Javascript

◦ DOM
◦ CDN vs Download
◦ Event Listener
◦ Closure
◦ Scope and types of scope
◦ Hoisting
◦ Array methods
◦ Array cloning
◦ Spread operator
◦ Map, for each and reduce()
◦ Filter
◦ Callback
◦ String methods
◦ Callback hell
◦ Currying
◦ Memoization
◦ Promise , promise methods
◦ Arrow function
◦ Types of conversion - int, parse
◦ Conditional statement
◦ Prototype
◦ Instance()
◦ Class , object, object methods
◦ Object destructuring
◦ Variable typing
◦ Event Loop
◦ Call stack
◦ Global Execution context
◦ Rest
◦ Spread
◦ Call
◦ Apply
◦ Bind
◦ Shadowing
◦ This
◦ IIFE
◦ Type casting
◦ Expression and statement
◦ Bitwise operators
◦ High order functions
◦ Undefined and null
◦ Do-while loop
◦ Continue
◦ Event Loop
◦ Pure Function
◦ Generator Function

Node, Express

◦ Patch & put
◦ Option Method
◦ Middlewares
◦ Callback
◦ Set-immediate
◦ Error middleware
◦ fs - modules
◦ Events
◦ HTTP modules - createServer()
◦ package.js dependencies
◦ Routing
◦ Query Parameters
◦ Session and cookie
◦ View engine
◦ Hbs
◦ Get post difference
◦ 400 error
◦ 500 error
◦ Cors in detail
◦ Body parser
◦ Applied file , write file
◦ Global execution context
◦ package.js
◦ Morgan
◦ NPM
◦ PM2
◦ Postman
◦ Error handling in express
◦ Types of errors
◦ app.use
◦ MVC
◦ TRACE
◦ Child process
◦ Process and thread
◦ Buffer and stream
◦ Middlewares
◦ res.end
◦ Static folder
◦ HTTP vs HTTPS
◦ CORS
◦ REPL
◦ Modules
◦ Express
◦ Request and Response
◦ Authentication and Authorization
◦ Middleware Libraries

MongoDB

◦ Collection
◦ CRUD
◦ Lookup
◦ Unwind
◦ In was all
◦ Capped collection
◦ Alias
◦ Mongo Utilities
◦ Mongo Dump
◦ Indexing
◦ Logical operator
◦ Comparison operator
◦ Namespace
◦ addtoSet()
◦ Sharing
◦ Insert vs save
◦ BSON
◦ Mongo export
◦ Data type
◦ Mongo Server
◦ Update operators - set unset inc
◦ Distinct - unique
◦ Accumulators
◦ Count
◦ Out
◦ Express
◦ Find one and update
◦ Indexing
◦ Replication
◦ Aggregation
◦ Normalisation
◦ Aggregation Pipeline
◦ Replica set
◦ Different types of indexing
◦ Backup and restore
◦ Logical and comparison operators
◦ Upsert
◦ Sharding vs Replica set
◦ Validation
◦ Data API
◦ Charts
◦ NodeJS Driver
◦ Searching
◦ Group
◦ Match
◦ Project

PostgreSQL

◦ Database Concepts
◦ SQL Syntax
◦ Data Definition Language (DDL)
◦ Data Manipulation Language (DML)
◦ Querying Data
◦ Joins and Relationships
◦ Indexes and Performance Tuning
◦ Data Constraints
◦ Views
◦ Stored Procedures and Functions
◦ Transactions and Concurrency
◦ Subqueries and Derived Tables
◦ Normalisation
◦ Data Aggregation
◦ Views and Materialised Views
◦ Security and Access Control
◦ Database Administration
◦ Trigger
◦ Pros and Cons of indexing
◦ UNION
◦ ACID properties
◦ Enum data type
◦ Aliases
◦ DISTINCT
◦ EXISTS
◦ GLOBAL TEMP
◦ LOCAL TEMP
◦ Primary keys
◦ Indexes
◦ INTERSECT
◦ MINUS
◦ EXCEPT
◦ SELECT TOP
◦ HAVING
◦ FETCH Data
◦ IN
◦ CROSS JOIN
◦ ANY
◦ ALL
◦ CASE
◦ INNER JOIN
◦ LEFT JOIN
◦ RIGHT JOIN
◦ LIMIT
◦ DROP
◦ ALTER

React , Redux

◦ React Components
◦ JSX
◦ State and Props
◦ React Hooks
◦ Event Handling
◦ Conditional Rendering
◦ Lists and Keys
◦ Forms and Form Handling
◦ Component Communication
◦ React Router
◦ Styling in React
◦ React Testing
◦ State Management
◦ React Performance Optimization
◦ Error Handling
◦ Server-Side Rendering (SSR)
◦ React Hooks and Custom Hooks
◦ Component lifecycle methods
◦ Lazy loading
◦ High order components
◦ Conditional rendering
◦ Proptypes
◦ Usecallback and usememo
◦ Hash router
◦ Link and switch
◦ Suspense
◦ Profiler
◦ Render Props
◦ Interceptor
◦ Custom hook to call
◦ Bind
◦ ContextAPI
◦ React Fibre
◦ Shadow DOM
◦ REST API
◦ Diffing Algorithm
◦ Pure Components
◦ Limitations of React
◦ Redux
◦ Action
◦ Dispatch
◦ Store
◦ Reducer
◦ UI Binding
◦ Portals
◦ Class Components

<u>Data Structure</u>

◦ Arrays
◦ Linked Lists
◦ Stacks
◦ Queues
◦ Trees
◦ Heaps
◦ Graphs
◦ Hash Tables
◦ Tries
◦ Graph Algorithms
◦ Sorting Algorithms
◦ Searching Algorithms
◦ Big O Notation
◦ Static and memory allocation
◦ Pros and cons of Recursion
◦ Jagged array
◦ Linear and nonlinear data structures
◦ Applications
◦ Divide and conquer strategy
◦ Underflow in stack
◦ Hash functions
◦ Spanning tree and minimum spanning tree
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦
◦

Javascript

DOM: The Document Object Model (DOM) is a programming interface for web documents. It represents the structure of an HTML or XML document as a tree-like model, allowing JavaScript to interact with and manipulate the content, structure, and style of web pages.

CDN vs Download: CDN stands for Content Delivery Network. It refers to a network of servers distributed globally that deliver web content to users based on their geographic location, resulting in faster and more reliable content delivery. Download refers to the process of retrieving a file or resource from a server to a local machine.

Event Listener: An event listener is a function that listens for a specific event to occur and executes a callback function in response. It allows JavaScript code to respond to user actions (such as clicks, key presses, or form submissions) or other events happening in the browser.

Closure: Closure is a feature in JavaScript that allows a function to retain access to variables from its outer (enclosing) lexical scope even after the outer function has finished executing. It enables powerful programming patterns like encapsulation, data privacy, and the creation of higher-order functions.

Scope and Types of Scope: Scope refers to the visibility and accessibility of variables, functions, and objects in a particular part of the code. JavaScript has global scope, function scope, and block scope (introduced in ES6 with the let and const keywords).

Hoisting: Hoisting is a JavaScript behavior where variable and function declarations are moved to the top of their containing scope during the compilation phase. This means that you can use variables and functions before they are declared in the code.

Array Methods: JavaScript provides several built-in methods for manipulating arrays, such as push, pop, shift, unshift, splice, slice, concat, join, map, forEach, reduce, filter, and many more. These methods make it easier to perform common operations on arrays.

Array Cloning: Array cloning refers to creating a copy of an existing array. There are several methods to clone an array in JavaScript, such as using the spread operator ([...array]), Array.from(), or slice() method.

Spread Operator: The spread operator (...) is used to expand elements of an array or object into places where multiple elements or key-value pairs are expected. It allows for easily combining arrays, creating shallow copies, and passing multiple arguments to functions.

Map, forEach, and Reduce: These are higher-order array methods in JavaScript. map is used to transform each element of an array and return a new array. forEach iterates over each element of an array and performs a specified action. reduce is used to reduce an array to a single value by applying a function to each element.

Filter: The filter method is used to create a new array containing elements from the original array that pass a certain condition defined by a callback function. It helps in filtering out specific elements based on certain criteria.

Callback: A callback is a function passed as an argument to another function. It allows asynchronous or event-driven programming by specifying what should happen once an operation is completed or an event occurs.

String Methods: JavaScript provides a variety of built-in methods for working with strings, such as toUpperCase, toLowerCase, charAt, split, substring, indexOf, replace, trim, and many more. These methods enable manipulation and transformation of string data.

Callback Hell: Callback hell refers to a situation where multiple nested callbacks are used, leading to code that is hard to read, understand, and maintain. It often occurs when dealing with asynchronous operations without proper handling techniques.

Currying: Currying is a technique in functional programming where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument. It allows for partial application and the creation of specialized functions.

Memoization: Memoization is an optimization technique that involves caching the result of a function based on its input parameters. It helps avoid unnecessary recomputation of expensive operations and improves performance in certain scenarios.

Promise and Promise Methods: Promises are used for handling asynchronous operations in JavaScript. They represent a value that may be available now, in the future, or never. Promises have methods like then, catch, finally, and static methods like Promise.all, Promise.race, etc., for managing asynchronous flows.

Arrow Function: Arrow functions are a concise syntax for defining functions in JavaScript. They have a shorter syntax and inherit the this value from the surrounding context.

Types of Conversion - int, parse: JavaScript provides methods like parseInt and parseFloat for converting strings to numbers. Implicit conversion can also occur when performing mathematical operations on variables.

Conditional Statement: Conditional statements (if-else, switch) are used to control the flowof execution in JavaScript based on certain conditions. They allow for different code paths to be executed depending on whether the conditions are true or false.

Prototype: In JavaScript, objects have a prototype property that allows them to inherit properties and methods from another object. Prototypes are used to achieve inheritance and object reusability.

Instance(): The instanceof operator in JavaScript is used to check if an object is an instance of a particular class or constructor function.

Class, Object, Object Methods: JavaScript introduced the class syntax in ECMAScript 2015 (ES6) to create reusable object blueprints. Objects are instances of classes and can have properties and methods associated with them.

Object Destructuring: Object destructuring is a syntax in JavaScript that allows extracting individual properties from objects and assigning them to variables in a concise way.

Variable Typing: JavaScript is a dynamically typed language, meaning that variables can hold values of any type, and their types can change during runtime.

Event Loop: The event loop is a mechanism in JavaScript that handles the execution of multiple tasks concurrently. It ensures that asynchronous operations, such as callbacks and promises, are executed in the appropriate order.

Call Stack: The call stack is a data structure used by JavaScript to keep track of function calls and their execution contexts. It follows the Last-In-First-Out (LIFO) principle.

Global Execution Context: The global execution context represents the environment in which the JavaScript code is executed initially. It includes the global object (window in browsers, global in Node.js) and global variables and functions.

Rest: The rest parameter syntax (...) in JavaScript allows representing an indefinite number of arguments as an array in function declarations and function calls.

Spread: The spread operator (...) can be used to expand an array into individual elements or merge multiple arrays or objects into a single array or object.

Call, Apply, Bind: call, apply, and bind are methods in JavaScript used to manipulate the this value and invoke functions with a specific context. They are commonly used in function borrowing, function currying, and event handling.

Shadowing: Shadowing occurs when a variable declared in an inner scope has the same name as a variable declared in an outer scope, causing the inner variable to take precedence.

This: this refers to the context in which a function is executed. Its value depends on how the function is invoked and can be affected by the use of arrow functions, explicit binding, or context manipulation.

IIFE: Immediately Invoked Function Expression (IIFE) is a JavaScript design pattern that involves wrapping a function in parentheses and immediately invoking it. It helps create a private scope and avoid polluting the global namespace.

Type Casting: Type casting involves converting the type of a value to another type. JavaScript provides explicit and implicit type casting mechanisms.

Expression and Statement: Expressions are pieces of code that evaluate to a value, while statements perform actions and do not produce a value.

Bitwise Operators: JavaScript provides bitwise operators (&, |, ^, ~, <<, >>, >>>) for performing bitwise operations on binary representations of numbers.

High-Order Functions: High-order functions are functions that can take other functions as arguments or return functions as results. They enable functional programming paradigms and code reuse.

Undefined and Null: undefined and null are special values in JavaScript. undefined represents the absence of a value, while null represents the intentional absence of a value.

Do-While Loop: The do-while loop is a control flow statement that executes a block of code at least once and then continues as long as a specified condition is true.

Continue: The continue statement is used in loops to skip the rest of the current iteration and continue to the next iteration.

Pure Function: A pure function is a function that always returns the same output for the same input and has no side effects. It relies only on its input parameters and does not modify external state.

Generator Function: Generator functions are special types of functions that can be paused and resumed during execution. They produce iterators that can generate a sequence of values.

NodeJS , Express

Patch & Put: PATCH and PUT are HTTP methods used for updating resources on the server. In Express, you can handle PATCH and PUT requests using route handlers and middleware functions.

Option Method: The OPTIONS method is an HTTP method used to retrieve the allowed methods and headers for a resource. In Express, you can handle OPTIONS requests using the app.options() method or by creating a route handler for the OPTIONS HTTP verb.

Middlewares: Middlewares are functions in Express that have access to the request and response objects and can perform tasks such as logging, authentication, request parsing, error handling, etc. They can be applied globally using app.use() or to specific routes.

Callback: Callbacks are functions passed as arguments to other functions and are commonly used in asynchronous programming in Node.js. They are invoked once an asynchronous operation is completed, allowing you to handle the result or error.

Set-immediate: setImmediate() is a Node.js global function used to schedule a function to be executed in the next iteration of the event loop, after I/O events. It allows you to defer the execution of code without blocking the event loop.

Error Middleware: Error middleware in Express is used to handle errors that occur during the request-response cycle. It is typically defined as the last middleware in the chain and can capture and process any unhandled errors.

fs - Modules: The fs module in Node.js provides functions for interacting with the file system, such as reading and writing files, creating directories, and manipulating file metadata.

Events: The events module in Node.js allows you to create and handle custom events. It provides an event emitter object that emits events, and listeners can be registered to perform actions when specific events occur.

HTTP modules - createServer(): The http module in Node.js provides functionality to create HTTP servers and make HTTP requests. The createServer() method is used to create an HTTP server that listens for incoming requests and allows you to define request handlers.

package.json Dependencies: The package.json file is used to manage Node.js project dependencies and metadata. It includes a dependencies field where you specify the required dependencies for your project, and you can use tools like npm or Yarn to install them.

Routing: Routing in Express defines how the server responds to different URL paths and HTTP methods. It allows you to map routes to specific request handlers and perform actions based on the requested URL.

Query Parameters: Query parameters are key-value pairs appended to the URL of an HTTP request. In Express, you can access query parameters using the req.query object, allowing you to extract and utilize the values in your application logic.

Session and Cookie: Sessions and cookies are used for maintaining state and identifying users in web applications. In Express, you can use middleware libraries like express-session and cookie-parser to handle session management and cookie parsing.

View Engine - Hbs: Hbs (Handlebars) is a popular view engine for Express that allows you to render dynamic HTML templates. It provides a way to inject data into templates and generate HTML output on the server.

Get Post Difference: GET and POST are HTTP methods used for different purposes. GET is used for retrieving data, while POST is used for submitting data to the server. In Express, you can handle GET and POST requests using separate route handlers or the same route with different methods.

400 Error: The 400 status code in HTTP refers to a "Bad Request" error, indicating that the server cannot understand or process the client's request due to malformed syntax or invalid parameters.

500 Error: The 500 status code in HTTP refers to an "Internal Server Error" error, indicating that an unexpected error occurred on the server while processing the request. It indicates a server-side problem rather than a client-side issue.

CORS in Detail: CORS (Cross-Origin Resource Sharing) is a mechanism that allows resources on a web page to be requested from another domain. It involves handling HTTP headers and server responses to control cross-origin requests.

Body Parser: The body-parser middleware in Express is used to parse the request body and make it available in the req.body object. It supports various formats like JSON, URL-encoded, and multipart form data.

File Handling - Read/Write: In Node.js, you can handle file read and write operations using the fs module. You can read files asynchronously or synchronously and write data to files using different methods provided by the module.

Global Execution Context: The global execution context in Node.js represents the environment in which the Node.js application runs. It includes global objects and functions that are accessible from any module or script.

Morgan: Morgan is a popular logging middleware for Express that logs HTTP requests and responses. It provides various logging formats and can be customized to fit specific logging needs.

NPM: NPM (Node Package Manager) is a package manager for Node.js. It allows you to install, manage, and publish packages and dependencies for your Node.js projects.

PM2: PM2 is aproduction process manager for Node.js applications. It allows you to manage and monitor your Node.js applications, including features like automatic restarts, load balancing, and clustering.

Postman: Postman is a popular API development and testing tool. It provides a user-friendly interface for making HTTP requests, inspecting responses, and testing API endpoints.

Error Handling in Express: Error handling in Express involves capturing and handling errors that occur during the request-response cycle. It can be done using custom error middleware or by utilizing the built-in error handling capabilities of Express.

Types of Errors: There are different types of errors in Node.js and Express, including built-in JavaScript errors (e.g., SyntaxError, TypeError), system-level errors (e.g., network errors), and application-specific errors (e.g., validation errors, custom errors).

app.use: The app.use() method in Express is used to mount middleware functions on the application's request processing pipeline. It can be used to apply middleware globally or to specific routes.

MVC: MVC (Model-View-Controller) is an architectural pattern commonly used in web applications. In the context of Express, it refers to a way of structuring and organizing code by separating concerns into models (data), views (presentation), and controllers (logic).

TRACE: TRACE is an HTTP method used for testing and debugging. It echoes back the received request to the client, allowing it to see what changes have been made by intermediate servers.

Child Process: The child_process module in Node.js allows you to create child processes, which are separate instances of the Node.js runtime. It enables you to run external commands, execute scripts, and communicate with other processes.

Process and Thread: In Node.js, the process represents the running instance of a Node.js application, while a thread is a unit of execution within a process. Node.js follows a single-threaded event loop model, but it can utilize multiple threads internally for I/O operations.

Buffer and Stream: Buffers and streams are used for handling binary data and data streams in Node.js. Buffers are used to manipulate binary data directly, while streams provide an abstraction for reading from or writing to data sources or destinations in a streaming manner.

res.end: The res.end() method in Express is used to end the response process and send the response back to the client. It allows you to provide a final data chunk or signal the end of the response without sending any data.

Static Folder: In Express, a static folder is a directory containing static files (e.g., HTML, CSS, images) that are served directly by Express without any additional processing. It can be defined using the express.static() middleware.

HTTP vs HTTPS: HTTP (Hypertext Transfer Protocol) is a protocol used for transmitting data over the internet, while HTTPS (HTTP Secure) is a secure version of HTTP that uses SSL/TLS encryption for secure communication.

CORS: CORS (Cross-Origin Resource Sharing) is a mechanism that allows web servers to specify which origins are allowed to access their resources. It involves handling HTTP headers to control cross-origin requests.

REPL: REPL (Read-Eval-Print Loop) is an interactive programming environment that allows you to execute code and get immediate results. Node.js provides a REPL environment that can be accessed through the command line.

Modules: Modules in Node.js are reusable blocks of code that encapsulate functionality and promote code organization and reusability. Node.js uses the CommonJS module system, allowing you to create and import modules using require() and module.exports.

Express: Express is a popular web application framework for Node.js. It provides a set of features and utilities for building web applications, including routing, middleware handling, template rendering, and HTTP request/response management.

Request and Response: In Express, the req object represents the HTTP request received from the client, while the res object represents the HTTP response that will be sent back to the client. They provide methods and properties to handle and manipulate the request and response data.

Authentication and Authorization: Authentication is the process of verifying the identity of a user, while authorization involves granting or denying access to certain resources or actions. Express provides middleware and libraries to handle authentication and authorization tasks.

Middleware Libraries: Middleware libraries in Express provide additional functionality and features to your application by extending the request processing pipeline. Examples of middleware libraries include body-parser, cookie-parser, passport, helmet, and many more.

<u>MongoDB</u>

Collection: In MongoDB, a collection is a grouping of MongoDB documents. It is equivalent to a table in relational databases and stores related documents together.

CRUD: CRUD stands for Create, Read, Update, and Delete. These are the basic operations that can be performed on MongoDB documents. It includes inserting new documents, querying and retrieving documents, updating existing documents, and deleting documents.

Lookup: The $lookup aggregation stage in MongoDB allows you to perform a left outer join between two collections. It retrieves matching documents from a secondary collection and adds them to the output documents.

Unwind: The $unwind aggregation stage in MongoDB is used to deconstruct an array field from the input documents and output one document for each element in the array. It enables further analysis and aggregation on the individual array elements.

In, Was, All: These are operators used in MongoDB queries. $in is used to match any of the specified values in an array. $was is a deprecated operator and was used for time-travel queries. $all is used to match documents that contain all the specified values in an array.

Capped Collection: A capped collection in MongoDB is a fixed-size collection that has a maximum number of documents or a maximum storage size. When the collection reaches its maximum size, older documents are automatically removed to accommodate new documents.

Alias: In MongoDB aggregation queries, an alias is an alternative name given to a field in the output document. It is used to rename or alias a field to a different name in the result.

Mongo Utilities: MongoDB provides a set of utilities and tools for managing and working with MongoDB databases, such as the mongo shell, mongodump, mongorestore, mongoimport, and mongoexport utilities.

Mongo Dump: mongodump is a MongoDB utility used to create a binary export of the data in a MongoDB database. It can be used to backup the data or transfer it to another MongoDB instance.

Indexing: Indexing in MongoDB improves query performance by creating indexes on fields. It allows faster data retrieval and can speed up query execution by reducing the number of documents that need to be scanned.

Logical Operator: MongoDB provides logical operators ($and, $or, $nor, $not) to combine multiple conditions in a query and perform logical operations such as AND, OR, NOR, and NOT.

Comparison Operator: MongoDB provides comparison operators ($eq, $ne, $gt, $lt, $gte, $lte, $in, $nin, $exists, $type) to compare values and perform conditional operations in query conditions.

Namespace: In MongoDB, a namespace refers to the combination of a database name and a collection name. It uniquely identifies a collection within a database.

addToSet(): The $addToSet operator in MongoDB is used to add elements to an array field only if the elements are not already present in the array. It helps maintain a unique array of values.

Sharing: Sharing in MongoDB refers to the distribution of data across multiple shards in a sharded cluster. It allows for horizontal scaling and improved performance by distributing the workload across multiple servers.

Insert vs Save: In MongoDB, the insert method is used to insert a new document into a collection, while the save method is used to insert a new document or update an existing document based on the _id field.

BSON: BSON (Binary JSON) is a binary representation of JSON-like documents used in MongoDB. It is a binary-encoded format that supports additional data types and is optimized for storage and performance.

Mongo Export: mongoexport is a MongoDB utility used to export data from a MongoDB collection to a JSON, CSV, or TSV file. It allows you to extract data from MongoDB for backup, analysis, or transfer purposes.

Data Type: MongoDB supports various data types, including string, number, boolean, date, array, object, null, and others. Understanding data types is important for schema design and query operations.

Mongo Server: The MongoDB server is the core component of MongoDB that manages database operations and handles client requests. It listens for incoming connections and manages data storage and retrieval.

Update Operators - set, unset, inc: MongoDB provides update operators to modify specific fields within a document. $set is used to update specific fields with new values, $unset is used to remove specific fields, and $inc is used to increment or decrement numeric field values.

Distinct - Unique: The distinct operation in MongoDB returns an array of unique values for a specific field in a collection. It is useful for obtaining unique values in a field without duplicates.

Accumulators: Accumulators are operators used in MongoDB's aggregation framework. They perform calculations on a group of documents and produce a single result. Examples include $sum, $avg, $min, $max, and $push.

Count: The count method in MongoDB is used to count the number of documents that match a specific query condition in a collection.

Out: The $out aggregation stage in MongoDB is used to write the results of an aggregation pipeline to a specified collection. It allows you to store the output of an aggregation pipeline as a new collection.

Express: Express is a popular web application framework for Node.js that provides a set of features and utilities for building web applications, including routing, middleware handling, template rendering, and HTTP request/response management.

Find One and Update: ThefindOneAndUpdate is a method in MongoDB used to find a single document in a collection that matches a specified query condition, update it with new values, and return the updated document. It provides a way to atomically find and update a document in a single operation.

Indexing: Indexing in MongoDB improves query performance by creating indexes on fields. It allows faster data retrieval and can speed up query execution by reducing the number of documents that need to be scanned.

Replication: Replication in MongoDB is the process of synchronizing data across multiple servers to provide redundancy, fault tolerance, and high availability. It involves maintaining multiple copies of data in a replica set, which consists of primary and secondary nodes.

Aggregation: Aggregation in MongoDB is a framework for processing and analyzing data in the database. It allows you to perform complex data transformations, grouping, filtering, and computations using a pipeline of stages.

Normalization: Normalization is a process in database design that involves organizing data into multiple related collections and minimizing data redundancy. It helps maintain data consistency and improves data integrity.

Aggregation Pipeline: The aggregation pipeline in MongoDB allows you to process data using a sequence of stages. Each stage performs a specific operation on the input documents and passes the results to the next stage.

Replica Set: A replica set in MongoDB is a group of MongoDB servers that host the same data. It provides redundancy and automatic failover in case the primary node becomes unavailable.

Different Types of Indexing: MongoDB supports different types of indexes, including single-field indexes, compound indexes (multiple fields), multi-key indexes (for arrays), text indexes (for text search), geospatial indexes (for location-based queries), and more.

Backup and Restore: Backup and restore operations in MongoDB involve creating copies of database data and restoring them to recover from data loss or system failures. MongoDB provides utilities like mongodump and mongorestore for performing backup and restore operations.

Logical and Comparison Operators: Logical operators in MongoDB, such as $and, $or, and $not, are used to combine multiple conditions in a query. Comparison operators, such as $eq, $ne, $gt, $lt, are used to compare field values.

Upsert: Upsert is a combination of insert and update operations in MongoDB. If a document matching the query condition is found, it will be updated. If not, a new document will be inserted.

Sharding vs Replica Set: Sharding and replica sets are different mechanisms for scaling and ensuring high availability in MongoDB. Sharding involves distributing data across multiple servers (shards) based on a shard key, while replica sets provide data redundancy and automatic failover.

Validation: MongoDB allows you to define validation rules on collections to enforce data integrity and consistency. You can specify validation constraints on fields, such as data types, value ranges, and required fields.

Data API: The MongoDB Data API is a feature that allows you to interact with MongoDB databases using a RESTful HTTP interface. It provides a way to access and manipulate MongoDB data using standard HTTP methods.

Charts: MongoDB Charts is a data visualization tool provided by MongoDB. It allows you to create visual representations of MongoDB data and generate charts, graphs, and dashboards without writing code.

NodeJS Driver: The MongoDB Node.js driver is a library that provides a programming interface for connecting to MongoDB from Node.js applications. It allows you to perform database operations, execute queries, and interact with MongoDB using JavaScript.

Searching: Searching in MongoDB involves querying documents based on specific criteria. MongoDB supports various query operators, including comparison operators, logical operators, text search, geospatial search, and more.

Group: The $group aggregation stage in MongoDB is used to group documents by a specified field and perform aggregate calculations on grouped data, such as counting, summing, averaging, and more.

Match: The $match aggregation stage in MongoDB is used to filter and select documents that match specified criteria in an aggregation pipeline. It allows you to narrow down the set of documents to be processed in subsequent stages.

Project: The $project aggregation stage in MongoDB is used to reshape or transform the documents in an aggregation pipeline. It allows you to include or exclude fields, create new computed fields, and modify the structure of the output documents.

PostgreSQL

Database Concepts: Database concepts refer to the fundamental principles and components of databases, such as tables, columns, rows, relationships, normalization, indexing, and data integrity.

SQL Syntax: SQL (Structured Query Language) is the language used to communicate with databases. Understanding SQL syntax is essential for performing various operations, such as querying, inserting, updating, and deleting data in PostgreSQL.

Data Definition Language (DDL): DDL consists of SQL statements used to define and manage the structure of database objects. It includes creating, altering, and dropping tables, views, indexes, constraints, and other database entities.

Data Manipulation Language (DML): DML consists of SQL statements used to manipulate and retrieve data within a database. It includes inserting, updating, deleting, and selecting data from tables.

Querying Data: Querying data involves retrieving specific information from a database using SELECT statements. It includes filtering, sorting, joining, and aggregating data to meet specific criteria.

Joins and Relationships: Joins are used to combine rows from multiple tables based on related columns. Understanding different types of joins (e.g., inner join, left join, right join) and establishing relationships between tables is crucial for retrieving data from related tables.

Indexes and Performance Tuning: Indexes are database structures used to improve query performance by allowing faster data retrieval. Understanding how to create, use, and optimize indexes can significantly enhance query performance.

Data Constraints: Data constraints are rules applied to columns in a table to enforce data integrity. Common constraints include primary key, foreign key, unique, not null, and check constraints.

Views: Views are virtual tables created from the result of a query. They allow you to encapsulate complex queries and provide a simplified and reusable way to access data.

Stored Procedures and Functions: Stored procedures and functions are precompiled database routines that can be executed with parameters. They encapsulate logic and provide reusability and modularity within the database.

Transactions and Concurrency: Transactions ensure the consistency and integrity of data by grouping multiple database operations into a single logical unit. Understanding how to handle transactions and manage concurrency (e.g., using locking mechanisms) is essential for maintaining data consistency.

Subqueries and Derived Tables: Subqueries are queries nested within other queries, allowing you to retrieve data based on intermediate results. Derived tables are temporary result sets generated from subqueries. Both techniques enhance the flexibility and power of querying data.

Normalization: Normalization is the process of organizing data in a database to eliminate redundancy and improve data integrity. It involves dividing data into multiple related tables and establishing relationships between them.

Data Aggregation: Data aggregation involves combining and summarizing data to derive meaningful insights. It includes functions like COUNT, SUM, AVG, MAX, and MIN to perform calculations on groups of data.

Materialized Views: Materialized views are precomputed views that store the result of a query as a physical table. They can significantly improve query performance for complex or frequently used queries.

Security and Access Control: Security and access control in PostgreSQL involve managing user accounts, roles, and permissions to ensure data confidentiality and integrity. It includes granting or revoking privileges on database objects.

Database Administration: Database administration involves tasks like database installation, configuration, monitoring, backup and recovery, performance tuning, and ensuring database availability and reliability.

Trigger: A trigger is a database object that automatically executes a specified action when a specific event occurs (e.g., before or after an INSERT, UPDATE, or DELETE operation).

Pros and Cons of Indexing: Indexing improves query performance by allowing faster data retrieval. However, indexes consume disk space and may slightly slow down data modification operations (INSERT, UPDATE, DELETE). Understanding the trade-offs is crucial for effective indexing.

UNION: UNION is an SQL operator used to combine the results of two or more SELECT statements into a single result set. It removes duplicate rows from the result set.

ACID Properties: ACID stands for Atomicity, Consistency, Isolation, and Durability, which are the properties that ensure reliable and transactionally consistent database operations.

Enum Data Type: The enum data type in PostgreSQL allows you to define a custom enumeration with a set of predefined values. It provides a way to enforce data integrity and restrict the allowed values for a column.

Aliases: Aliases are used to provide alternate names for database objects (tables, columns), expressions, or query results. They make query results more readable and can be used for renaming objects temporarily.

DISTINCT: The DISTINCT keyword is used in SELECT statements to retrieve unique values from a column. It eliminates duplicate values from the result set.

EXISTS: The EXISTS keyword is used in a subquery to check the existence of rows that satisfy a specified condition. It returns true if the subquery returns any rows; otherwise, it returns false.

26.GLOBAL TEMP and LOCAL TEMP: GLOBAL TEMP and LOCAL TEMP are options used when creating temporary tables in PostgreSQL. GLOBAL TEMP tables are visible to all sessions and are dropped at the end of the session or when explicitly deleted. LOCAL TEMP tables are visible only to the session that created them and are dropped automatically at the end of the session.

Primary Keys: A primary key is a column or a combination of columns that uniquely identifies each row in a table. It ensures data integrity and provides a way to reference individual rows from other tables.

Indexes: Indexes in PostgreSQL are database structures used to speed up data retrieval by creating an optimized lookup mechanism. They are created on specific columns and improve query performance by allowing faster data access.

INTERSECT, MINUS, EXCEPT: INTERSECT, MINUS, and EXCEPT are set operators used in SQL queries. INTERSECT returns the common rows from two or more result sets, MINUS returns the rows that exist in the first result set but not in the second, and EXCEPT returns the rows that exist in the first result set but not in the second, removing any duplicates.

SELECT TOP: The SELECT TOP clause is used to retrieve a specified number of rows from the top of a result set. It is commonly used in PostgreSQL to limit the number of rows returned by a query.

HAVING: The HAVING clause is used in combination with the GROUP BY clause to filter the result set based on aggregated values. It allows you to apply conditions to groups created by the GROUP BY clause.

FETCH Data: The FETCH clause is used to retrieve a specific number of rows from a result set. It is typically used in combination with the OFFSET clause to implement pagination and retrieve data in chunks.

IN: The IN operator is used to check if a value matches any value in a specified list or subquery. It is commonly used in WHERE clauses to filter data based on multiple possible values.

CROSS JOIN: A CROSS JOIN, also known as a Cartesian product, combines each row from one table with every row from another table. It results in a Cartesian join, where each row from one table is paired with every row from the other table.

ANY: The ANY operator is used to compare a value with a set of values returned by a subquery. It checks if the value matches any value in the set.

ALL: The ALL operator is used to compare a value with all the values returned by a subquery. It checks if the value matches all the values in the set.

CASE: The CASE statement is used to perform conditional logic in SQL queries. It allows you to evaluate different conditions and return different values based on the conditions.

INNER JOIN: An INNER JOIN combines rows from multiple tables based on a related column between them. It returns only the rows that have matching values in both tables.

LEFT JOIN: A LEFT JOIN, also known as a left outer join, returns all the rows from the left table and the matching rows from the right table. If there is no match, it returns NULL values for the right table columns.

RIGHT JOIN: A RIGHT JOIN, also known as a right outer join, returns all the rows from the right table and the matching rows from the left table. If there is no match, it returns NULL values for the left table columns.

LIMIT: The LIMIT clause is used to limit the number of rows returned by a SELECT statement. It specifies the maximum number of rows to be retrieved from the result set.

DROP: The DROP statement is used to remove database objects, such as tables, views, indexes, or constraints, from the database.

ALTER: The ALTER statement is used to modify the structure of database objects. It can be used to alter tables, columns, constraints, and other database entities.

React , Redux

React Components: React components are the building blocks of a React application. They are reusable, self-contained pieces of code that encapsulate a specific functionality and can be composed together to build complex user interfaces.

JSX: JSX is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. It is used to describe the structure and appearance of React components.

State and Props: State represents the internal data of a component that can change over time. Props (short for properties) are passed from parent components to child components and are used to provide data and configuration to the child components.

React Hooks: React Hooks are functions that allow you to use state and other React features in functional components. They enable functional components to have local state, lifecycle methods, and other features previously available only in class components.

Event Handling: Event handling in React involves defining event handlers and attaching them to DOM elements or React components. It allows you to respond to user interactions, such as button clicks or form submissions.

Conditional Rendering: Conditional rendering in React allows you to display different components or content based on certain conditions. It enables dynamic rendering based on the state of the component or other factors.

Lists and Keys: Lists in React are used to render collections of similar elements. Keys are unique identifiers assigned to each element in a list to help React efficiently update and manipulate the list.

Forms and Form Handling: React provides a way to handle form inputs and manage form state using controlled components. Controlled components store form data in their state and update it based on user input.

Component Communication: React provides various techniques for components to communicate with each other, such as props drilling, context API, and third-party libraries like Redux or MobX.

React Router: React Router is a popular routing library for React applications. It allows you to define and manage different routes in your application, enabling navigation between different pages or views.

Styling in React: Styling in React can be done using CSS-in-JS libraries like styled-components or by using CSS modules. These approaches allow you to encapsulate styles and apply them directly to React components.

React Testing: React testing involves writing tests to ensure that React components and their behavior work as expected. Testing frameworks like Jest and testing libraries like React Testing Library are commonly used for testing React applications.

State Management: State management in React involves managing the application's state and its changes. Redux, MobX, and the Context API are popular state management solutions used in React applications.

React Performance Optimization: React provides techniques for optimizing the performance of React applications, such as using memoization, shouldComponentUpdate, and React's built-in performance profiling tools.

Error Handling: Error handling in React involves catching and handling errors that occur during rendering or component lifecycle methods. Error boundaries, provided by React, help capture and handle errors in a controlled manner.

Server-Side Rendering (SSR): SSR is a technique where the initial rendering of a React application is performed on the server and the HTML is sent to the client. It improves initial page load performance and SEO.

React Hooks and Custom Hooks: React Hooks are used to add state and other React features to functional components. Custom Hooks are reusable functions that encapsulate common stateful logic and can be shared across components.

Component Lifecycle Methods: Lifecycle methods are methods that are invoked at different stages of a component's lifecycle, such as when it is mounted, updated, or unmounted. They provide hooks to perform specific actions during these stages.

Lazy Loading: Lazy loading is a technique where components or code chunks are loaded only when they are needed, typically to improve performance by reducing the initial load time of the application.

Higher Order Components: Higher Order Components (HOCs) are functions that take a component and return a new component with enhanced functionality. They are used for code reuse, cross-cutting concerns, and component composition.

PropTypes: PropTypes is a library used for type-checking and validating the props passed to React components. It helps catch bugs by enforcing correct prop types and improves component documentation.

useCallback and useMemo: useCallback and useMemo are React Hooks used to optimize the performance of functional components. useCallback memoizes callback functions, while useMemo memoizes the result of a function.

Hash Router: Hash Router is a routing technique where the route information is stored in the URL fragment identifier (the part after the hash symbol). It is a simpler alternative to browser history-based routing.

Link and Switch: Link is a React component provided by React Router for declarative navigation between different routes. Switch is another component that renders only the first matching route inside it.

Suspense: Suspense is a React feature that allows components to suspend rendering and show fallback content (such as loading spinners) while waiting for asynchronous data or lazy-loaded components.

Profiler: The Profiler component in React is used to measure and analyze the performance of a React application. It helps identify performance bottlenecks and optimize rendering and component lifecycles.

Render Props: Render Props is a pattern in React where a component accepts a function as a prop and renders that function, passing it data or functionality. It allows component composition and code reuse.

Interceptor: In the context of React, an interceptor is a component or function that intercepts and modifies requests or responses in a network communication layer, such as making API calls or handling authentication.

Custom Hook to Call: Custom hooks are reusable functions that encapsulate logic and stateful behavior. They can be created to abstract common functionality and provide a clean and reusable way to handle specific tasks in React.

Bind: In React, binding refers to the process of attaching an event handler to a component instance to ensure that the correct context (this value) is maintained when the event is triggered.

Context API: Context API is a feature provided by React that allows data to be shared and accessed by multiple components without explicitly passing it through props. It provides a way to avoid prop drilling.

React Fiber: React Fiber is the internal reconciliation algorithm introduced in React 16. It enables React to perform better and more granular control over the rendering process, resulting in improved performance and interactivity.

Shadow DOM: Shadow DOM is a web standard that encapsulates DOM and CSS within a component. It provides a way to encapsulate component styles and DOM structure, preventing conflicts with styles and elements outside the component.

REST API: REST API refers to a set of architectural principles and guidelines for building web services that adhere to the principles of Representational State Transfer (REST). It involves creating endpoints to perform CRUD operations on resources.

Diffing Algorithm: Diffing algorithm, also known as reconciliation, is the process in React that compares the previous virtual DOM with the new virtual DOM to determine the minimal set of changes required to update the actual DOM.

Pure Components: Pure components are a type of React component that automatically implements a shallow equality check for props and state. They are optimized for performance by reducing unnecessary re-rendering.

Limitations of React: React has certain limitations, such as the inability toperform server-side rendering without additional setup, limited support for two-way data binding (compared to frameworks like Angular), and a steeper learning curve for complex state management scenarios.

Redux:

Redux: Redux is a predictable state container for JavaScript applications. It provides a centralized store to manage the application state and follows a unidirectional data flow pattern.

Action: Actions in Redux are plain JavaScript objects that represent an intention to change the state. They are dispatched to the store and trigger the corresponding state updates.

Dispatch: Dispatch is a function provided by the Redux store that is used to send actions to the store. It is the way to trigger state changes in Redux.

Store: The Redux store is an object that holds the application state. It allows access to the state, dispatches actions, and manages the state updates.

Reducer: A reducer is a pure function that takes the current state and an action as input and returns the new state. It specifies how the state should be updated in response to different actions.

UI Binding: UI binding in Redux involves connecting the application's UI components to the Redux store. It enables the components to access the state and trigger state changes through actions.

Portals: Portals in React and Redux provide a way to render components outside the normal component tree hierarchy. They allow components to be rendered at a different DOM location, such as modals or overlays.

Class Components: Class components are React components defined as JavaScript classes. They are the traditional way of defining components and provide access to component lifecycle methods.

<u>Data Structure</u>

Arrays: Arrays are ordered collections of elements, stored in contiguous memory locations. They allow you to store and access multiple values using numerical indices.

Linked Lists: Linked lists are data structures made up of nodes, where each node contains data and a reference to the next node in the sequence. They provide dynamic memory allocation and efficient insertion/deletion at the expense of random access.

Stacks: Stacks are data structures that follow the Last-In-First-Out (LIFO) principle. Elements are added and removed from only one end, known as the top of the stack.

Queues: Queues are data structures that follow the First-In-First-Out (FIFO) principle. Elements are added to the rear (enqueue) and removed from the front (dequeue) of the queue.

Trees: Trees are hierarchical data structures composed of nodes. Each node can have child nodes, forming parent-child relationships. Trees are used to represent hierarchical relationships and enable efficient search, insertion, and deletion operations.

Heaps: Heaps are complete binary trees that satisfy the heap property. They can be min-heaps (the value of each node is greater than or equal to its parent) or max-heaps (the value of each node is less than or equal to its parent).

Graphs: Graphs consist of nodes (vertices) and edges that connect these nodes. They can be used to represent relationships and connections between objects. Graphs can be directed or undirected and can have weighted or unweighted edges.

Hash Tables: Hash tables (or hash maps) are data structures that use hash functions to store and retrieve values based on a unique key. They provide efficient lookup, insertion, and deletion operations.

Tries: Tries (also known as prefix trees) are tree-like data structures used to efficiently store and search for strings or sequences. They are particularly useful for searching for words or autocomplete functionality.

Graph Algorithms: Graph algorithms are algorithms designed to solve problems on graphs, such as finding shortest paths, detecting cycles, traversing graphs, or determining connectivity.

Sorting Algorithms: Sorting algorithms arrange elements in a specific order, such as ascending or descending. Common sorting algorithms include bubble sort, insertion sort, selection sort, merge sort, quicksort, and heapsort.

Searching Algorithms: Searching algorithms aim to find a specific element within a data structure. Common searching algorithms include linear search, binary search, depth-first search (DFS), and breadth-first search (BFS).

Big O Notation: Big O notation is used to describe the time complexity (execution time) and space complexity (memory usage) of algorithms. It helps analyze and compare the efficiency of different algorithms.

Static and Memory Allocation: Static allocation refers to the allocation of memory at compile-time, while dynamic allocation involves allocating memory at runtime. Understanding memory allocation is essential for efficient memory management.

Pros and Cons of Recursion: Recursion is a programming technique where a function calls itself. It simplifies problem-solving in some cases but can lead to performance issues and stack overflow errors if not used carefully.

Jagged Array: A jagged array is an array where each element can be an array of different lengths. It provides flexibility in representing irregularly shaped data structures.

Linear and Nonlinear Data Structures: Linear data structures store elements in a linear order, such as arrays, linked lists, stacks, and queues. Nonlinear data structures, like trees and graphs, allow elements to be connected in multiple ways.

Applications: Data structures have various applications, such as organizing and managing data in databases, implementing algorithms, representing networks or hierarchies, and optimizing operations like searching and sorting.

Divide and Conquer Strategy: The divide and conquer strategy involves breaking down a problem into smaller, more manageable subproblems, solving them independently, and combining the solutions to solve the original problem.

Underflow in Stack: Underflow in a stack occurs when you try to pop an element from an empty stack. It is an error condition that needs to be handled to prevent unexpected behavior.

Hash Functions: Hash functions are algorithms that convert input data (such as keys) into a fixed-size numeric value (hash code). They are used in hash tables for efficient data retrieval.

Spanning Tree and Minimum Spanning Tree: A spanning tree is a subgraph that includes all the nodes of a connected graph without forming any cycles. A minimum spanning tree is a spanning tree with the minimum total edge weight.