1. **What do you mean by Nodejs? What are its features? WAP to create an app in Node.js**

Node.js (Node) is an open source development platform for executing JavaScript code server-side. Node is useful for developing applications that require a persistent connection from the browser to the server and is often used for real-time applications such as chat, news feeds and web push notifications.
Features:-

- **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.

- **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

- **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.

- **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.

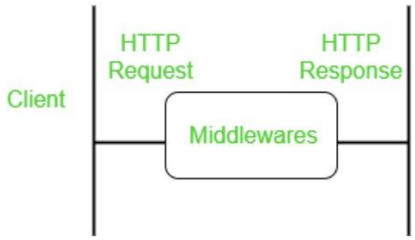
- **License** − Node.js is released under the MIT license

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
```

```
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

2. **What is Middleware in Express? State its Types with appropriate example**

Middleware is a framework for handling the different routing of the webpage and it works between the request and response cycle. Middleware gets executed after the server receives the request and before the controller actions send the response. Middleware has access to the request object, responses object, and next, it can process the request before the server sends a response. An Express-based application is a series of middleware function calls.



Types of express middleware

- **Application level middleware (app.use)**
  which runs for all routes in an app object. Application level middleware is applicable to the whole application

```
Eg. app.use(function(req, res, next){
        console.log("request ip: ",req.ip);
        next();
});
```

- **Router level middleware (router.use)**
  Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of express.Router().

```
Eg. app.get('/user",function(req,res,next){
        console.log('first middleware");
        next();
}, function(req,res,next){
        console.log("second middleware");
        next();
});
```

- **Built-in middleware**
  Express.static,express.json,express.urlencoded

- **Error handling middleware** (app.use(err,req,res,next))
  Express JS comes with default error handling params, define error-handling middleware functions in the same way as other middleware functions, except error-handling functions have four arguments instead of three:
  Eg. app.use(function (err, req, res, next) {
  console.error(err.stack)
  res.status(500).send('Something broke!')
  })

- **Third Party middleware** bodyparser,cookieparser

3. **Explain Forms and their types with example**

Forms are an integral part of any modern web application. It allows the users to interact with the application as well as gather information from the users. Forms can perform many tasks that depend on the nature of your business requirements and logic. React offers a stateful, reactive approach to build a form. The component rather than the DOM usually handles the React form.
There are two types of form input in React.

- Uncontrolled components-
  It is similar to the traditional HTML form inputs. Here, the form data is handled by the DOM itself. It maintains its own state and will be updated when the input value changes. To write an uncontrolled component, there is no need to write an event handler for every state update, and you can use a ref to access the value of the form from the DOM.
- Controlled components-
  A controlled component is bound to a value, and its changes will be handled in code by using **event-based callbacks**. Here, the input form element is handled by the react itself rather than the DOM. In this, the mutable state is kept in the state property and will be updated only with the setState() method.

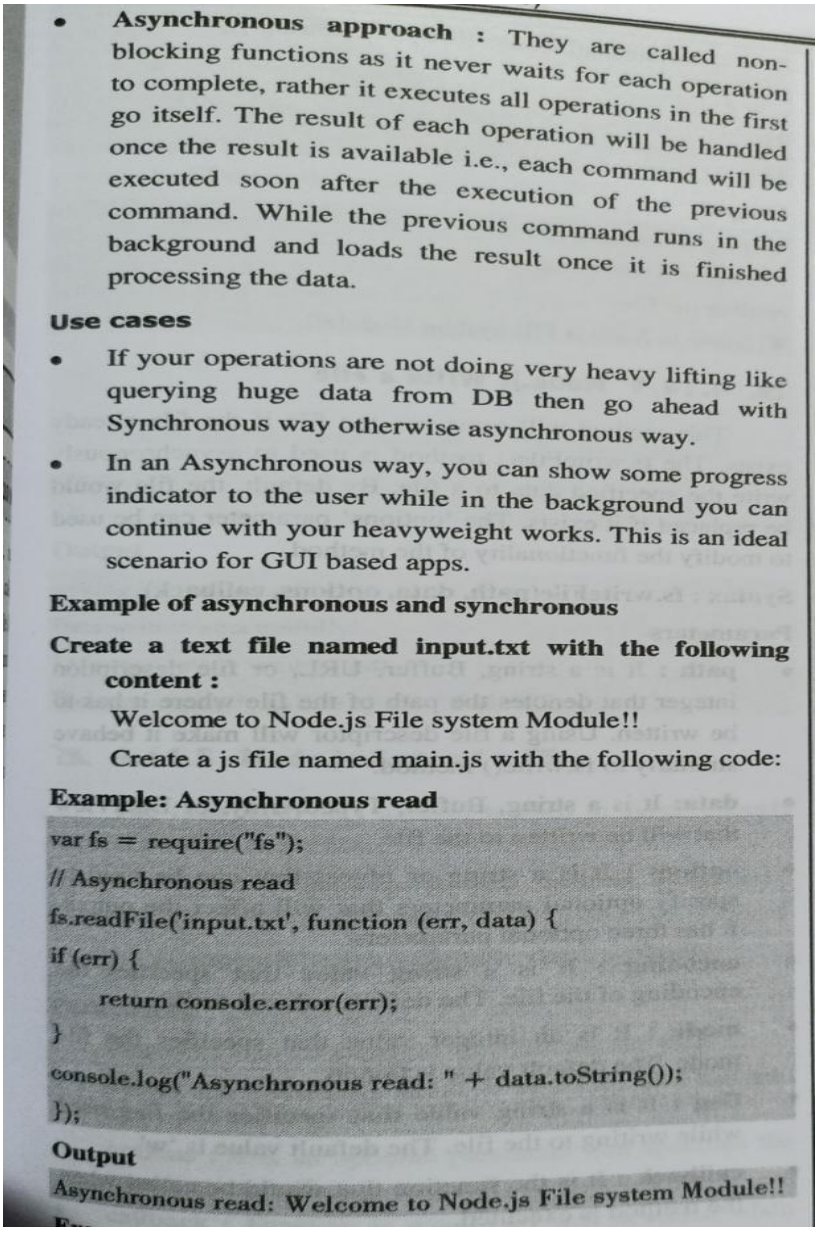
4. **What is an Express Generator?**

Express Generator is a Node.js Framework like ExpressJS which is used to create Express Applications easily and quickly. It acts as a tool for

generating express applications.
Features of Express-Generator:

- It generates express Applications in one go using only one command
- The generated site has a modular structure that we can modify according to our needs for our web application.
- The generated file structure is easy to understand.
- We can also configure options while creating our site like which type of view we want to use (For example, ejs, pug, and handlebars).

5. How is Asynchronous Programming done in Node js?



- **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.

Example of asynchronous and synchronous

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

Welcome to Node.js File system Module!!

Create a js file named main.js with the following code:

Example: Asynchronous read

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

Output

```
Asynchronous read: Welcome to Node.js File system Module!!
```

6. **Explain Props. What do you mean by Default props?**

Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and works similar to the HTML attributes. It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.
Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.
It is not necessary to always add props in the reactDom.render() element. You can also set default props directly on the component constructor.
Eg.

```
import React, { Component } from 'react';
class App extends React.Component {
  render() {
      return (
          <div>
            <h1>Default Props Example</h1>
            <h3>Welcome to {this.props.name}</h3>
            <p>Javatpoint is one of the best Java training institutes in Noida,
Delhi, Gurugram, Ghaziabad and Faridabad.</p>
          </div>
        );
    }
}
App.defaultProps = {
   name: "JavaTpoint"
}
export default App;
```

7. **Explain REPL**

The Read-Eval-Print Loop or REPL is a shell interface. This interface reads and evaluates each line of input and then prints the result. The Read-Eval-Print Loop helps us to interact with our application runtime present in a specific state. The commands are read and evaluated by the REPL and print the result. After printing the result, REPL goes back to the start to read, evaluate and print our next input.
Read − Reads user's input, parses the input into JavaScript data-structure, and stores it in memory.
Eval − Takes and evaluates the data structure.

Print − Prints the result.
Loop − Loops the above command until the user presses ctrl-c twice.

[

8. Difference between Stateful and Stateless Component:

Stateful Components	Stateless Components.
1. Components with state (lol). 2. Used when we need to add functionality. 3. Components have state object. 4. They keep a track of change in data via state objects. 5. Also known as smart components.	1. Without state. 2. Used when we just need to display on screen. 3. Do not have a state object. 4. Rendered via props. Always render the same thing. 5. Also known as presentational or dumb components.

9. **Explain Cookies in Express**

cookies are small strings that contain key-value pairs of information sent from the web server to the browser to get information about the user. The browser will then save them locally. This way, subsequent requests can be made to the server to immediately update user content on the website depending on the previous requests that a user made to the server. A cookie is HTTP generated; thus, called an HTTP cookie.
Now to use cookies with Express, we will require the cookie-parser.
cookie-parser - cookie-parser looks at the headers in between the client and the server transactions, reads these headers, parses out the cookies being sent, and saves them in a browser. In other words, cookie-parser will help us create and manage cookies depending on the request a user makes to the server. To use it, we will require it in our index.js file; this can be used the same way as we use other middleware. Here, we will use the

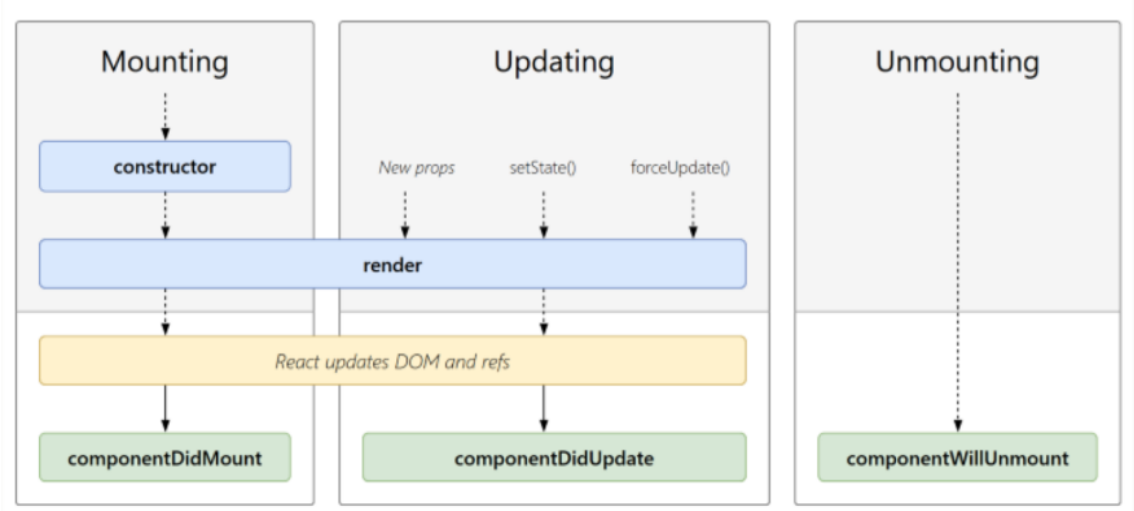
following code.

```
var cookieParser = require('cookie-parser');
app.use(cookieParser());
var express = require('express');
var app = express();
app.get('/', function(req, res){
res.cookie('name', 'express').send('cookie set'); //Sets name = express
});
app.listen(3000);
```

To check if your cookie is set or not, just go to your browser, fire up the console, and enter −

```
console.log(document.cookie);
```

10. Explain the Component Lifecycle with a neat Diagram



In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle.

Mounting

means putting elements into the DOM.React has four built-in methods that gets called, in this order, when mounting a component:

constructor()
getDerivedStateFromProps()
render()
componentDidMount()

The render() method is required and will always be called, the others are optional and will be called if you define them.

Updating

The next phase in the life cycle is when a component is *updated*.

A component is updated whenever there is a change in the component's state or props.

React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

Unmounting
Unmounting

The next phase in the lifecycle is when a component is removed from the DOM, or *unmounting* as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

- componentWillUnmount()

The componentWillUnmount method is called when the component is about to be removed from the DOM.