**Question Set-01**

Top of Form

**Q.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.

Bottom of Form

Top of Form

**Q.2What 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 and Use via this.props.reactProp inside component's render() method.

Bottom of Form

Top of Form

**Q.3What 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.

Bottom of Form

Top of Form

**Q.4What 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.

Bottom of Form

Top of Form

**Q.5What 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.

Bottom of Form

Top of Form

**Q.6What 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.

Bottom of Form

Top of Form

**Q.7What 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.

Bottom of Form

Top of Form

**Q.8Does 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.

Bottom of Form

Top of Form

**Q.9How Node prevents blocking code?**

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

Bottom of Form

Top of Form

**Q.10How 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.

Bottom of Form

Top of Form

**Q.11How 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.

Bottom of Form

Top of Form

**Q.12 How to avoid Callback Hell in Node.js?**

Node.js internally uses a single-threaded event loop to process queued events. But this approach may lead to blocking the entire process if there is a task running longer than expected. Node.js addresses this problem by incorporating callbacks also known as higher-order functions. So whenever a long-running process finishes its execution, it triggers the callback associated. Sometimes, it could lead to complex and unreadable code. More the no. of callbacks, longer the chain of returning callbacks would be. There are four solutions which can address the callback hell problem: Make your program modular, Use async/await mechanism, Use promises mechanism and Use generators

Bottom of Form

Top of Form

**Q.13 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.

Bottom of Form

Top of Form

**Q.14What 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.

Bottom of Form

Top of Form

**Q.15What 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.

Bottom of Form

Top of Form

**Q.16What 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.

Bottom of Form

Top of Form

**Q.17What is JSX?**

JSX is a syntax extension to JavaScript and comes with the full power of JavaScript. JSX produces React elements. You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

Bottom of Form

Top of Form

**Q.18What is ReactDOM?**

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

Bottom of Form

Top of Form

**Q.19What 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.

Bottom of Form

Top of Form

**Q.20What 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 and Transform

Bottom of Form

Top of Form

**Q.21What is prop drilling?**

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.

Bottom of Form

Top of Form

**Q.22What is Key?**

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.

Bottom of Form

Top of Form

**Q.23What 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.

Bottom of Form

Top of Form

**Q.24What is the difference between ShadowDOM and VirtualDOM**

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 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 but also simpler reusable and composable components can be implemented as custom elements where the internal representation is hidden in the shadow DOM like.

Bottom of Form

Top of Form

**Q.25What'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. Every I/O requires a callback - once they are done they are pushed onto the event loop for execution. Since most modern kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed.

Bottom of Form

Top of Form

**Q.26What'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.

Bottom of Form

Top of Form

**Q.27What is Mongoose**

Mongoose is an Object Document Mapper (ODM), which means that by using Mongoose, you can define objects with a strongly-typed schema that can be further mapped to a MongoDB document. It offers a schema-based solution for modeling application data. Mongoose comes with built-in typecasting, validation, query building, business logic hooks, and many more out-of-the-box features.

Bottom of Form

Top of Form

**Q.28What is REPL In Node.Js**

REPL or “Read Eval Print Loop” is a simple program that can accept commands, evaluate them, and prints the results. What REPL does is to create an environment that is similar to a Unix/Linux shell or a Window console, wherein you can enter command and system, and it will respond with the output. Here are the functions that REPL performs: READ (This reads the input provided by the user, parses it into JavaScript data structure, and stores it in the memory.), EVAL (This executes the data structure), PRINT (This prints the outcome generated after evaluating the command.) and LOOP (This loops the above command until the user presses Ctrl+C twice.)

Bottom of Form

Top of Form

**Q.29How to check if an object is an array or not in JavaScript**

The best way to find whether an object is instance of a particular class or not using toString method from Object.prototype

Bottom of Form

Top of Form

**Q.30List down the two arguments that async.queue takes as input in Node.js**

Task Function and Concurrency Value

Bottom of Form

Top of Form

**Q.31What is the purpose of module.exports in Node.js**

This is used to expose functions of a particular module or file to be used elsewhere in the project. This can be used to encapsulate all similar functions in a file which further improves the project structure.

Bottom of Form

Top of Form

**Q.32What is node.js streams**

Streams are instances of EventEmitter which can be used to work with streaming data in Node.js. They can be used for handling and manipulating streaming large files(videos, mp3, etc) over the network. They use buffers as their temporary storage.

Bottom of Form

Top of Form

**Q.33What are node.js buffers**

In general, buffers is a temporary memory that is mainly used by stream to hold on to some data until consumed. Buffers are introduced with additional use cases than JavaScript’s Unit8Array and are mainly used to represent a fixed-length sequence of bytes. This also supports legacy encodings like ASCII, utf-8, etc. It is a fixed(non-resizable) allocated memory outside the v8.

Bottom of Form

Top of Form

**Q.34Explain the concept of stub in Node.js**

Stubs are used in writing tests which are an important part of development. It replaces the whole function which is getting tested.

Bottom of Form

Top of Form

**Q.35What is a thread pool and which library handles it in Node.js**

The Thread pool is handled by the libuv library. libuv is a multi-platform C library that provides support for asynchronous I/O-based operations such as file systems, networking, and concurrency.

Bottom of Form

Top of Form

**Q.36How to make node modules available externally**

module.export

Bottom of Form

Top of Form

**Q.37What is the default scope of Node.js application**

Local

Bottom of Form

Top of Form

**Q.38Which module is used to serve static files in Node.js**

node-static

Bottom of Form

Top of Form

**Q.39What 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.

Bottom of Form

Top of Form

**Q.40What 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.

Bottom of Form

Top of Form

**Q.41How 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.

Bottom of Form

Top of Form

**Q.42Explain the process of Sharding.**

Sharding is the process of splitting data up across machines. We also use the term “partitioning” sometimes to describe this concept. We can store more data and handle more load without requiring larger or more powerful machines, by putting a subset of data on each machine.

Bottom of Form

Top of Form

**Q.43What is a Replica Set in MongoDB**

To keep identical copies of your data on multiple servers, we use replication. It is recommended for all production deployments. Use replication to keep your application running and your data safe, even if something happens to one or more of your servers. Such replication can be created by a replica set with MongoDB. A replica set is a group of servers with one primary, the server taking writes, and multiple secondaries, servers that keep copies of the primary’s data. If the primary crashes, the secondaries can elect a new primary from amongst themselves.

Bottom of Form

Top of Form

**Q.44What is Scaffolding in Express.js**

Scaffolding is creating the skeleton structure of application. There are 2 way to do this: Express application generator and Yeoman

Bottom of Form

Top of Form

**Q.45What 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.

Bottom of Form

Top of Form

**Q.46What is Middleware in Express.js**

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

Bottom of Form

Top of Form

**Q.47What Function Arguments Are Available To Express.js Route Handlers**

The arguments available to an Express.js route handler function are: req (the request object), res (the response object) and next (optional, a function to pass control to one of the subsequent route handlers)

Bottom of Form

Top of Form

**Q.48How Can I Authenticate Users in Express**

Authentication is another opinionated area that Express does not venture into. You may use any authentication scheme you wish.

Bottom of Form

Top of Form

**Q.49Which Template Engines Does Express Support**

Express supports any template engine that conforms with the (path, locals, callback) signature.

Bottom of Form

Top of Form

**Q.50How 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.

**Question set-02**

## *Q1*: How does React work?

Answer

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.

## *Q2*: What are React Hooks?

Answer

**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.

## *Q3*: What are props in React?

Answer

**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:

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

## *Q4*: What are the advantages of ReactJS?

Answer

Below are the advantages of ReactJS:

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

## *Q5*: How is React different from AngularJS (1.x)?

Answer

For example, AngularJS (1.x) approaches building an application by extending HTML markup and injecting various constructs (e.g. Directives, Controllers, Services) at runtime. As a result, AngularJS is very opinionated about the greater architecture of your application — these abstractions are certainly useful in some cases, but they come at the cost of flexibility.

By contrast, React focuses exclusively on the creation of components, and has few (if any) opinions about an application’s architecture. This allows a developer an incredible amount of flexibility in choosing the architecture they deem “best” — though it also places the responsibility of choosing (or building) those parts on the developer.

## *Q6*: What Is Replication In MongoDB?

Answer

**Replication** is the process of synchronizing data across multiple servers.

Replication provides redundancy and increases [data availability](https://www.mongodb.com/docs/manual/reference/glossary/#std-term-high-availability). With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.

In some cases, replication can provide increased read capacity as clients can send read operations to different servers. Maintaining copies of data in different data centres can increase data locality and availability for distributed applications. You can also maintain additional copies for dedicated purposes, such as disaster recovery, reporting, or backup.

*Q7*: What are *Higher-Order Components* (HOC) in React?

Answer

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,

1. Code reuse, logic and bootstrap abstraction
2. Render High jacking
3. State abstraction and manipulation
4. Props manipulation

## *Q8*: What are advantages of using React Hooks?

Answer

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.

*Q9*: What are the *limitations* of React?

Answer

Below are the list of limitations:

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

*Q10*: What are the differences between a *Class component* and *Functional component*?

Answer

**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

*Q11*: What are the key features of Node.js?

Answer

Let’s look at some of the key features of Node.js.

* **Asynchronous event driven IO helps concurrent request handling –** All APIs of Node.js are asynchronous. This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests. 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. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node.js also become faster.
* **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. Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.
* **Node.js library uses JavaScript –** This is another important aspect of Node.js from the developer’s point of view. 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.

## *Q12*: What do you mean by Asynchronous API?

Answer

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.

## *Q13*: What is Callback Hell and what is the main cause of it?

Answer

Asynchronous JavaScript, or JavaScript that uses callbacks, is hard to get right intuitively. A lot of code ends up looking like this:

fs.readdir(source, function (err, files) {

if (err) {

console.log('Error finding files: ' + err)

} else {

files.forEach(function (filename, fileIndex) {

console.log(filename)

gm(source + filename).size(function (err, values) {

if (err) {

console.log('Error identifying file size: ' + err)

} else {

console.log(filename + ' : ' + values)

aspect = (values.width / values.height)

widths.forEach(function (width, widthIndex) {

height = Math.round(width / aspect)

console.log('resizing ' + filename + 'to ' + height + 'x' + height)

this.resize(width, height).write(dest + 'w' + width + '\_' + filename, function(err) {

if (err) console.log('Error writing file: ' + err)

})

}.bind(this))

}

})

})

}

})

See the pyramid shape and all the }) at the end? This is affectionately known as **callback hell**.

The **cause of callback hell** is when people try to write JavaScript in a way where execution happens visually from top to bottom. Lots of people make this mistake! In other languages like C, Ruby or Python there is the expectation that whatever happens on line 1 will finish before the code on line 2 starts running and so on down the file.

## *Q14*: What is Reconciliation in ReactJS?

Answer

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**.

## *Q15*: What is Sharding in MongoDB?

Answer

**Sharding** is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations. MongoDB supports horizontal scaling through sharding. MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

## *Q16*: What is the difference between returning a callback and just calling a callback?

Answer

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.

function do2(callback) {

log.trace('Execute function: do2');

return callback('do2 callback param');

}

var do2Result = do2((param) => {

log.trace(`print ${param}`);

return `return from callback(${param})`; // we could use that return

});

log.trace(`print ${do2Result}`);

Output:

C:\Work\Node>node --use-strict main.js

[0] Execute function: do2

[0] print do2 callback param

[0] print return from callback(do2 callback param)

*Q17*: When should we *embed one document within another* in MongoDB?

Answer

You should consider embedded documents (subdocuments) for:

* When the relationship is one-to-few (not many, not unlimited). For unlimited use case, you should start considering separating subdocuments into another collection.
* When retrieval is likely to happen together, that will improve performance
* When updates are likely to happen at the same time. Although starting from MongoDB 4.0, you can use multi-documents transactions, a single document transaction would be more performant
* When the field is rarely updated

## *Q18*: Does Mongodb support Foreign Key constraints?

Answer

No.

One of the great things about relational database is that it is really good at keeping the data consistent within the database. One of the ways it does that is by using foreign keys. A foreign key constraint is that let's say there's a table with some column which will have a foreign key column with values from another table's column.

In MongoDB, there's no guarantee that foreign keys will be preserved. It's upto the programmer to make sure that the data is consistent in that manner. Constraints can not be enforced by MongoDB either. It can't even enforce a specific type for a field, due to the schemaless nature of MongoDB.

*Q19*: Explain advantages of *BSON* over *JSON* in MongoDB?

Answer

* **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.

## *Q21*: How can you achieve Transaction in MongoDB?

Answer

In MongoDB, an operation on a single document is **atomic**.

Because you can use embedded documents and arrays to capture relationships between data in a single document structure instead of normalizing across multiple documents and collections, this single-document atomicity obviates the need for multi-document transactions for many practical use cases.

For situations that require atomicity of reads and writes to multiple documents (in single or multiple collections), MongoDB supports **multi-document (distributed) transactions**.

## *Q22*: How does Node.js handle Child Threads?

Answer

Node.js, in its essence, is a single **thread** process. It does not expose **child threads** and **thread** management methods to the developer. **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.

## *Q23*: How does concurrency work in Node.js?

Answer

The thing with node.js is that everything runs concurrently, except for your code.

So, what that means is that there are actually lots of threads running inside Node.js virtual machine (or a thread pool if you wish), and those threads are utilized whenever you call an async function like performing i/o operations on files, accessing databases, requesting urls, etc.

However, for your code, there is only a single thread, and it processes events from an [event queue](http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/). So, when you register a callback its reference is actually passed to the background worker thread, and once the async operation is done, new event is added to the event-queue with that callback

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.

*Q24*: How to avoid *Callback Hell* in Node.js?

Answer

Node.js internally uses a single-threaded event loop to process queued events. But this approach may lead to blocking the entire process if there is a task running longer than expected. Node.js addresses this problem by incorporating callbacks also known as higher-order functions. So whenever a long-running process finishes its execution, it triggers the callback associated. Sometimes, it could lead to complex and unreadable code. More the no. of callbacks, longer the chain of returning callbacks would be.

There are four solutions which can address the callback hell problem:

* **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}

## *Q25*: How to query MongoDB with like?

Answer

I want to query something as SQL's like query:

select \*

from users

where name like '%m%'

How to do the same in MongoDB?

**Answer**

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

// or

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

You're looking for something that contains "m" somewhere (SQL's '%' operator is equivalent to Regexp's '.\*'), not something that has "m" anchored to the beginning of the string.

## *Q26*: Rewrite promise-based Node.js applications to async/await

Problem

Rewrite this code to async/await:

function asyncTask() {

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)

}

}

## *Q27*: What are Pure Components?

Answer

**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.

## *Q28*: What is prop drilling and how can you avoid it?

Answer

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.

*Q29*: What is Key and benefit of using it in lists?

Answer

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>

);

When you don’t have stable IDs for rendered items, you may use the item index as a key as a last resort:

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

<li key={index}>

{todo.text}

</li>

);

**Note:**

1. We don’t recommend using indexes for keys if the order of items may change. This can negatively impact performance and may cause issues with component state
2. If you extract list item as separate component then apply keys on list component instead li tag.

There will be a warning in the console if the key is not present on list items.

*Q30*: What is stream and what are types of streams available in Node.js?

Answer

A stream is an abstract interface for working with streaming data in Node.js.

Streams basically provide two major advantages over using other data handling methods:

* **Memory efficiency**: you don't need to load large amounts of data in memory before you are able to process it
* **Time efficiency**: it takes way less time to start processing data, since you can start processing as soon as you have it, rather than waiting till the whole data payload is available

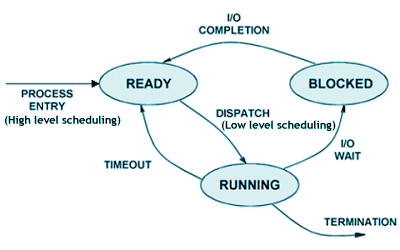
There are 4 types of streams in Node.js:

1. **Writable:** streams to which we can write data. For example, fs.createWriteStream() lets us write data to a file using streams.
2. **Readable:** streams from which data can be read. For example: fs.createReadStream() lets us read the contents of a file.
3. **Duplex:** streams that are both Readable and Writable. For example, net.Socket
4. **Transform:** streams that can modify or transform the data as it is written and read. For example, in the instance of file-compression, you can write compressed data and read decompressed data to and from a file.

## *Q31*: What is a Blocking Code in Node.js?

Answer

A **blocking** call causes results to be returned synchronously.



Performing a blocking system call causes the process to enter the blocked state. Control is let back to the process only after the I/O event that is being waited upon occurs.

const fs = require("fs");

const contents = fs.readFileSync("file.txt", "utf8");

// this line is not reached until the read results are in

console.log(contents);

*Q32*: What is an *Aggregation Pipeline* in MongoDB?

Answer

Aggregation operations process multiple documents and return computed results. You can use aggregation operations to:

* **Group** values from multiple documents together.
* **Perform operations** on the grouped data to return a single result.
* **Analyze** data changes over time.

MongoDB provides aggregation operations through aggregation pipelines — a series of operations that process data documents sequentially. An aggregation pipeline consists of one or more [stages](https://www.mongodb.com/docs/manual/reference/operator/aggregation-pipeline/#std-label-aggregation-pipeline-operator-reference) that process documents:

* Each stage performs an operation on the input documents. For example, a stage can **filter** documents, **group** documents, and **calculate** values.
* The documents that are output from a stage are passed to the next stage.
* An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

Consider:

db.orders.aggregate([

// Stage 1: Filter pizza order documents by pizza size

{

$match: { size: "medium" }

},

// Stage 2: Group remaining documents by pizza name and calculate total quantity

{

$group: { \_id: "$name", totalQuantity: { $sum: "$quantity" } }

}

])

## *Q34*: What's the Event Loop?

Answer

**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.

Every I/O requires a callback - once they are done they are pushed onto the event loop for execution. Since most modern kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed.

## *Q37*: How replication works in MongoDB?

Answer

A replica set consists of a primary node and a secondary node too. With the help of a replica set, all the data from primary node to the secondary node replicates. Replication is a process of synchronizing the data. Replication provides redundancy and it also increases the availability of data with the help of multiple copies of data on the different database server. It also protects the database from the loss of a single server.

*Q38*: How to apply *validation* on props in ReactJS?

Answer

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

1. React.PropTypes.string
2. React.PropTypes.number
3. React.PropTypes.func
4. React.PropTypes.node
5. React.PropTypes.bool

For example, we define propTypes for user component as below,

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

## *Q39*: Is Node.js entirely based on a single-thread?

Answer

Yes, it’s true that Node.js processes all requests on a single thread. But it’s just a part of the theory behind Node.js design. In fact, more than the single thread mechanism, it makes use of events and callbacks to handle a large no. of requests asynchronously.

Moreover, Node.js has an optimized design which utilizes both JavaScript and C++ to guarantee maximum performance. JavaScript executes at the server-side by Google Chrome v8 engine. And the C++ lib UV library takes care of the non-sequential I/O via background workers.

To explain it practically, let’s assume there are 100s of requests lined up in Node.js queue. As per design, the main thread of Node.js event loop will receive all of them and forwards to background workers for execution. Once the workers finish processing requests, the registered callbacks get notified on event loop thread to pass the result back to the user.

## *Q40*: Is it possible to use Class in Node.js?

Answer

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);

}

}

You can export a class just like anything else:

module.exports = class Animal {

};

Once imported into another module, then you can treat it as if it were defined in that file:

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

class Cat extends Animal {

...

}

## *Q41*: Update MongoDB field using value of another field

Answer

Consider SQL command:

UPDATE Person SET Name = FirstName + ' ' + LastName

In Mongo, is it possible to update the value of a field using the value from another field?

**Answer**

You cannot refer to the document itself in an update (yet). You'll need to iterate through the documents and update each document using a function like:

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

function (elem) {

db.person.update(

{

\_id: elem.\_id

},

{

$set: {

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

}

}

);

}

);

## *Q42*: What is the purpose of super(props)?

Answer

When you pass props to super, the props get assigned to this. Take a look at the following scenario:

constructor(props) {

super();

console.log(this.props) //undefined

}

How ever when you do :

constructor(props) {

super(props);

console.log(this.props) //props will get logged.

}

Note that passing or not passing props to super has **no effect** on later uses of this.props outside constructor. That is render, shouldComponentUpdate, or event handlers **always** have access to it.

*Q43*: When not to use Node.js?

Answer

We can use Node.js for a variety of applications. But it is a single threaded framework, so we **should not use** it for cases

* **where the application requires long processing time (If the server is doing some calculation)**,

it won’t be able to process any other requests. Hence, Node.js is best when processing needs less dedicated CPU time.

*Q44*: When to use Redis or MongoDB?

Answer

* **Use MongoDB if you don't know yet how you're going to query your data or what schema to stick with.** MongoDB is suited for Hackathons, startups or every time you don't know how you'll query the data you inserted. MongoDB does not make any assumptions on your underlying schema. While MongoDB is schemaless and non-relational, this does not mean that there is no schema at all. It simply means that your schema needs to be defined in your app (e.g. using Mongoose). Besides that, MongoDB is great for prototyping or trying things out. Its performance is not that great and can't be compared to Redis.
* **Use Redis in order to speed up your existing application.** It is very uncommon to use Redis as a standalone database system (some people prefer referring to it as a "key-value"-store).

## *Q45*: Why is a Covered Query important?

Answer

A covered query is a query that can be satisfied entirely using an index and does not have to examine any documents. Because the index contains all fields required by the query, MongoDB can both match the query conditions and return the results using only the index.

Querying only the index can be much **faster** than querying documents outside of the index. Index keys are typically smaller than the documents they catalog, and indexes are typically available in RAM or located sequentially on disk.

### ***Q46*: What do you understand by Virtual DOM? Explain it's working.**

A virtual DOM is a lightweight JavaScript object which originally is just a copy of the real DOM. It is a node tree that lists the elements, their attributes, and content as objects and their properties. React’s render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which are caused by various actions done by the user or by the system.

* Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
* Then the difference between the previous DOM representation and the new one is calculated.
* Once the calculations are done, the real DOM will be updated with only the things that have actually changed.

### ***Q47*: What are the differences between a class component and a functional component?**

**Class Components**

* Class-based components use 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 compared 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 rendered functions in the class component.
* Functional Components can have state and mimic lifecycle events using Reach Hooks.

### ***Q48*: What are the key features of Node.js?**

* **Asynchronous event-driven IO helps concurrent request handling –** All APIs of Node.js are asynchronous. This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests. 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. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence the processing of requests within Node.js also becomes faster.
* **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. Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.
* **Node.js library uses JavaScript –** This is another important aspect of Node.js from the developer’s point of view. The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.

### ***Q49*: Explain the advantages of BSON over JSON in MongoDB?**

* 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 the length of strings and sub-objects. 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 the 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.

### ***Q50*: If Node.js is single-threaded then how does it handle concurrency?**

Node provides a single thread to programmers so that code can be written easily and without a bottleneck. Node internally uses multiple POSIX threads for various I/O operations such as File, DNS, Network calls, etc.  
  
When Node gets an 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, the **event loop** runs and checks the queue and if the execution stack of Node is empty then it adds the queue result to the execution stack. This is how Node manages concurrency.

### ***Q51*: What is JSX in React.JS?**

JSX is a syntax extension to JavaScript and comes with the full power of JavaScript. JSX produces React elements. You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions.

### ***Q52*: 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.

### ***Q53*: What is the purpose of MongoDB?**

MongoDB is a document-oriented database manager designed for storing high volumes of data. It saves data in binary JSON format and implements the concept of collection and documentation. MongoDB is a cross-platform, NoSQL database equipped with high performance, high scalability, and flexibility that allow for seamless querying and indexing.

### ***Q54*: What is the purpose of ExpressJS?**

ExpressJS is a web application framework designed to support and host Node.js projects. It is an open-source framework available under the MIT license. ExpressJS manages the workflow between the front-end and the database and facilitates a smooth and secure transfer of data. It boasts of excellent error handling and web design functionality to optimize the web development process.

### ***Q55*: Define Data Modelling?**

Data modelling is a term used in the context of Mongoose and MongoDB. As the name suggests, data modelling refers to the process of creating a data model for the data at hand so it can be stored in a database. A data model is a conceptual representation of data objects, the relation among different data objects, and the rules that define those relations.

* It helps to represent the data visually while also enforcing business rules, regulatory compliances, and government policies on the data.
* It is implemented to ensure consistency in naming conventions, default values, semantics, security, and quality of the data.

Bottom of Form