - Common data types include strings, numbers, booleans, null, undefined,objects, and arrays

```
let message = "Hello, World!"; // String
let count = 42; // Number
let isActive = true; // Boolean
let student = null; // Null
let course; // Undefined
```

Creating Functions:

1. **Function Declaration**

It starts with the function keyword, followed by the function name and any parameters the function needs.

```
// Function declaration
function add(a, b) {
   console.log(a + b);
}

// Calling a function
add(2, 3);
```

2. **Function Expression:**
   Function Expression is another way to define a function. Here, the function is defined inside a variable, and the function's value (its returned value) is stored in that variable.

```
// Function Expression
const add = function (a, b) {
   console.log(a + b);
}

// Calling function
add(2, 3);
```

3. **Arrow Functions**

   They provide a shorter syntax for writing functions. Instead of using the function keyword, you use an arrow (=>).

```
let add = (a, b) => a + b;
console.log(add(3, 2));
```

4. **Callbacks**
   A **callback function** is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

```
//function
function greet(name, callback) {
        console.log('Hi'+' '+name);
        callback();
```

```
}

//callback function
function callMe() {
console.log('I am callback function');
}

//passing function as an argument
greet('Peter',callMe);
```

Objects and Arrays

Objects : Collections of key-value pairs.

```
Let person = {
    firstName: "John",
    lastName: "Doe",
    age: 25
    };
        console.log(person.firstName); // John
```

Arrays:

Create Array using Literal:

Creating an array using array literal involves using square brackets [] to define and initialize the array

```
// Creating an Empty Array
let a = [];
console.log(a);

// Creating an Array and Initializing with Values
let b = [10, 20, 30];
console.log(b);
```

output:

```
[]
[ 10, 20, 30 ]
```

2. Create using new Keyword (Constructor)

The "**Array Constructor**" refers to a method of creating arrays by invoking the Array constructor function.

```
// Creating and Initializing an array with values
let a = new Array(10, 20, 30);

console.log(a);
```

Output

```
[ 10, 20, 30 ]
```

Understanding Node.js:

Node.js is a runtime environment that allows you to run JavaScript code on the server side. It uses the V8 JavaScript engine (the same one used by Google Chrome) to execute code, and it provides anon-blocking, event-driven architecture for building scalable network applications.

Installing Node.js

1. **Download Node.js**

   Visit theofficial Node.js website.

   Download the installer for your operating system (Windows, macOS, Linux).

2. **Install Node.js**

   Run the installer and follow the instructions.

   Verify the installation by opening a terminal or command prompt and typing:

   node –v

   npm -v

   This should display the installed versions of Node.js and npm (Node Package Manager).

Working with Node Packages:

One of the most powerful features of the Node.js framework is the ability to easily extend it with additional Node Packaged Modules (NPMs) using the Node Package Manager (NPM).

1. **Initialize a Project**

- Create a new directory for your project and navigate into it.

```
mkdir my-node-app
cd my-node-app
```

- Initialize a new Node.js project:

```
npm init -y
```

This creates a package.json file in your project directory, which keeps track of your project's dependencies and configuration.

2. **Installing Packages**

Install a package (e.g., Express for web server functionality):

```
npm install expresso
```

Installed packages are listed in the package.json file under "dependencies".

3. **Using Packages**

Require the installed package in your JavaScript files:

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

Using the Node Package Manager

Option	Description	Example
search	Finds module packages in the repository	npm search express
install	Installs a package either using a package.json file, from the repository, or a local location	npm install npm install express npm install express@0.1.1 npm install ../tModule.tgz
install –g	Installs a package globally	npm install express –g

remove	Removes a module	npm remove express
pack	Packages the module defined by the package.json file into a .tgz file	npm pack
view	Displays module details	npm view express
publish	Publishes the module defined by a package.json file to the registry	npm publish
unpublish	Unpublishes a module you have published	npm unpublish myModule
owner	Allows you to add, remove, and list owners of a package in the repository	npm add bdayley myModule npm rm bdayley myModule npm ls myModule

Using package.json:

All Node modules must include a package.json file in their root directory. The package.json file is a simple JSON text file that defines the module including dependencies. The package.json file can contain a number of different directives to tell the Node Package Manager how to handle the module.
The following is an example of a package.json file with a name, version, description, and
dependencies:

```
{
"name": "my_module",
"version": "0.1.0",
"description": "a simple node.js module",
"dependencies" : {
"express" : "latest"
}
}
```

The only required directives in the package.json file are name and version. The rest depend on what you want to include.

Creating a Node.js Packaged Module

1. Create a project folder named .../censorify. This is the root of the package.
2. Inside that folder, create a file named censortext.js.
3. Add the code.
4. Run the Application

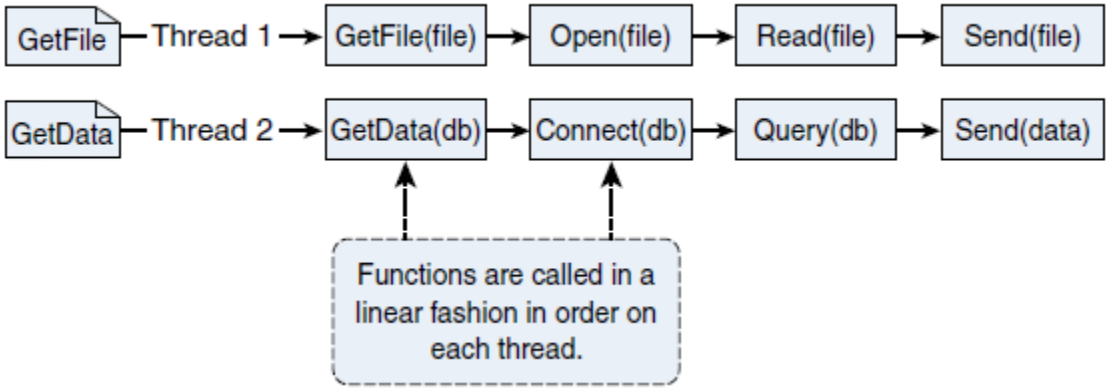
```
const http = require('http');
http. createServer(function(req,response)
{
   response.writeHead(200,{‘Content-Type’: ‘text/plain’});
   response.write("hello this is sreyas");
   response.end();
}).listen(4000);
```

Understanding the Node.js Event Model:

Node.js applications are run in a single-threaded event-driven model. Although Node.js implements a thread pool in the background to do work, the application itself doesn’t have any concept of multiple threads.

Comparing Event Callbacks and Threaded Models:
In the traditional threaded web model, a request comes in to the webserver and is assigned to an available thread. Then the handling of work for that request continues on that thread until the request is complete and a response is sent.

Example:
Threaded model processing two requests, GetFile and GetData. The GetFile request first opens the file, reads the contents, and then sends the data back in a response. All this occurs in order on the same thread. The GetData request connects to the DB, queries the necessary data, and then sends the data in the response.



The Node.js event model does things differently. Instead of executing all the work for each request on individual threads, work is added to an event queue and then picked up by a single thread running an event loop. The event loop grabs the top item in the event queue, executes it, and then grabs the next item. When executing code that is no longer live or has blocking I/O, instead of calling the function directly, the function is added to the event queue along with a callback that is executed after the function completes. When all events on the Node.js event queue have been executed, the Node application terminates.

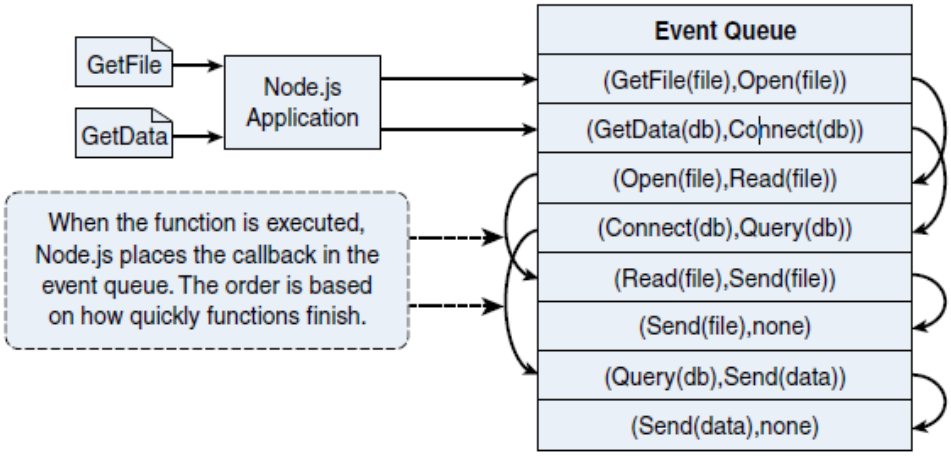


Figure 4.2 Processing two requests on a single event-driven thread using the Node.js event model

Blocking I/O in Node.js

The Node.js event model of using the event callbacks is great until you run into the problem of functions that block waiting for I/O. Blocking I/O stops the execution of the

current thread and waits for a response before continuing. Some examples of blocking I/O are

- Reading a file
- Querying a database
- Socket request
- Accessing a remote service

The reason Node.js uses event callbacks is not to have to wait for blocking I/O. Therefore, any requests that perform blocking I/O are performed on a different thread in the background. Node.js implements a thread pool in the background. When an event that requires blocking I/O is retrieved from the event queue, Node.js retrieves a thread from the thread pool and executes the function there instead of on the main event loop thread. This prevents the blocking I/O from holding up the rest of the events in the event queue.

The function executed on the blocking thread can still add events back to the event queue to be processed.

Example:

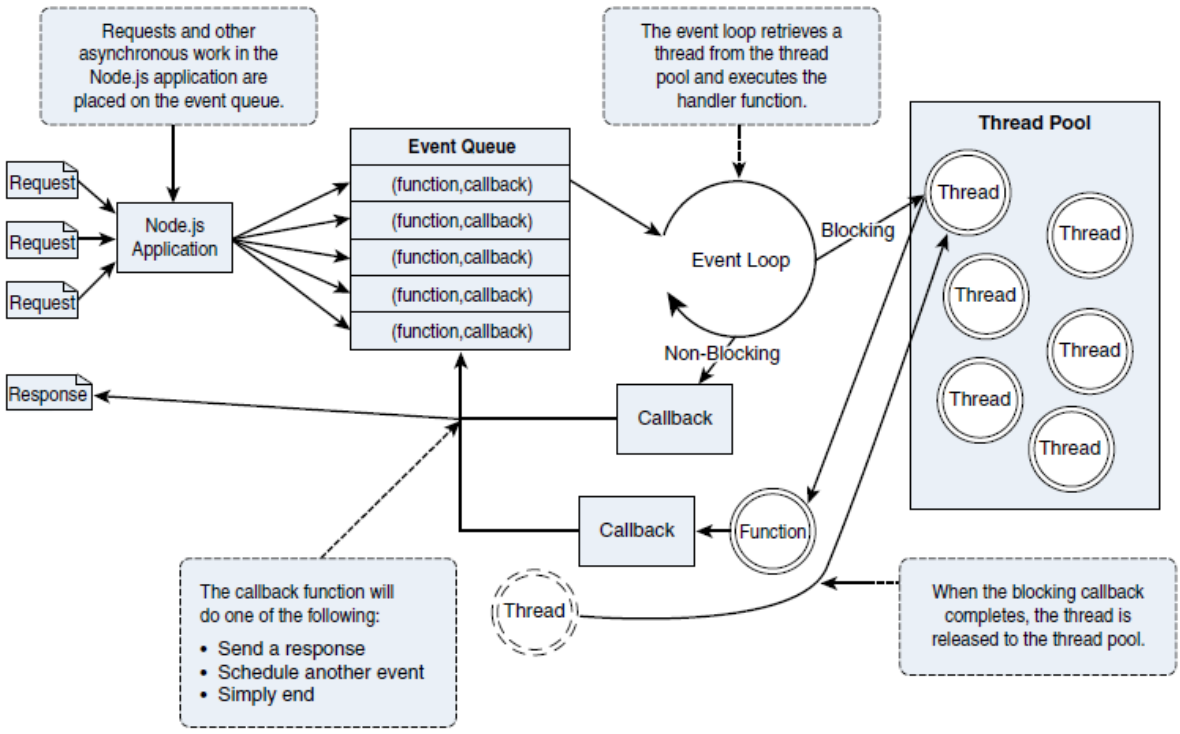


Figure 4.3 In the Node.js event model, work is added as a function with callback to the event queue, and then picked up on the event loop thread. The function is then executed on the event loop thread in the case of non-blocking, or on a separate thread in the case of blocking

Adding Work to the Event Queue :

- Make a call to one of the blocking I/O library calls such as writing to a file or connecting to a database.
- Add a built-in event listener to a built-in event such as an http.request or server. connection.
- Create your own event emitters and add custom listeners to them.
- Use the process.nextTick option to schedule work to be picked up on the next cycle of the event loop.
- Use timers to schedule work to be done after a particular amount of time or at periodic intervals.

Implementing Timers

You can use functions like setTimeout, setInterval, and process.nextTick to add tasks to the event queue.

1. **setTimeout**

syntax:

setTimeout(myFunc, 1000);

example:

```
function simpleTimeout(consoleTimer){
 console.timeEnd(consoleTimer);
}
 console.time("twoSecond");
 setTimeout(simpleTimeout, 2000, "twoSecond");
 console.time("oneSecond");
 setTimeout(simpleTimeout, 1000, "oneSecond");
```

output:

```
C:\books\node\ch04> node simple_timer.js
50MilliSecond: 50.489ms
oneSecond: 1000.688ms
twoSecond: 2000.665ms
```

2. **setInterval**

Interval timers are created using the setInterval(callback, delayMilliSeconds, [args]) method built into Node.js. When you call setInterval(), the callback function

is executed every interval after delayMilliSeconds has expired. For example, the following executes myFunc() every second:

syntax:

setInterval(code)
setInterval(code, delay)

setInterval(func)
setInterval(func, delay)
setInterval(func, delay, arg1)

```
const intervalID = setInterval(myCallback, 1500, "Parameter 1", "Parameter 2");

function myCallback(a, b) {
  // Your code here
  // Parameters are purely optional.
  console.log(a);
  console.log(b);
}
```

3. **setImmediate:**

setImmediate(func)
setImmediate(func, param1)
setImmediate(func, param1, param2)

```
console.log("Started");
setImmediate(() => {
  console.log("Now  Executing setImmediate Block...");
});
  console.log("Task1");
  console.log("Task2");
```

Using nextTick to Schedule Work:

A useful method of scheduling work on the event queue is the process.nextTick (callback) function. This function schedules work to be run on the next cycle of the event loop. Unlike the setImmediate() method.

```
var fs = require("fs");
fs.stat("nexttick.js", function(){
 console.log("nexttick.js Exists");
 });
 setImmediate(function(){
 console.log("Immediate Timer 1 Executed");
 });

setImmediate(function(){
 console.log("Immediate Timer 2 Executed");
 });

 process.nextTick(function(){
 console.log("Next Tick 1 Executed");
 });

 process.nextTick(function(){
 console.log("Next Tick 2 Executed");
 });
```

Output:

```
Next Tick 1 Executed
Next Tick 2 Executed
Immediate Timer 1 Executed
Immediate Timer 2 Executed
nexttick.js Exists
```

Implementing Callbacks

Callbacks are functions passed as arguments to other functions and are invoked after the completion of certain tasks

Passing Additional Parameters to Callbacks

Most callbacks have automatic parameters passed to them, such as an error or result buffer.

```
function greet(name, callback) {
        console.log(`Hello, ${name}!`);
        callback();
        }
```

```
function afterGreet() {
console.log('This function runs after the greet function');
}

greet('John', afterGreet);
```

output:

```
Hello, John!
This function runs after the greet function
```

Example with Asynchronous Code:

```
const fs = require('fs');
fs.readFile('example.txt', 'utf8', (err, data) => {
if (err)
{
console.error(err);
 return;
}
console.log(data);
});
```

fs.readFile reads a file asynchronously, and the callback function is executed after the file reading is complete