CALLBACKS

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there's no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Code:

Non-Blocking Code Example

Create a text file named input.txt with the following content.

I will be printed last

Update main.js to have the following code −

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
        if (err) return console.error(err);
        console.log(data.toString());
});

console.log("Program Ended");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Program Ended
I will be printed last
```

SESSIONS:
A website is based on the HTTP protocol. HTTP is a stateless protocol which means at the end of every request and response cycle, the client and the server forget about each other.

This is where the session comes in. A session will contain some unique data about that client to allow the server to keep track of the user's state. In session-based authentication, the user's state is stored in the server's memory or a database

How sessions works:
When the client makes a login request to the server, the server will create a session and store it on the server-side. When the server responds to the client, it sends a cookie. This cookie will contain the session's unique id stored on the server, which will now be stored on the client. This cookie will be sent on every request to the server.

We use this session ID and look up the session saved in the database or the session store to maintain a one-to-one match between a session and a cookie. This will make HTTP protocol connections stateful.

In a cookie, you can't store large data but in sessions you can.

STREAMS
Streams are objects that let you read data from a source or write data to a destination in continuous fashion. In Node.js, there are four types of streams −

Readable − Stream which is used for read operation.
Writable − Stream which is used for write operation.
Duplex − Stream which can be used for both read and write operation.
Transform − A type of duplex stream where the output is computed based on input.

Each type of Stream throws several events at different instance of times.
For example, some of the commonly used events are −

data − This event is fired when there is data is available to read.
end − This event is fired when there is no more data to read.
error − This event is fired when there is any error receiving or writing data.
finish − This event is fired when all the data has been flushed to underlying system.

EVENT LOOP

Node.js is a single-threaded event-driven platform that is capable of running non-blocking, asynchronously programming. These functionalities of Node.js make it memory efficient. The event loop allows Node.js to perform non-blocking I/O operations despite the fact that JavaScript is single-threaded. It is done by assigning operations to the operating system whenever and wherever possible.

Most operating systems are multi-threaded & hence can handle multiple operations executing in the background. When one of these operations is completed, the kernel tells Node.js and the respective callback assigned to that operation is added to the event queue which will eventually be executed.

Features of Event Loop:

- Event loop is an endless loop, which waits for tasks, executes them and then sleeps until it receives more tasks.
- The event loop executes tasks from the event queue only when the call stack is empty i.e. there is no ongoing task.
- The event loop allows us to use callbacks and promises.
- The event loop executes the tasks starting from the oldest first.

Example:

```
console.log("This is the first statement");
setTimeout(function(){
        console.log("This is the second statement");
        }, 1000);
console.log("This is the third statement");
```

Output:

```
This is the first statement
This is the third statement
This is the second statement
```

STATE

- State is a plain JavaScript object used by React to represent an information about the component's current situation
- stores component's dynamic data and it enables a component to keep track of changes between renders.
- To change what is happening on the screen

- A state can be modified based on user action or network changes
- A component's state can change over time; whenever it changes, the component re-renders. The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.
- Every time the state of an object changes, React re-renders the component to the browser.

Code:

```
function App( ){
        let counter=0;

        const incrementer= ( ) => {
                counter+=1;
                console.log(counter);
        };

        return (
                <div className=”App”>
                   <h1>Hello React </h1>
                   <h2>Counter {counter} </h2>
                   <button onClick= {incrementer}> Click </button>
                </div>
        );
}
```

MVC

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects

Model: The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

View: The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller: Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

BUFFERS

A buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another. In node, each buffer corresponds to some raw memory allocated outside V8. A buffer acts like an array of integers, but cannot be resized. The Buffer class is global. It deals with binary data directly and can be constructed in a variety of ways.

S. No.	String	StringBuffer
1	String is immutable.	It is mutable.
2	It is slow in terms of executing the concatenation task.	It is fast in terms of executing the concatenation task.
3	Here the length of the string class is static.	Here the length can be modified whenever required, as it is dynamic in behaviour.
4	It is less efficient.	It is more efficient in nature as compared to the string class.
5	String consumes more as compared to the stringbuffer.	StringBuffer uses less memory as compared to the string.
5	It utilises a string constant pool to store the values.	It prefers heap memory to store the objects.
6	It overrides both equal() and hashcode() techniques of object class.	It cannot override equal() and hashcode() methods.

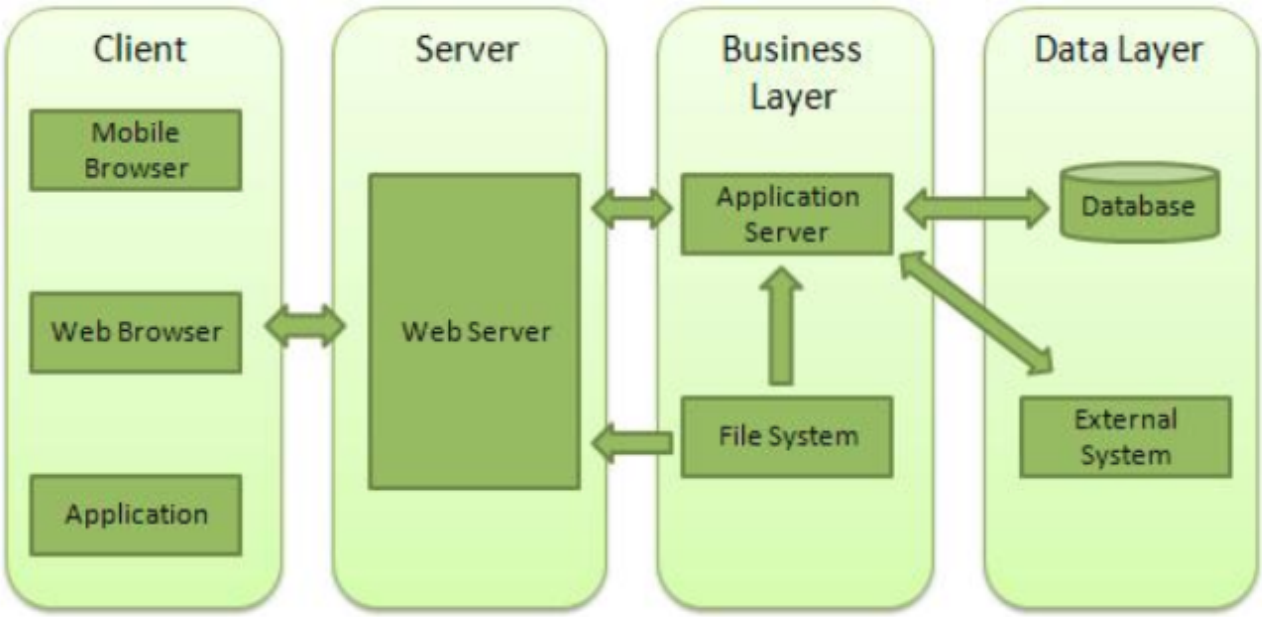
WEB MODULE

A **Web Server** is a software application which handles HTTP requests sent by the HTTP client, like web browsers, and returns web pages in response to the clients. Web servers usually deliver html documents along with images, style sheets, and scripts.

Most of the web servers support server-side scripts, using scripting languages or redirecting the task to an application server which retrieves data from a database and performs complex logic and then sends a result to the HTTP client through the Web server.

Apache web server is one of the most commonly used web servers. It is an open source project.

Web Application Architecture: Web application is usually divided into four layers −



- Client − This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.
- Server − This layer has the Web server which can intercept the requests made by the clients and pass them the response.
- Business − This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.
- Data − This layer contains the databases or any other source of data

DOM and VIRTUAL DOM

The DOM represents the web page often called a document with a logical tree and each branch of the tree ends in a node and each node contains object programmers can modify the content of the document using a scripting language like javascript and the changes and updates to the dom are fast because of its tree-like structure but after changes, the updated element and its children have to be re-rendered to update the application UI so the re-rendering of the UI which make the dom slow all the UI components you need to be rendered for every dom update so real dom would render the entire list and not only those item that receives the update

- **VDOM** is the virtual representation of Real DOM
- React update the state changes in Virtual DOM first and then it syncs with Real DOM
- Virtual DOM is just like a blueprint of a machine, can do the changes in the blueprint but those changes will not directly apply to the machine.
- Virtual DOM is a programming concept where a virtual representation of a UI is kept in memory synced with "Real DOM " by a library such as ReactDOM and this process is called reconciliation
- Virtual DOM makes the performance faster, not because the processing itself is done in less time. The reason is the amount of changed information – rather than wasting time on updating the entire page, you can dissect it into small elements and interactions

Real DOM	Virtual DOM
DOM manipulation is very expensive	DOM manipulation is very easy
There is too much memory wastage	No memory wastage
It updates Slow	It updates fast
It can directly update HTML	It can't update HTML directly
Creates a new DOM if the element updates.	Update the JSX if the element update
It allows us to directly target any specific node (HTML element)	It can produce about 200,000 Virtual DOM Nodes / Second.
It represents the Ul of your application	It is only a virtual representation of the DOM

FILE SYSTEMS

To handle file operations like creating, reading, deleting, etc., Node.js provides an inbuilt module called FS (File System). Node.js gives the functionality of file I/O by providing wrappers around the standard POSIX functions. All file system operations can have synchronous and asynchronous forms depending upon user requirements. To use this File System module, use the require() method: var fs = require('fs');

Common use for File System module:

- Read Files
- Write Files
- Append Files
- Close Files
- Delete Files

What is Synchronous and Asynchronous approach?

- Synchronous approach: They are called blocking functions as it waits for each operation to complete, only after that, it executes the next operation, hence blocking the next command from execution i.e. a command will not be executed until & unless the query has finished executing to get all the result from previous commands.
- Asynchronous approach: They are called non-blocking functions as it never waits for each operation to complete, rather it executes all operations in the first go itself. The result of each operation will be handled once the result is available i.e. each command will be executed soon after the execution of the previous command. While the previous command runs in the background and loads the result once it is finished processing the data.
- Use cases:
    - If your operations are not doing very heavy lifting like querying huge data from DB then go ahead with Synchronous way otherwise Asynchronous way.
    - In an Asynchronous way, you can show some progress indicator to the user while in the background you can continue with your heavyweight works. This is an ideal scenario for GUI based apps.