1. How does React work?

React creates a virtual DOM. When state changes in a component it firstly runs a "diffing" algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of diff.

1. What are the advantages of ReactJS?

* Increases the application’s performance with Virtual DOM
* JSX makes code is easy to read and write
* It renders both on client and server side
* Easy to integrate with other frameworks (Angular, BackboneJS) since it is only a view library
* Easy to write UI Test cases and integration with tools such as JEST.

1. What is props in React?

Props are inputs to a React component. They are single values or objects containing a set of values that are passed to React Components on creation using a naming convention similar to HTML-tag attributes. i.e, They are data passed down from a *parent component* to a *child component*.

The primary purpose of props in React is to provide following component functionality:

* Pass custom data to your React component.
* Trigger *state* changes.
* Use via *this.props.reactProp* inside component's render() method.

1. Which are the most important features of MongoDB?

* Flexible data model in form of documents
* Agile and highly scalable database
* Faster than traditional databases
* Expressive query language

1. What Is Replication In MongoDB?

Replication is the process of **synchronizing data across multiple servers**. Replication provides **redundancy** and **increases data availability**. With multiple copies of data on different database servers, replication protects a database from the loss of a single server. Replication also allows you to recover from hardware failure and service interruptions.

1. What are Higher-Order components?

A higher-order component (HOC) is a function **that takes a component and returns a new component**. Basically, it’s a pattern that is derived from React’s compositional nature We call them as “pure’ components” because they can accept any dynamically provided child component but they won’t modify or copy any behavior from their input components.

const EnhancedComponent = higherOrderComponent(WrappedComponent);

HOC can be used for many use cases as below,

* Code reuse, logic and bootstrap abstraction
* Render High jacking
* State abstraction and manipulation
* Props manipulation

1. What are the differences between a class component and functional component?

**Class Components**

* Class-based Components uses ES6 class syntax. It can make use of the lifecycle methods.
* Class components extend from React.Component.
* In here you have to use this keyword to access the props and functions that you declare inside the class components.

**Functional Components**

* Functional Components are simpler comparing to class-based functions.
* Functional Components mainly focuses on the UI of the application, not on the behavior.
* To be more precise these are basically render function in the class component.
* Functional Components can have state and mimic lifecycle events using Reach Hooks

1. What are the key features of Node.js?

* **Asynchronous event driven IO** **helps concurrent request handling** – All APIs of Node.js are asynchronous. Thus it will not wait for the response from the previous requests.
* **Fast in Code execution** – Node.js uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome.
* **Single Threaded but Highly Scalable** – Node.js uses a single thread model for event looping. The response from these events may or may not reach the server immediately. However, this does not block other operations. Thus making Node.js highly scalable.
* **Node.js library uses JavaScript** – The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.
* **There is an Active and vibrant community for the Node.js framework** – The active community always keeps the framework updated with the latest trends in the web development.
* **No Buffering** – Node.js applications never buffer any data. They simply output the data in chunks.

1. What are the limitations of React?

* React is just a view library, not a full-blown framework
* There is a learning curve for beginners who are new to web development.
* Integrating React.js into a traditional MVC framework requires some additional configuration
* The code complexity increases with inline templating and JSX.
* Too many smaller components leading to over-engineering or boilerplate

1. What do you mean by Asynchronous API?

All APIs of Node.js library are aynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.

1. What is Callback Hell?

The asynchronous function requires callbacks as a return parameter. When multiple asynchronous functions are chained together then callback hell situation comes up.

1. What is Reconciliation?

When a component’s props or state change, React decides **whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one**. When they are not equal, React will update the DOM. This process is called reconciliation.

1. What is the difference between returning a callback and just calling a callback?

return callback();

//some more lines of code; - won't be executed

callback();

//some more lines of code; - will be executed

Of course returning will help the context calling async function get the value returned by callback.

1. When should we embed one document within another in MongoDB?

You should consider embedding documents for:

* contains relationships between entities
* One-to-many relationships
* Performance reasons

1. Does Mongodb Support Foreign Key Constraints?

No. MongoDB does not support such relationships. The database does not apply any constraints to the system (i.e.: foreign key constraints), so there are no "cascading deletes" or "cascading updates". Basically, in a NoSQL database it is up to you to decide how to organise the data and its relations if there are any.

1. Explain advantages of BSON over JSON in MongoDB?

* BSON stands for Binary Javascript Object Notation. It is a binary-encoded serialization of JSON documents. BSON has been extended to add some optional non-JSON-native data types, like dates and binary data.
* BSON is designed to be efficient in space, but in some cases is not much more efficient than JSON. In some cases BSON uses even more space than JSON. The reason for this is another of the BSON design goals: traversability. BSON adds some "extra" information to documents, like length of strings and subobjects. This makes traversal faster.
* BSON is also designed to be fast to encode and decode. For example, integers are stored as 32 (or 64) bit integers, so they don't need to be parsed to and from text. This uses more space than JSON for small integers, but is much faster to parse.
* In addition to compactness, BSON adds additional data types unavailable in JSON, notably the BinData and Date data types.

1. How Node prevents blocking code?

By providing callback function. Callback function gets called whenever corresponding event triggered.

1. How can you achieve transaction and locking in MongoDB?

To achieve concepts of transaction and locking in MongoDB, we can use the nesting of documents, also called embedded (or sub) documents. MongoDB supports atomic operations within a single document.

1. How does Node.js handle child threads?

Node.js, in its essence, is a single thread process. It does not expose child threads and thread management methods to the developer. Technically, Node.js does spawn child threads for certain tasks such as asynchronous I/O, but these run behind the scenes and do not execute any application JavaScript code, nor block the main event loop.

If threading support is desired in a Node.js application, there are tools available to enable it, such as the ChildProcess module.

1. How to avoid Callback Hell in Node.js?

* **Make your program modular** - It proposes to split the logic into smaller modules. And then join them together from the main module to achieve the desired result.
* **Use async/await mechanism** - Async /await is another alternative for consuming promises, and it was implemented in ES8, or ES2017. Async/await is a new way of writing promises that are based on asynchronous code but make asynchronous code look and behave more like synchronous code.
* **Use promises mechanism** - Promises give an alternate way to write async code. They either return the result of execution or the error/exception. Implementing promises requires the use of .then() function which waits for the promise object to return. It takes two optional arguments, both functions. Depending on the state of the promise only one of them will get called. The first function call proceeds if the promise gets fulfilled. However, if the promise gets rejected, then the second function will get called.
* **Use generators** - Generators are lightweight routines, they make a function wait and resume via the yield keyword. Generator functions uses a special syntax function\* (). They can also suspend and resume asynchronous operations using constructs such as promises or thunks and turn a synchronous code into asynchronous.

function\* HelloGen() {

yield 100;

yield 400;

}

var gen = HelloGen();

console.log(gen.next()); // {value: 100, done: false}

console.log(gen.next()); // {value: 400, done: false}

console.log(gen.next()); // {value: undefined, done: true}

1. How to query MongoDB with “like”?

db.users.find({"name": /.\*m.\*/})

// or

db.users.find({"name": /m/})

1. If Node.js is single threaded then how it handles concurrency?

Node provides a single thread to programmers so that code can be written easily and without bottleneck. **Node internally uses multiple POSIX threads** for various I/O operations such as File, DNS, Network calls etc.

When Node gets I/O request it creates or uses a thread to perform that I/O operation and once the operation is done, it pushes the result to the event queue. On each such event, event loop runs and checks the queue and if the execution stack of Node is empty then it adds the queue result to execution stack.

This is how Node manages concurrency.

1. Rewrite promise-based Node.js applications to **Async/Await**

Rewrite this code to Async/Await:

function PromiseTask() {

return functionA()

.then((valueA) => functionB(valueA))

.then((valueB) => functionC(valueB))

.then((valueC) => functionD(valueC))

.catch((err) => logger.error(err))

}

Answer

async function asyncTask() {

try {

const valueA = await functionA()

const valueB = await functionB(valueA)

const valueC = await functionC(valueB)

return await functionD(valueC)

} catch (err) {

logger.error(err)

}

}

1. What are Pure Components?

PureComponent is exactly the same as Component except that it handles the **shouldComponentUpdate** method for you. When props or state changes, **PureComponent** will do a shallow comparison on both props and state. Component, on the other hand, won’t compare current props and state to next out of the box. Thus, the component will re-render by default whenever **shouldComponentUpdate** is called.

1. What are React Hooks?

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class. With Hooks, you can extract stateful logic from a component so it can be tested independently and reused. Hooks allow you to reuse stateful logic without changing your component hierarchy. This makes it easy to share Hooks among many components or with the community.

1. What are advantages of using React Hooks?

Primarily, hooks in general enable the extraction and reuse of stateful logic that is common across multiple components without the burden of higher order components or render props. Hooks allow to easily manipulate the state of our functional component without needing to convert them into class components.

Hooks don’t work inside classes (because they let you use React without classes). By using them, we can totally avoid using lifecycle methods, such as **componentDidMount**, **componentDidUpdate**, **componentWillUnmount**. Instead, we will use built-in hooks like **useEffect** .

1. What is Aggregation in MongoDB?

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. MongoDB provides three ways to perform aggregation:

* the aggregation pipeline,
* the map-reduce function,
* and single purpose aggregation methods and commands.

db.users.aggregate({

$lookup:{

from:"posts",

localField:"\_id",

foreignField:"userid",

as :"postdata"

}

})

1. What is ReactDOM?

It's a top-level React API to render a React element into the DOM, via the ReactDOM.render method.

1. What is Sharding in MongoDB?

Sharding is a method for storing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

1. What is Stream and what are types of Streams available in Node.js?

Streams are a collection of data that might not be available all at once and don’t have to fit in memory. Streams provide chunks of data in a continuous manner. It is useful to read a large set of data and process it.

There is four fundamental type of streams:

* Readable.
* Writeable.
* Duplex.
* Transform.

**Readable** streams as the name suggest used in reading a large chunk of data from a source. **Writable** streams are used in writing a large chunk of data to the destination.

**Duplex** streams are both readable and writable ( Eg socket). **Transform** stream is the duplex stream which is used in modifying the data (eg zip creation).

1. What is prop drilling and how can you avoid it?

When building a React application, there is often the need for a deeply nested component to use data provided by another component that is much higher in the hierarchy. The simplest approach is to simply pass a prop from each component to the next in the hierarchy from the source component to the deeply nested component. This is called prop drilling.

The primary disadvantage of prop drilling is that components that should not otherwise be aware of the data become unnecessarily complicated and are harder to maintain.

To avoid prop drilling, a common approach is **to use React context**. This allows a Provider component that supplies data to be defined, and allows nested components to consume context data via either a **Consumer** component or a **useContext hook**

1. What is Key and benefit of using it in lists?

A **key** is a special string attribute you need to include when creating lists of elements. Keys help React identify which items have changed, are added, or are removed.

For example, most often we use IDs from your data as keys

const todoItems = todos.map((todo) =>

<li key={todo.id}>

{todo.text}

</li>

);

1. What is a Blocking Code

If application has to wait for some I/O operation in order to complete its execution any further then the code responsible for waiting is known as blocking code

1. What is the alternative of binding this in the constructor?

You can use property initializers to correctly bind callbacks. This is enabled by default in create react app. You can use an arrow function in the callback. The problem here is that a new callback is created each time the component renders

1. What is the difference between ShadowDOM and VirtualDOM?

**Virtual DOM**

Virtual DOM is about avoiding unnecessary changes to the DOM, which are expensive performance-wise, because changes to the DOM usually cause re-rendering of the page. Virtual DOM also allows to collect several changes to be applied at once, so not every single change causes a re-render, but instead re-rendering only happens once after a set of changes was applied to the DOM.

**Shadow DOM**

**Shadow dom is mostly about encapsulation of the implementation.** A single custom element can implement more-or-less complex logic combined with more-or-less complex DOM. An entire web application of arbitrary complexity can be added to a page by an import and <body><my-app></my-app> but also simpler reusable and composable components can be implemented as custom elements where **the internal representation is hidden in the shadow DOM like <date-picker></date-picker>.**

1. What's the Event Loop?

The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.

1. What's the difference between **useRef** and **createRef**?

**createRef** will always create a new ref. In a class-based component, you would typically put the ref in an instance property during construction **(e.g. this.input = createRef())**. You don't have this option in a function component.

**useRef** takes care of returning the same ref each time as on the initial rendering.

1. What's the difference between a "smart" component and a "dumb" component?

* Smart components manage their state or in a Redux environment are connected to the Redux store.
* Dumb components are driven completely by their props passed in from their parent and maintain no state of their own.

1. How to apply validation on Props in ReactJS?

When the application is running in development mode, React will automatically check for all props that we set on components to make sure they must right correct and right data type. For incorrect type, it will generate warning messages in the console for development mode whereas it is disabled in production mode due performance impact. The mandatory prop is defined with **isRequired**.

The set of predefined prop types are below

* *React.PropTypes.string*
* *React.PropTypes.number*
* *React.PropTypes.func*
* *React.PropTypes.node*
* *React.PropTypes.bool*

import PropTypes from 'prop-types';

class User extends React.Component {

render() {

return (

<h1>Welcome, {this.props.name}</h1>

<h2>Age, {this.props.age}

);

}

}

User.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number.isRequired

};

1. Is it possible to use Class in Node.js?

With ES6, you are able to make "actual" classes just like this:

class Animal {

constructor(name) {

this.name = name;

}

print() {

console.log('Name is :' + this.name);

}

}

module.exports = class Animal {

};

var Animal = require('./Animal');

class Cat extends Animal {

...

}

1. Update MongoDB field using value of another field

db.person.find().snapshot().forEach(

function (elem) {

db.person.update(

{

\_id: elem.\_id

},

{

$set: {

name: elem.firstname + ' ' + elem.lastname

}

}

);

}

);

1. Why would you need to bind event handlers to this?

Binding is not something that is specific to React, but rather how this works in Javascript. When you define a component using an ES6 class, a common pattern is for an event handler to be a method on the class. In JavaScript, class methods are not bound by default. If you forget to bind **this.someEventHandler** and pass it to **onChange**, this will be undefined when the function is actually called.

Generally, if you refer to a method without () after it, such as **onChange={this.someEventHandler},** you should bind that method.

1. Mention the purpose of MongoDB

MongoDB is defined as the document acquainted database executive which is designed for storing the data. MongoDB stores the data under the format of binary JSON, which implements the theory of anthology and documentation.

MongoDB has a NoSQL database equipped with high performance, high scalability, and seamless query and indexing flexibility

1. Mention the purpose of ExpressJS

ExpressJS can be explained as the web application framework designed to support and host NodeJS projects. ExpressJS is the open-source framework that is available under MIT. It manages the workflow between the front end and the database, facilitating the smooth and secure transfer of the data.

In addition, ExpressJS is filled with excellent error-handling web design functionality for optimizing web development procedures.

1. Explain Mongoose

A mongoose is the object document mapper used for defining objects through a strongly typed schema that can be mapped to the MongoDB document. Thus, Mongoose offers a schema-based solution for modeling the application data.

In addition, Mongoose had built-in typecasting, validation, query building, business logic hooks, and other features.

1. What is data modeling?

This term is being used in the context of Mongoose and MongoDB. Data modeling **is the procedure of creating the data model for data at hand and after which it can be stored in a database.** The data model represents the data object, its relation, and rules which define the connections.

With data modeling, **data visualization can be represented while enforcing the data’s business rules, compliance, and policy.** Data modeling is executed to **ensure consistency on convention, values, semantics, security, and data quality.**

1. Define REPL under NodeJS

REPL is Read Eval Print Loop, the program for accepting commands, evaluating them, and printing results. Thus, REPL does the same that Unix or Linux used to create an environment for entering commands and systems that will respond to the output.

1. What is containerization?

This is **the alternative to traditional hypervisor machine virtualization, which involves encapsulatin**g the application in the container within the operating environment. Under containerization**, sharing of different containers is done instead of cloning the operating system** for virtual machines.

1. What is the purpose of indexing in MongoDB?

Indexes are being used for supporting and facilitating the execution of queries in MongoDB. MongoDB needs to scan every document if the index is not there and then select the appropriate form that matches the query statement.

Or, if the query has an index assigned to it, MongoDB can use the index to limit the number of documents.

1. Share the difference between classes and interface in TypeScript

They both are the structures that are responsible for promoting object-oriented programming and also checking type in TypeScript.

But under the class, a blueprint is there, which allows the creation of objects collection which shares the same configuration.

At the same time, the interface is the collection of properties and methods which describe the thing. It doesn’t provide implementation or initialization of the objects.

1. Explain decorators in typescript

A decorator is a declaration attached to the class declaration, method, accessor, property, or parameter. They are functions that take their target as the argument. Decorators will allow you to run the arbitrary code in the target execution or replace the target with a new one.

function first() {

console.log("first(): factory evaluated");

return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {

console.log("first(): called");

};

}

function second() {

console.log("second(): factory evaluated");

return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {

console.log("second(): called");

};

}

class ExampleClass {

@first()

@second()

method() {}

}

1. Explain event emitter under NodeJS

Node.js uses events module to create and handle custom events. The **EventEmitter** class can be used to create and handle custom events module.

All EventEmitters emit the event **newListener** when new listeners are added and **removeListener** when existing listeners are removed.

const EventEmitter = require('events').EventEmitter;

let emitter = new EventEmitter()

const handlerFn = (data, isTrue) => {

    console.log("Foo Event handler :", data.msg)

    console.log(isTrue)

}

emitter.on("foo", (data, isTrue) => {

    console.log("Inside Foo Event :", data.msg)

    isTrue = false //Immutable

    data.msg = "Good Bye" //Mutable

    emitter.removeListener("foo", handlerFn)

})

emitter.on("foo", handlerFn)

emitter.emit("foo", { msg: "Hello World" }, true)

1. State the feature which is utilized under the NodeJS for importing outside libraries

The summon require is being utilized for bringing in the external libraries. Example **“var http=require (“http”)”.** It will stack up the HTTP library from which the single sent out for protesting through HTTP available.

1. State the kind of API work in NodeJS

Mainly the two variants of API work under NodeJS

* Synchronous, which has to block capacities.
* Asynchronous, which have non-blocking capacities.

1. Define asynchronous API

Every API of Node is asynchronous, which means that it is non-blocking. Asynchronous API means that the server of NodeJS will never wait for API to return data.

Instead, the server will move to the following API after calling it from a notification mechanism of events of NodeJS. Thus, it helps in getting the response from previous API calls.

1. What is the default scope of Node.js application

Local

1. Which module is used to serve static files in Node.js

node-static

1. What is a Document in MongoDB

A Document in MongoDB is an ordered set of keys with associated values. It is represented by a map, hash, or dictionary.

1. What is the Mongo Shell

It is a JavaScript shell that allows interaction with a MongoDB instance from the command line. With that one can perform administrative functions, inspecting an instance, or exploring MongoDB.

1. How do you Delete a Document in MongoDB

The CRUD API in MongoDB provides **deleteOne** and **deleteMany** for this purpose. Both of these methods take a filter document as their first parameter. The filter specifies a set of criteria to match against in removing documents.

1. What is routing and how routing works in Express.js

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on). Each route can have one or more handler functions, which are executed when the route is matched.

1. What is Middleware in Express.js

Middleware is a function that is invoked by the Express routing layer before the final request handler.

const connect = require("connect");

const http = require("http");

const app = connect();

const books = (bookName) => {

    return (req, res) => {

        console.log("BookName :", bookName)

        return res.end("Books coming soon...")

    }

}

app.use("/books", books("The book"))

let server = http.createServer(app)

server.listen(9090, "localhost", () => console.log("HTTP Server started..."))

1. How Do I Render Plain Html in Express

There’s no need to “render” HTML with the **res.render()** function. If you have a specific file, use the **res.sendFile()** function. If you are serving many assets from a directory, use the **express.static()** middleware function.

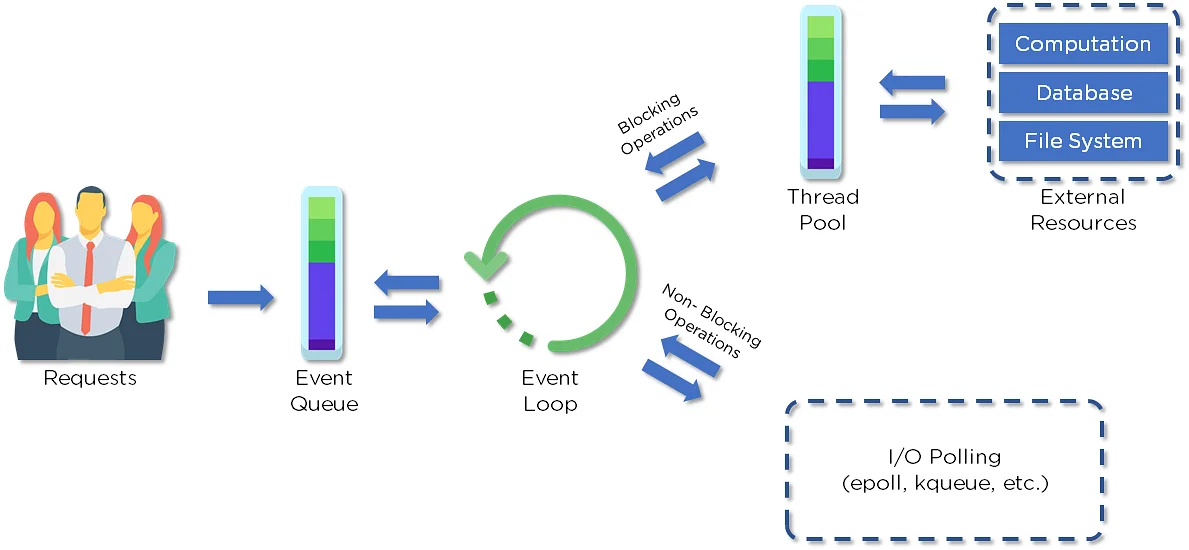
**NODE.JS**

1. What is Node.js? Where can you use it?

Node.js is an open-source, cross-platform JavaScript runtime environment and library to run web applications outside the client’s browser. It is used to create server-side web applications.

Node.js is perfect for data-intensive applications as it uses an asynchronous, event-driven model. You can use I/O intensive web applications like video streaming sites. You can also use it for developing: Real-time web applications, Network applications, General-purpose applications, and Distributed systems

1. How does Node.js work?



1. Why is Node.js Single-threaded?

Node.js is single-threaded for async processing. By doing async processing on a single-thread under typical web loads, more performance and scalability can be achieved instead of the typical thread-based implementation.

1. If Node.js is single-threaded, then how does it handle concurrency?

The Multi-Threaded Request/Response Stateless Model is not followed by the Node JS Platform, and it adheres to the Single-Threaded Event Loop Model. The Node JS Processing paradigm is heavily influenced by the **JavaScript Event-based model and the JavaScript callback** system. As a result, Node.js can easily manage more concurrent client requests. **The event loop is the processing** model's beating heart in Node.js.

1. Explain callback in Node.js

A callback function is called after a given task. It allows other code to be run in the meantime and prevents any blocking. Being an asynchronous platform, Node.js heavily relies on callback. All APIs of Node are written to support callbacks.

1. What are the advantages of using promises instead of callbacks?

* The control flow of asynchronous logic is more specified and structured.
* The coupling is low.
* We've built-in error handling.
* Improved readability.

1. What is NPM

NPM stands for Node Package Manager, responsible for managing all the packages and modules for Node.js.

Node Package Manager provides two main functionalities:

* Provides online repositories for node.js packages/modules, which are searchable on search.nodejs.org
* Provides command-line utility to install Node.js packages and also manages Node.js versions and dependencies

1. What are the modules in Node.js?

Modules are like JavaScript libraries that can be used in a Node.js application to include a set of functions. To include a module in a Node.js application, use the require() function with the parentheses containing the module's name.

|  |  |
| --- | --- |
| Core Modules | Description |
| HTTP | Includes classes, methods, and events to create a Node.js HTTP server |
| Util | Includes utility functions useful for developers |
| Fs | Includes events, classes, and methods to deal with file I/O operations |
| url | Includes methods for URL parsing |
| query string | Includes methods to work with query string |
| Stream | Includes methods to handle streaming data |
| Zlib | Includes methods to compress or decompress files |

1. What is the purpose of the module .Exports?

In Node.js, a module encapsulates all related codes into a single unit of code that can be parsed by moving all relevant functions into a single file. You may export a module with the module and export the function, which lets it be imported into another file with a needed keyword.

1. Which database is more popularly used with Node.js?

MongoDB is the most common database used with Node.js. It is a NoSQL, cross-platform, document-oriented database that provides high performance, high availability, and easy scalability.

1. What are some of the most commonly used libraries in Node.js?

There are two commonly used libraries in Node.js:

* **ExpressJS** - Express is a flexible Node.js web application framework that provides a wide set of features to develop web and mobile applications.
* **Mongoose** - Mongoose is also a Node.js web application framework that makes it easy to connect an application to a database.

1. What does event-driven programming mean?

An event-driven programming approach uses events to trigger various functions. An event can be anything, such as typing a key or clicking a mouse button. A call-back function is already registered with the element executes whenever an event is triggered.

1. What is an Event Loop in Node.js?

Event loops handle asynchronous callbacks in Node.js. It is the foundation of the non-blocking input/output in Node.js, making it one of the most important environmental features.

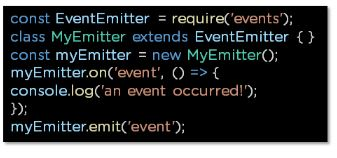
1. Differentiate between process.nextTick() and setImmediate()?

**process.nextTick()** postpones the execution of action until the next pass around the event loop, or it simply calls the callback function once the event loop's current execution is complete, whereas **setImmediate()** executes a callback on the next cycle of the event loop and returns control to the event loop for any I/O operations.

1. What is an EventEmitter in Node.js?

EventEmitter is a class that holds all the objects that can emit events

Whenever an object from the EventEmitter class throws an event, all attached functions are called upon synchronously



1. What are the two types of API functions in Node.js?

The two types of API functions in Node.js are:

* Asynchronous, non-blocking functions
* Synchronous, blocking functions

1. What is the package.json file?

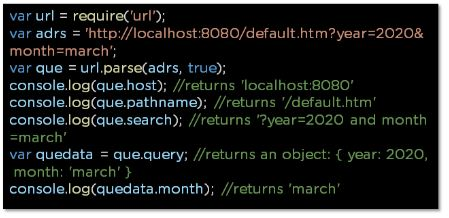
The package.json file is the heart of a Node.js system. This file holds the metadata for a particular project. The package.json file is found in the root directory of any Node application or module

This is what a package.json file looks like immediately after creating a Node.js project using the command: npm init



1. How would you use a URL module in Node.js?

The URL module in Node.js provides various utilities for URL resolution and parsing. It is a built-in module that helps split up the web address into a readable format.



1. How do you create a simple Express.js application?

import express,{Request,Response} from 'express';

const app =express()

app.use(express.json())

let notes =[

    {id:100,title:"Shopping",body:"Go to Shopping"},

    {id:101,title:"Playing",body:"Go to Play"},

    {id:102,title:"Sleeping",body:"Go to Sleep"}

]

// Endpoint - http://localhost:9000/

app.get('/', (req:Request, res:Response) =>{

  return res.send("Hello from GET endpoint")

})

// Endpoint - http://localhost:9000/notes

app.get('/notes', (req:Request, res:Response) =>{

    res.json(notes)

})

// Endpoint - http://localhost:9000/notes/101

app.get('/notes/:noteId', (req:Request, res:Response) =>{

  const {noteId}=req.params;

  const notefound = notes.find(n=>n.id === +noteId)

  if(notefound){

    console.log(req.params)

    res.send(req.params)

      return res.status(200).send({...notefound})

  }

  else{

      return res.status(404).send({error:`No results found for Note ${noteId}`})

  }

})

// Delete Endpoint - http://localhost:9000/notes/101

app.delete('/notes/:noteId', (req:Request, res:Response) =>{

    const { noteId } = req.params;

    const noteFound = notes.find(n => n.id === +noteId)

    if(noteFound){

        const duplicatedNotes = notes.filter(n => n.id !== +noteId)

        notes = [...duplicatedNotes]

        return res.status(200).send({...noteFound})

    }else{

        return res.status(404).send({message : "Not Found"})

    }

  })

// Post Endpoint - http://localhost:9000/notes

app.post("/notes", (req, res) =>{

    const { title,body }= req.body;

    if(title&&body){

        let newNotes ={title,body,id:(notes[notes.length - 1].id+1)}

        notes.push(newNotes)

        return res.status(201).send({...newNotes})

    }

    else

    return res.status(401).send({err : "Title/body Not Found"})

})

//Patch Endpoint - http://localhost:9000/notes/101

app.patch("/notes/:noteId", (req, res) =>{

    const { noteId } = req.params;

    const position = notes.findIndex(n => n.id === +noteId)

    if(position >= 0){

        const { title, body } = req.body;

        if(title && body){

            notes[position].title = title;

            notes[position].body = body;

            return res.send({...notes[position]})

        }else{

            return res.send({error : "Invalid Body"})

        }

    }else{

        return res.send({error : "Note ID does not exist"})

    }

})

app.listen(9000, "localhost", () =>{

    console.log("Server Running on Port 9000")

})

1. What are streams in Node.js?

Streams are objects that enable you to read data or write data continuously.

There are four types of streams:

* Readable – Used for reading operations
* Writable − Used for write operations
* Duplex − Can be used for both reading and write operations
* Transform − A type of duplex stream where the output is computed based on input

1. How do you create a simple server in Node.js that returns Hello World?



const http = require("http")

const url = require("url")

const requestListner = (req, res) => {

    const queryString = url.parse(req.url, true).query;

    const { key, id, name } = queryString

    console.log("Query String :", { key, id, name })

    res.end("Query Received")

}

const server = http.createServer(requestListner)

server.listen("9090", "localhost", () => console.log("HTTP Server started at port 9090"))

1. Explain asynchronous and non-blocking APIs in Node.js.

All Node.js library APIs are asynchronous, which means they are also non-blocking

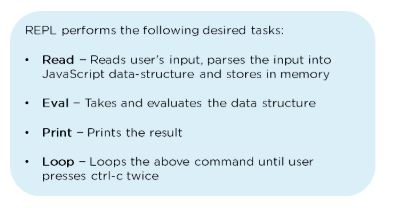
A Node.js-based server never waits for an API to return data. Instead, it moves to the next API after calling it, and a notification mechanism from a Node.js event responds to the server for the previous API call

1. How do we implement async in Node.js?



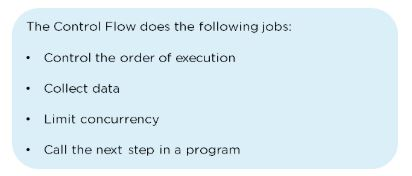
1. What is REPL in Node.js?

REPL stands for Read Eval Print Loop, and it represents a computer environment. It’s similar to a Windows console or Unix/Linux shell in which a command is entered. Then, the system responds with an output



1. What is the control flow function? How does control flow manage the function calls?

The control flow function is a piece of code that runs in between several asynchronous function calls.



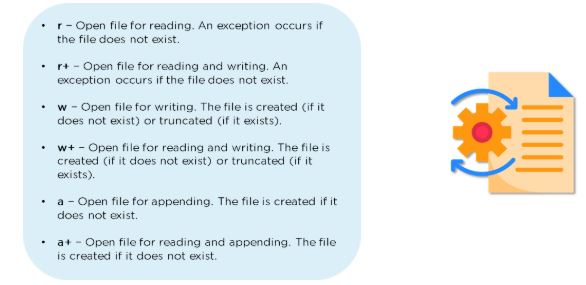
1. What is the buffer class in Node.js?

Buffer class stores raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap. Buffer class is used because pure JavaScript is not compatible with binary data

1. What is piping in Node.js?

Piping is a mechanism used to connect the output of one stream to another stream. It is normally used to retrieve data from one stream and pass output to another stream

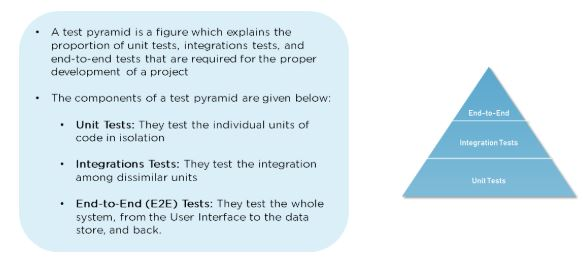
1. What are some of the flags used in the read/write operations in files?



1. What is callback hell?

* Callback hell, also known as the **pyramid of doom**, is the result of intensively nested, unreadable, and unmanageable callbacks, which in turn makes the code harder to read and debug
* improper implementation of the asynchronous logic causes callback hell

1. What is a test pyramid in Node.js?



1. Explain the concept of middleware in Node.js.

Middleware is a function that receives the request and response objects. Most tasks that the middleware functions perform are:

* Execute any code
* Update or modify the request and the response objects
* Finish the request-response cycle
* Invoke the next middleware in the stack

1. What are the different types of HTTP requests?

HTTP defines a set of request methods used to perform desired actions. The request methods include:

* GET: Used to retrieve the data
* POST: Generally used to make a change in state or reactions on the server
* HEAD: Similar to the GET method, but asks for the response without the response body
* DELETE: Used to delete the predetermined resource

1. How would you connect a MongoDB database to Node.js?



**EXPRESS JS**

1. What is Express.js?

* Express.js is a free open-source, light-weight node js based web application framework.
* It is designed for building (single-page, multi-page, and hybrid) web applications and APIs.
* It has been developed by TJ Holowaychuk in 2010 and written in JavaScript.

Express.js Features:

1. **Middlewares**: Set up middlewares in order to respond to HTTP/RESTful Requests.
2. **Routing**: It is possible to defines a routing table in order to perform different HTTP operations.
3. **Templates**: Dynamically renders HTML Pages based on passing arguments to templates.
4. **High** **Performance**: Express prepare a thin layer, therefore, the performance is adequate.
5. **Database** **Support**: Express supports RDBMS as well as NoSQL databases.
6. **MVC** **Support**: Organize the web application into an MVC architecture.
7. Manages everything from routes to rendering view and preforming HTTP request.
8. How to allow CORS in Express.js (Node.js)? Explain with an Example?

**Cross-origin resource sharing (CORS)** is a mechanism that allows restricted resources to be requested from another domain/server.

There are mainly three ways you can do this using

* 1. res.setHeader() - It allow to set only single header
  2. res.header() OR res.set() - It allow to set multiple headers.
  3. express cors module

1. Enable CORS for all resources using res.setHeader().

app.use(function(req, res, next) {

// Website you wish to allow to connect

res.setHeader("Access-Control-Allow-Origin", "\*");

// Request methods you wish to allow

res.setHeader(

"Access-Control-Allow-Methods",

"GET, POST, OPTIONS, PUT, PATCH, DELETE"

);

// Request headers you wish to allow

res.setHeader(

"Access-Control-Allow-Headers",

"X-Requested-With,content-type"

);

// Set to true if you need the website to include cookies in the requests sent

// to the API (e.g. in case you use sessions)

res.setHeader("Access-Control-Allow-Credentials", true);

// Pass to next layer of middleware

next();

});

2. Enable CORS for all resources using res.header()

app.use(function(req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

res.header(

"Access-Control-Allow-Headers",

"Origin, X-Requested-With, Content-Type, Accept"

);

next();

});

3. Enable CORS for all resources using express CORS module.

var express = require("express");

var cors = require("cors");

var app = express();

app.use(cors());

app.get("/", function(req, res, next) {

res.json({ msg: "This is CORS-enabled for all origins!" });

});

app.listen(80, function() {

console.log("CORS-enabled web server listening on port 80");

});

1. What is Scaffolding in Express.js?

Scaffolding is creating the skeleton structure of application

There are 2 way to do this:

* Express application generator
* Yeoman

1. Serving static files in Express?

#Example:

app.use(express.static('public'))

app.use('/static', express.static(path.join(\_\_dirname, 'public')))

1. What is routing and how routing works in Express.js?

**Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).**

Route Syntax:

**app.METHOD(PATH, HANDLER)**

app.get('/', function (req, res) {

res.send('Express.js Interview Questions')

})

1. Dynamic routing and how it works in Express.js?

When someone **pass parameters in URL i.e. Parametrized URL,** this routing phenomenon is called dynamic routing.

var express = require('express'),

app = express();

app.get('/article/:id', function(req , res){

res.render('article' + req.params.id);

})

1. What is Middleware in Express.js?

**Middleware is a function that is invoked by the Express routing layer before the final request handler.**

Middleware functions can perform the following tasks:

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware function in the stack.

If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

**Type of Middleware:**

1. Application-level middleware
2. Router-level middleware
3. Error-handling middleware
4. Built-in middleware
5. Third-party middleware

1. Application-level middleware:

This kind of middleware method is bind to the app Object using app.use() method. It applies on all routes.

//This middleware will execute for each route.

app.use(function (req, res, next) {

console.log('Current Time:', Date.now())

next()

})

2. Router-level middleware:

Router-level middleware binds to an specific instance of express.Router()

3. Built-in middleware:

Starting with version 4.x, Express no longer depends on Connect.Express has the following built-in middleware functions:

1. express.static serves static assets such as HTML files, images, and so on.
2. express.json parses incoming requests with JSON payloads.
3. express.urlencoded parses incoming requests with URL-encoded payloads.

4. Third-party middleware:

There are a lots of third party middleware, such as

1. Body-parser
2. Cookie-parser
3. Mongoose
4. Sequelize
5. Cors
6. Express-validator

#Example:

var bodyParser = require('body-parser');

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: false }))

Note: Multiple Middleware can be used as an array on a single route.

var middlewareArray = [middleware1, middleware2]

app.get('/home', middlewareArray, function (req, res, next) {

//Code snippets

})

1. Database integration in Express.js?

Express.js supports many RDBMS & NoSQL Ddatabase like

1. MongoDB
2. MySQL
3. Oracle
4. PostgreSQL
5. SQL Server
6. SQLite

#Example: Install MongoDB

>>npm install mongodb

var MongoClient = require('mongodb').MongoClient

MongoClient.connect('mongodb://localhost:27017/test\_db', function (err, db) {

if (err) throw err

db.collection('mammals').find().toArray(function (err, result) {

if (err) throw err

console.log(result)

})

})

1. Error handling in Express.js and How to redirect 404 errors to a page in Express.js?

var express = require('express'),

app = express();

app.use(function (err, req, res, next) {

console.error(err.stack)

res.status(500).send('Something went wrong, Express.js Interview Questions')

})

//To redirect 404 errors

var express = require('express'),

app = express();

app.use(function(req, res, next) {

res.status(404).json({

errorCode: 404,

errorMsg: "route not found"

});

});

1. How to implement JWT authentication in Express app ? Explain with an Example?

import express, { Request, Response } from 'express';

import "./db";

import { UserRouter } from "./router/user.route";

import { PatientRouter } from './router/patient.route';

import cors from "cors";

import { request } from 'http';

import { sign, verify } from "jsonwebtoken";

import { UserModel } from "./model/user.model";

import bcrypt from "bcryptjs";

const SECRET\_KEY = "My Super Secret Key";

let \_token = "";

let \_userToken = '';

const app = express();

app.use(cors());

app.use(express.json());

app.get("/", (req: Request, res: Response) => {

    return res.end("Landing page")

});

app.post("/login", async (req: Request, res: Response) => {

    const { email, password,role } = req.body;

    const hashedPassword = bcrypt.hashSync(password)

    try {

      const foundUser = await UserModel.findOne({ email, hashedPassword,role });

      if (foundUser) {

        \_token = sign({ id: foundUser.\_id }, SECRET\_KEY);

        return res.send({ ...foundUser.\_doc, token: \_token ,message:"success"}).status(200);

      } else {

        return res.send({ error: "User does not exist",message:"failed" }).status(404);

      }

    } catch (err) {

      return res.send({ error: "Error while fetching user" }).status(500);

    }

  });

  const ensureToken = (req: Request, res: Response, next: Function) => {

    let authHeader = req.headers.authorization;       // 'Bearer <token>'

    if(!authHeader){

      return res.send({'error' : "Authorization header not found"})

    }

     \_userToken = authHeader.split(" ")[1]

    next();

  };

app.use("/admin",ensureToken, UserRouter);

app.use("/patient",ensureToken, PatientRouter);

app.listen(9000, "localhost", () => console.log("Server Running at PORT : 9000"));

1. What do you mean by Express JS and what is its use?

Express JS is an application framework that is light-weighted node JS. Variety of versatile, helpful and vital options are provided by this JavaScript framework for the event of mobile additionally as internet applications with the assistance of node JS.

Express JS Use – Express.js could be a light-weight internet application that helps in organizing the net application into MVC design on the server aspect.

1. What function are arguments available to Express JS route handlers?

The arguments which are available to an Express JS route handler-function are-

1. Req: the request object
2. Res: the response object
3. Next (optional): a function that is employed to pass management to 1 of the following route handlers.
4. How to enable debugging in express app?

On UNIX operating system the command would be as follows:

**$ DEBUG=express:\* node index.js**

On Windows the command would be:

**set DEBUG=express:\* & node index.js**

1. What is the use of next in Express JS?

Next -It passes management to a consecutive matching route. OR a operate to pass management to 1 of the following route handlers.

The argument could also be omitted, however, is beneficial in cases wherever you have got a series of handlers and you’d wish to pass management to 1 of the following route handlers, and skip this one.

***app.get('/user details/:id?', function(req, res, next));***

* Req and Res: It represents the request and response objects
* Next: It passes management to a consecutive matching route.

1. Define templating in ExpressJs?

Templating are powerful engine used for removing the cluttering of our server code with HTML. Some of the templates which are with ExpressJs are:

* Pug
* Mustache
* EJS

1. What is cookie and what it does?

Cookie is a data sent from server and stores on the client side. It keeps the information of user’s actions.

These are some following purpose of using cookie:

1. Session management
2. User tracking
3. Personalization
4. How to use cookies in ExpressJs?

Cookies are used in ExpressJs by installing ‘cookie-parser’ middleware.

To install cookie-parser we use command as:

npm install --save cookie-parser

To set new cookie in our application, we define a new route.

app.get('/',function(req, res){

res.cookie('name','express').send('cookie set')

});

1. How to delete cookie in ExpressJs?

The clearCookie method is used to delete cookies in ExpressJs.

clearCookie('cookie\_name');

app.get('/clear\_cookie\_temp',function(req,res)){

clearCookie('temp');

res.send('cookie temp is cleared');

});

1. How to get the full url In ExpressJs?

var port = req.app.settings.port || cfg.port;

res.locals.requested\_url = req.protocol + '://' + req.host + ( port == 80 || port == 443 ? '' : ':'+port ) + req.path;

1. How does an Express code look like?

var express = require('express');

var app = express();

app.get('/', function (req, res) {

res.send('Welcome to JavaTpoint!');

});

var server = app.listen(8000, function () {

var host = server.address().address;

var port = server.address().port;

console.log('Example app listening at http://%s:%s', host, port);

});

**MONGODB**

<https://www.interviewbit.com/mongodb-interview-questions/>

<https://www.javatpoint.com/mongodb-interview-questions>

<https://www.guru99.com/mongodb-interview-questions.html>

<https://www.tutorialspoint.com/mongodb/mongodb_interview_questions.htm>

<https://mindmajix.com/mongodb-interview-questions>