MERN Stack Interview Questions [2024-25]

Q1. Who is a Mern Stack Developer?

A MERN Stack Developer is a skilled programmer who specializes in building web applications using four key technologies: MongoDB, Express, React, and Node.js. These technologies work together to create both the front-end (what the user sees and interacts with) and back-end (the server-side logic that powers the application) of a website.

Q2. List the abbreviation of MERN

MERN in abbreviated form is:

- **MongoDB**: A NoSQL database for storing data.
- **Express.js**: A lightweight web application framework for Node.js.
- **React**: A JavaScript library for building user interfaces.
- **Node.js**: A JavaScript runtime for server-side development.

Its popularity stems from its end-to-end use of JavaScript, which simplifies development and enables efficient, real-time data flow between client and server. This makes the MERN Stack an excellent choice for single-page applications (SPAs) and dynamic websites.

Q3. Explain the MVC architecture?

The Model-View-Controller (MVC) framework is a way of organizing code for web applications. It separates the application into three parts: the Model, the View, and the Controller. Each part has a specific job to do. The Model is responsible for storing and managing data. It represents the data in a way that is easy for the application to understand.

Q4. What Is Replication In MongoDB?

Replication in MongoDB involves creating multiple copies of data across different servers, known as replica sets. This process enhances read capacity by allowing clients to distribute read operations across these replica sets. Storing data copies across various data centers improves data localization and availability for distributed applications. Additionally, maintaining surplus copies serves specific purposes such as backup, reporting, and disaster recovery.

Q5. What in React are Higher-Order Components (HOC)?

A higher-order component (HOC) is a function that takes a component as input and generates another component. Essentially, it stems from the compositional structure of React. These components are termed “pure” because they can adopt any dynamically provided child component without replicating or altering the behavior of the input components.

HOC may be utilised in the following usage cases:

- Reuse of code, reasoning, and bootstrap abstraction
- Represent Highjacking
- State manipulation and abstraction
- Props manipulation

Q6. What is Reconciliation in React JS?

React assesses the necessity for a real DOM update when there’s a change in a component’s props or state. This evaluation involves comparing the newly returned element with the one previously displayed. If they are not equal, React proceeds to update the DOM. This process is referred to as reconciliation.

Q7. What is Sharding in MongoDB?

Sharding is a method for distributing data across multiple machines to facilitate data sharing. MongoDB utilizes sharding to support installations with extensive data sets and demanding performance requirements. This enables MongoDB to achieve horizontal scalability. At the collection level, MongoDB distributes data across the shards in the cluster.

Q8. What is the purpose of MongoDB?

MongoDB serves as a document-oriented database manager specifically crafted for the storage of substantial data volumes. It stores data in a binary JSON format and incorporates the concepts of collections and documents. Being a cross-platform, NoSQL database, MongoDB is distinguished by its high performance, scalability, and flexibility, enabling smooth querying and indexing operations.

Q9. What is the purpose of ExpressJS?

ExpressJS is a web application framework meticulously crafted to facilitate the development and hosting of Node.js projects. Under the MIT license, it stands as an open-source framework. ExpressJS adeptly orchestrates the interaction between the front-end and the database, ensuring a secure and seamless data transfer.

Q10. What are the data types in MongoDB?

MongoDB accommodates an extensive array of data types as values within its documents. In MongoDB, documents bear a resemblance to JavaScript objects, and they adhere to the key/value-pair structure inherent in JSON. Beyond the fundamental key/value concept of JSON, MongoDB introduces support for various additional data types. The prevalent data types in MongoDB include:

- Null
- Boolean
- Number
- String
- Date
- Regular expression
- Array
- Embedded document
- Object ID
- Binary Data

Q11. What is REPL In Node JS?

REPL, short for “Read Eval Print Loop,” is a straightforward program designed to receive commands, assess them, and display the outcomes. Its purpose is to establish an environment akin to a Unix/Linux shell or a Windows console, allowing users to input commands and queries while receiving corresponding outputs. The functions performed by REPL include:

- 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.
- LOOP – This loops the above command until the user presses Ctrl+C twice.

Q12. What is meant by “Callback” in Node JS?

A callback serves as the asynchronous counterpart to a function. Node.js extensively utilizes callbacks, invoking them upon the conclusion or completion of a specific task. For example, consider a function tailored for file reading; it initiates the file reading process and promptly relinquishes control to the execution environment, enabling the execution of subsequent instructions.

Q13. What are pure components in MERN Stack?

In the MERN stack, pure components can be viewed as standard components, differing primarily in their engagement with the shouldComponentUpdate method. Pure components are primarily tasked with conducting a comparison of props and state whenever there is a change in either props or state.

Q14. How does Node JS handle Child Threads?

Node.js, in its most basic state, operates as a single-threaded process. Developers lack access to child threads or thread management techniques. Although certain operations, like asynchronous I/O, prompt the creation of child threads, these activities occur in the background without disrupting the main event loop or executing any JavaScript code for the application.

Q15. What are some features of MongoDB?

- **Indexing:** It offers support for generic secondary indexes and delivers distinctive capabilities for unique, compound, geospatial, and full-text indexing.
- **Aggregation:** It furnishes an aggregation framework founded on the concept of data processing pipelines.
- **Special collection and index types:** It includes support for time-to-live (TTL) collections, allowing data that should expire at a specific time to be managed effectively.
- **File storage:** It offers a user-friendly protocol for storing extensive files and their corresponding metadata.
- **Sharding:** Sharding involves the segmentation of data across multiple machines.

Q16. How do you manage packages in your node.js project?

The management of dependencies can be handled by various package installers along with their corresponding configuration files. Among them, npm and yarn are commonly utilized. Both these tools offer comprehensive JavaScript libraries, incorporating advanced features for controlling environment-specific configurations. To ensure consistency in library versions across a project, package.json and package-lock.json are employed. This practice helps avoid compatibility issues when transitioning the application to different environments.

Q17: How to handle routing in Express JS?

Express.js manages routing through the use of the `express.Router()` method. This method yields an instance of a router, enabling the definition of routes for the application. Below is an illustration of how to define a basic route using this router:

```
const express = require('express')
const router = express.Router()
router.get('/', (req, res) => {
  res.send('Hello, World!')
}
module.exports = router
```

Q18: What is middleware in Node.js and how is it used?

In Node.js, middleware is a function that takes in the request and response objects, as well as the next middleware function in the application's request-response cycle. It can be employed to alter the request or response objects, as well as to execute various tasks such as logging, authentication, and error handling.

```
//Here's an example of a middleware function that logs the request method and URL:
```

```
function logMiddleware(req, res, next) {
  console.log(`[${req.method}] ${req.url}`);
  next();
}
app.use(logMiddleware);
```

Q19. What is RESTful API?

A RESTful API (Representational State Transfer Application Programming Interface) is an architectural style for constructing web APIs. It utilizes HTTP methods like GET, POST, PUT, and DELETE to execute CRUD (create, read, update, delete) operations on resources, identifying these resources through URLs. A key characteristic of a RESTful API is its statelessness, signifying that each request carries all the essential information for its completion.

Q20. Explain the event loop in Node JS.

The event loop in JavaScript facilitates asynchronous programming. In JS, all operations occur on a single thread, yet by employing clever data structures, we can simulate the effect of multithreading. The Event Loop manages asynchronous tasks by queuing and listening to events.

Q21. What are node JS streams?

Streams in Node.js are instances of EventEmitter designed for handling streaming data. They prove particularly useful in managing and manipulating large files, such as videos or mp3s, over the network. Streams employ buffers as temporary storage. There are primarily four main types of streams:

- **Writable:** Streams to which data can be written, exemplified by `fs.createWriteStream()`.
- **Readable:** Streams from which data can be read, as illustrated by `fs.createReadStream()`.
- **Duplex:** Streams that are both Readable and Writable, exemplified by `net.Socket`.
- **Transform:** Duplex streams, capable of modifying or transforming data as it is both written and read, such as `zlib.createDeflate()`.

Q22. What are Node JS buffers?

Buffers, in a general sense, are temporary memory utilized by streams to retain data until it is consumed. Unlike JavaScript's Uint8Array, buffers introduce additional use cases and are primarily employed to represent a fixed-length sequence of bytes. Buffers support legacy encodings such as ASCII, utf-8, etc. They are allocated as fixed (non-resizable) memory outside the V8 engine.

Q23. Why use Express.js over Node.js?

In backend development, Node.js is commonly paired with Express.js for improved ease and scalability. While Vanilla JavaScript suffices for front-end coding, larger web apps benefit from React or Angular. A full app with just Node.js can lead to complex code, so combining Node.js with Express.js leverages speed and simplicity, creating scalable web APIs. This synergy is seen in popular stacks like MEAN and MERN.

Q24. What is MongoDB?

- MongoDB is an open-source NoSQL database written in C++ language. It uses JSON-like documents with optional schemas.
- It provides easy scalability and is a cross-platform, document-oriented database.
- MongoDB works on the concept of Collection and Document.
- It combines the ability to scale out with features such as secondary indexes, range queries, sorting, aggregations, and geospatial indexes.
- MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

Q25. What is a Collection in MongoDB?

In MongoDB, a collection is a set of documents. If a document is akin to a row in a relational database, then a collection can be likened to a table. Documents within a collection can possess varying "shapes," meaning collections have dynamic schemas.

Q26. Explain the term "Indexing" in MongoDB.

In MongoDB, indexes play a crucial role in optimizing query resolution. Essentially, an index stores a compact portion of the dataset in a format conducive to efficient traversal. It retains the values of a specific field or set of fields, organized based on the specified field values within the index.

Q27: Is Node.js entirely single-threaded?

No, Node.js is not entirely single-threaded. It uses an event-driven, non-blocking I/O model that allows multiple operations to be processed simultaneously. However, the execution of JavaScript code is single-threaded.

Q28. How to Connect Node.js to a MongoDB Database?

MongoDB is a NoSQL database used to store large amounts of data without any traditional relational database table. Instead of rows & columns, MongoDB used collections & documents to store data. A collections consist of a set of documents & a document consists of key-value pairs which are the basic unit of data in MongoDB.

Let’s see how we can connect nodejs with MongoDB:

```
const express = require(“express”);
const ejs = require(“ejs”);
const mongoose = require(“mongoose”);
const bodyParser = require(“body-parser”);
mongoose.connect(“mongodb://localhost:27017/newCollection”, {
useNewUrlParser: true,
useUnifiedTopology: true
});
const contactSchema = {
email: String,
query: String,
};
const Contact = mongoose.model(“Contact”, contactSchema);
const app = express();
app.set(“view engine”, “ejs”);
app.use(bodyParser.urlencoded({
   extended: true
}));
app.use(express.static(__dirname + '/public'));
app.get(“/contact”;, function(req, res){
res.render(“contact”);
});
app.post(&quot;/contact&quot;, function (req, res) {
  console.log(req.body.email);
const contact = new Contact({
   email: req.body.email,
   query: req.body.query,
});
contact.save(function (err) {
   if (err) {
      throw err;
   } else {
      res.render(“contact”);
   }
});
});
app.listen(3000, function(){
   console.log(“App is running on Port 3000”);


});
```

Q29. Can you elaborate on the MongoDB Aggregation Pipeline?

The MongoDB Aggregation Pipeline serves as a framework for data processing and transformation within MongoDB. It involves a series of sequential stages, facilitating operations like filtering, projection, grouping, and sorting on documents. Each stage in the pipeline processes the data and forwards the results to the subsequent stage, culminating in the generation of the final output.

Q30. How can you use the like operator to query MongoDB?

In MongoDB, you can implement a functionality similar to the “like” operator by employing regular expressions with the `$regex` operator in the `$match` pipeline stage of an aggregation query. For instance, the subsequent query identifies documents in which the “name” field commences with “Nick”:

```
db.myCollection.aggregate([
  { $match: { name: { $regex: /^Nick/ } } }
])
```

Q31. Name a few techniques to optimize React app performance.

Some techniques to optimize React app performance include:

- Memorization: Enhance performance by memoizing functions and components, preventing unnecessary re-renders.
  Virtualization: Leverage libraries such as `react-virtualized` for optimized rendering of extensive lists or grids.
- Code Splitting: Divide your code into smaller segments and load them selectively based on necessity.
- Lazy Loading: Employ React's `lazy()` and `Suspense` to load components lazily.
- Optimize Renders: Use shouldComponentUpdate, PureComponent, or React.memo to prevent unnecessary renders.
- Minimize Re-renders: Utilize `shouldComponentUpdate`, `PureComponent`, or `React.memo` to minimize unnecessary renders.
- Avoid Unnecessary State Updates: Exercise caution with `setState` to minimize unnecessary re-renders.
- Server-Side Rendering (SSR): Optimize initial load times by rendering components on the server side.

Q32. What is the purpose of the module.exports?

In Node.js, a module consolidates cohesive code into a singular unit that can be parsed by consolidating relevant functions within a single file. Exporting a module involves defining and exporting functions, allowing them to be imported into other files using the required keyword.

Q33. Can you explain CORS?

CORS, which stands for Cross-Origin Resource Sharing, is a mechanism based on HTTP headers. It facilitates the ability of a web application hosted at one origin (domain) to request access to resources from a server located at a different origin. In essence, CORS is a browser-based system that permits controlled access for scripts running on a client's browser to interact with and retrieve resources from origins or domains beyond their own.

Q34. What is DOM diffing?

When elements are rendered twice, the Virtual DOM initiates a comparison process to identify the components that have undergone changes. It identifies and focuses on the altered components on the page, excluding those that remain unchanged. This approach minimizes DOM modifications resulting from user interactions and enhances browser performance by optimizing DOM manipulation. The primary objective is to execute functions swiftly and efficiently.

Q35. What is the role of MongoDB in the MERN Stack?

MongoDB acts as the database in the MERN Stack, storing data in a flexible, document-oriented format. Instead of traditional rows and columns, MongoDB uses collections of JSON-like documents. This schema-less structure makes it highly adaptable for modern applications where data structures can evolve rapidly.

Q36. Can you explain the difference between SQL and NoSQL databases?

- **SQL Databases**:
  - Use structured tables with predefined schemas.
  - Suitable for applications with fixed schemas (e.g., financial systems).
  - Examples: MySQL, PostgreSQL.
- **NoSQL Databases**:
  - Use dynamic schemas and are more flexible.
  - Ideal for handling unstructured or semi-structured data like user profiles, logs, or sensor data.
  - Examples: MongoDB, CouchDB.

Q37. What is Express.js, and how does it work?

Express.js is a web application framework for Node.js that simplifies the process of building server-side logic. It provides:

- A robust routing mechanism for handling HTTP requests.
- Middleware to manage tasks like logging, authentication, and data parsing.
- APIs for building RESTful services efficiently.

It acts as the glue between the frontend (React) and the database (MongoDB), enabling smooth communication.

Q38. What is Node.js, and why is it used in the MERN Stack?

Node.js is a runtime environment that allows developers to run JavaScript on the server side. Its key benefits in the MERN Stack include:

- Non-blocking, event-driven architecture for handling multiple concurrent connections.
- A unified programming language (JavaScript) for the frontend and backend.
- Rich ecosystem of libraries through npm.

Q39. What is npm, and why is it important in MERN Stack development?

npm (Node Package Manager) is a tool for:

- Installing libraries and frameworks like React, Express, or Mongoose.
- Managing project dependencies.
- Running scripts for tasks like testing, building, or deploying.

In MERN projects, npm simplifies the setup and maintenance of libraries required for development.

Q40. How does the MERN Stack handle routing?

Routing is managed differently on the frontend and backend:

- **Frontend (React)**: React Router is used to handle client-side routing, enabling seamless navigation without refreshing the page.

- **Backend (Express.js)**: Routes are defined to handle HTTP requests (GET, POST, PUT, DELETE) and serve API responses.

Q41. What is middleware in Express.js?

Middleware is software between an application and another service (like a database or server). It can intercept requests and responses and perform actions like logging, authentication, or data manipulation before passing them on. Express.js uses middleware extensively.

Middleware in Express.js is a function that executes during the lifecycle of a request. For example:

- **Application-level middleware**: Runs globally for all routes.
- **Router-level middleware**: Runs for specific routes.

Example middleware:

```
app.use((req, res, next) => {

  console.log('Request received');

  next();

});
```

Q42. How do you create a REST API with Express.js?

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

const app = express();

app.use(express.json());

// Sample endpoints

app.get('/api/items', (req, res) => {
    res.json({ message: 'Get all items' });
});
app.post('/api/items', (req, res) => {
  res.json({ message: 'Item created' });
});
// Start server
app.listen(3000, () => console.log('API is running on port 3000'));
});
```

This API handles GET and POST requests for a sample /api/items endpoint.

Q43. What is CORS, and why do you need it in the MERN Stack?

CORS (Cross-Origin Resource Sharing) is a security feature implemented by browsers to restrict access to resources from a different domain. In a MERN Stack application, the backend (Node.js/Express) and frontend (React) might run on different domains or ports, triggering CORS errors.

You can enable CORS in Express.js using the cors middleware:

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

app.use(cors());
```

Q44. How do you implement authentication in the MERN Stack?

Authentication in the MERN Stack is typically done using JWT (JSON Web Tokens):

1. **Backend (Node.js)**:
   - Generate a JWT when a user logs in.
   - Use middleware to verify the token for protected routes.

```
const jwt = require('jsonwebtoken');

const secretKey = 'yourSecretKey';

app.post('/login', (req, res) => {

  const token = jwt.sign({ userId: req.body.id }, secretKey, { expiresIn: '1h' });

  res.json({ token });

});
```

2. **Frontend (React)**:
   - Store the token in localStorage or cookies.
   - Include the token in API requests for protected routes.

Q45. How do you deploy a MERN Stack application?

Deploying a MERN Stack application involves hosting each component:

1. **Frontend (React)**:
   - Use platforms like Netlify or Vercel.
   - Run npm run build to create an optimized production build.
2. **Backend (Node.js/Express)**:
   - Use hosting services like Heroku, AWS, or Render.
3. **Database (MongoDB)**:
   - Use MongoDB Atlas for cloud-based hosting.

Ensure proper environment variables for production, including database URIs and API keys.

Q46. What are higher-order components (HOCs) in React?

HOCs are functions that take a component and return a new enhanced component. They are used to reuse logic across multiple components.

Example:

```
function withLogger(WrappedComponent) {

  return function(props) {

    console.log('Props:', props);

    return <WrappedComponent {...props} />;

  };

}
```

Q47. How does MongoDB handle schema design for large-scale applications?

MongoDB supports flexible schemas, which can be optimized for performance:

- **Embedded Documents**: Nest related data in a single document for quick reads.
- **References**: Use references for relationships that require frequent updates or large data.

For scaling, MongoDB provides sharding, which distributes data across multiple servers.

Q48. Write an example of a schema in Mongoose.

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

const userSchema = new mongoose.Schema({

  name: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  password: { type: String, required: true },

  createdAt: { type: Date, default: Date.now },

});

const User = mongoose.model('User', userSchema);

module.exports = User;
```

This schema defines a user model with fields like name, email, and password, along with timestamps.

Q49. What are WebSockets, and how do you use them with Node.js?

WebSockets enable real-time, two-way communication between the client and server. Use the socket.io library for implementation:

```
const io = require('socket.io')(server);

io.on('connection', (socket) => {

  console.log('A user connected');

  socket.on('message', (msg) => console.log(msg));
});
```

Q50. How do you integrate third-party APIs in a MERN application?

- Use libraries like axios or the fetch API to make HTTP requests.
- Handle authentication (if required) by including API keys in headers.

```
axios.get('https://api.example.com/data', { headers: { Authorization: 'Bearer token' } });
```

Q51. What is the difference between front-end and back-end development?

- Front-end development: Focuses on the visual elements and user interaction of a website or application. Technologies used include HTML, CSS, and JavaScript to create the user interface and user experience (UI/UX).
- Back-end development: Deals with the server-side logic, databases, and APIs (Application Programming Interfaces) that power the website's functionality. Languages like Python, Ruby, PHP, or Node.js are commonly used.

Q52. What is the difference between GET and POST HTTP methods?

- GET: Used to retrieve data from a server. Data is typically sent as query parameters appended to the URL. GET requests are considered safe and idempotent (repeating the request doesn't change server state).
- POST: Used to send data to the server, often for creating or updating resources. Data is sent in the request body. POST requests are not idempotent (repeating might create duplicate data).

Q53. Explain the concept of RESTful APIs.

REST (REpresentational State Transfer) APIs follow architectural principles for building web APIs. They use HTTP verbs (GET, POST, PUT, DELETE), standard data formats (JSON), and resource-based URLs to provide a predictable and scalable way to interact with server-side functionality.

Q54. Explain the purpose of a web server.

A web server is a software that receives HTTP requests from web browsers and returns responses. It stores web page files, processes server-side scripts, and interacts with databases to deliver user content. Common web servers include Apache and Nginx.

Q55. What is the difference between synchronous and asynchronous programming?

- Synchronous: Code execution blocks until a task is completed before moving on to the next line. This can lead to unresponsive user interfaces if tasks take a long time.
- Asynchronous: Code execution continues while waiting for an operation to finish. Callbacks or promises handle the response when the operation completes, improving responsiveness.

Q56. How do you use promises in JavaScript?

Promises represent the eventual result of an asynchronous operation (success or failure). They provide a cleaner way to handle asynchronous code compared to callbacks. You can chain promises for handling complex asynchronous workflows.

Q57. What is the Fetch API?

The Fetch API is a modern, browser-built API for making asynchronous HTTP requests. Compared to XMLHttpRequest, it provides a cleaner syntax and promise-based approach.

Q58. What is Express.js, and how is it used?

Express.js is a popular Node.js web application framework that simplifies building web applications. It provides features like routing, middleware, templating engines, and simplifies handling HTTP requests and responses.

Q59. Explain the difference between SQL and NoSQL databases.

- SQL (Structured Query Language): Relational databases with a fixed schema (data structure) and access data using SQL queries. Examples: MySQL, PostgreSQL.
- NoSQL (Not Only SQL): Non-relational databases with flexible schemas suited for handling large amounts of unstructured or diverse data. Examples: MongoDB, Cassandra.

Q60. What is Mongoose, and why is it used?

Mongoose is an ODM library for MongoDB in Node.js. It provides a layer of abstraction over the native MongoDB driver, allowing developers to interact with MongoDB using a more object-oriented approach. Mongoose defines schemas for data structure, simplifies CRUD operations, and offers features like validation and middleware.

Q61. Explain the concept of authentication and authorization.

- Authentication: Verifies a user's identity (who they are). Common methods include username/password login, social logins, or tokens.
- Authorization: Determines what a user is allowed to do (permissions). This involves checking user roles or access levels to control their actions within the application.

Q62. How do you implement user authentication in a web application?

There are several ways to implement user authentication:

- **Session-based**: Stores user information (like a session ID) on the server side and a cookie on the client side.
- **Token-based (JWT):** Issues a token (JSON Web Token) to the client after a successful login. The client includes the token in subsequent requests for authorization.
- **Social login:** Allows users to log in with existing social media accounts (e.g., Google, Facebook).

Q63. What is JWT (JSON Web Token)?

JWT (JSON Web Token) is a compact, self-contained token containing encoded information (user ID, roles) for authentication. The token is signed with a secret key, allowing the server to verify its authenticity. JWTs are stateless (don't require server-side session management) and popular for building APIs.

Q64. How do you secure a RESTful API?

Securing a RESTful API involves several measures:

- Authentication and authorization: Ensure only authorized users can access resources.
- HTTPS: Encrypt communication between client and server to protect data transmission.
- Input validation: Sanitize user input to prevent injection attacks (SQL injection, XSS).
- Rate limiting: Prevent denial-of-service attacks by limiting API requests per user.

Q65. What are WebSockets, and how are they used?

WebSockets are a two-way communication channel between a browser and a server. Unlike HTTP requests, WebSockets provide a persistent connection, allowing real-time data exchange. This is useful for features like chat applications, live updates, or collaborative editing.

Q66. Explain the concept of server-side rendering (SSR).

Server-side rendering (SSR) is a technique where the server generates the initial HTML content of a web page before sending it to the browser. This approach improves initial page load performance and SEO (Search Engine Optimization) as search engines can easily crawl and index the content.

Q67. What are microservices and why are they important?

Microservices are an architectural style for building applications as a collection of small, independent services. Each service focuses on a specific business capability and communicates with others through APIs. This approach promotes modularity, scalability, and independent deployment of services.

Q68. How do you implement microservices in a web application?

Implementing microservices involves:

- Breaking down the application into small, well-defined services.
- Developing each service using appropriate technologies (e.g., Node.js, Python).
- Defining clear APIs for communication between services.
- Using a containerization technology like Docker for packaging and deployment.

- Employing an orchestration platform like Kubernetes to manage service lifecycles and scaling.

Q69. What is GraphQL and how does it differ from REST?

GraphQL is a query language for APIs that allows clients to request specific data they need rather than fetching entire datasets returned by traditional REST APIs. This reduces data over-fetching and improves performance.

Q70. How do you implement GraphQL in a Node.js application?

You can use libraries like graphql or apollo-server to implement a GraphQL server in Node.js. These libraries provide tools for defining schemas, resolvers (functions that fetch data), and handling incoming GraphQL queries.

Q71. What is Docker and how is it used in development?

Docker is a containerization platform that allows developers to package applications with their dependencies into standardized units called containers. Containers ensure consistent environments across development, testing, and production, simplifying deployment and collaboration.

Q71. Explain the concept of containerization.

Containerization is a virtualization technique that packages an application with its dependencies (libraries, configuration files) into a lightweight, portable container. This ensures the application runs consistently regardless of the underlying environment.

Q72. What is Kubernetes and how does it work?

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. It allows you to orchestrate multiple containers, schedule their deployment across a cluster of machines, and manage scaling based on resource needs.

Q73. What are some common security vulnerabilities in web applications?

- SQL injection: Exploiting user input to inject malicious SQL code into database queries.
- Cross-Site Scripting (XSS): Injecting malicious scripts into web pages, executed by users' browsers.
- Cross-Site Request Forgery (CSRF): Tricking a user's browser into performing unauthorized actions on a trusted website.
- Insecure data handling: Not properly validating and sanitizing user input, leading to potential data breaches.

Q74. What is Cross-Site Scripting (XSS) and how do you prevent it?

XSS involves injecting malicious scripts into web pages that can steal user data, redirect users to phishing sites, or deface the website. You can prevent XSS by:

- Escaping user input: Encode special characters before displaying them in the web page.
- Validate user input: Ensure user input conforms to expected formats.
- Use HTTP Only flags for sensitive cookies: Mitigate the risk of XSS attacks stealing cookie data.

Q75. What is Cross-Site Request Forgery (CSRF) and how do you prevent it?

CSRF attacks trick a user's authenticated browser into performing unauthorized actions on a trusted website. Here's how to prevent CSRF:

- Use CSRF tokens: Generate unique tokens for each user session and include them in forms or API requests.
- Implement the SameSite attribute for cookies: Restrict cookie accessibility to mitigate the risk of CSRF attacks.

Q76. How do you handle large-scale data in a web application?

Handling large datasets involves choosing appropriate storage solutions and optimizing data access. Here are some approaches:

- NoSQL databases: Consider using NoSQL databases (e.g., MongoDB) for storing large amounts of unstructured or diverse data.
- Data partitioning: Divide data into smaller chunks for faster retrieval and management.
- Caching: Implement caching mechanisms to store frequently accessed data in memory for faster retrieval.
- Database optimization: Design efficient database schemas and queries to optimize data access times.

Q77. What is the difference between horizontal and vertical scaling?

- Horizontal scaling: Adding more servers to distribute workload and handle increased traffic.
- Vertical scaling: Upgrading existing server hardware (CPU, RAM) to increase processing power for a single server.

Q78. How do you ensure the scalability of a web application?

Scalability is the ability of an application to handle growing traffic and data volumes. Here are some strategies:

- Choose a cloud-based architecture: Cloud platforms offer elastic scalability to automatically provision resources based on demand.
- Design for horizontal scaling: Build your application with loosely coupled components to enable easy addition of more servers.
- Implement caching mechanisms: Reduce database load and improve performance by caching frequently accessed data.
- Monitor performance metrics: Proactively identify performance bottlenecks and scale resources accordingly.

Q79. Why is MERN stack popular for web development?

The MERN stack is popular due to several reasons:

JavaScript-based: Using JavaScript throughout the entire stack simplifies development and reduces learning curve.

Open-source: All components are open-source, providing a large community, extensive documentation, and readily available resources.

Scalability: MERN applications can scale effectively to handle increasing traffic and data volumes.

Full-stack development: It enables developers to build both front-end and back-end aspects of the application using a single language.

Q80. What are the advantages of using MongoDB as a database?

MongoDB offers several advantages:

Document-oriented: Stores data in flexible JSON-like documents, making it suitable for unstructured and semi-structured data.

Scalability: Can scale horizontally by adding more servers, accommodating increasing data volumes and traffic.

High performance: Designed for fast data access and retrieval.

Easy to learn: Its query language is intuitive and similar to JavaScript syntax.

Q81. Explain the role of Express.js in MERN stack development.

Express.js acts as the back-end framework in a MERN application. It provides:

Routing: Defines how different URLs are handled and mapped to specific functionalities.

API development: Enables building RESTful APIs to interact with the front-end and database.

Middleware: Allows implementing functionalities like authentication, authorization, and logging.

Template engine support: Supports templating languages like Pug or EJS for server-side rendering.

Q82. How does Node.js work in a MERN application?

Node.js serves as the runtime environment for JavaScript on the server. It:

Executes JavaScript code: Allows running JavaScript code outside of the browser, enabling server-side development.

Provides asynchronous I/O: Efficiently handles multiple requests simultaneously without blocking the main thread.

Offers a rich ecosystem: Provides a vast collection of packages and modules for various functionalities.

Handles communication: Facilitates communication between the front-end (React.js) and back-end (Express.js).

Q83. What is the purpose of a database schema in MongoDB?

The database schema in MongoDB defines the structure of documents within a collection. It:

Specifies data types: Defines the data types of fields in each document (e.g., strings, numbers, dates, arrays).

Ensures data consistency: Helps maintain data integrity by enforcing data type validation.

Improves query performance: Enables efficient data retrieval and indexing.

Facilitates data modeling: Helps organize data in a meaningful way for better understanding and management.

Q84. Explain the concept of RESTful APIs in MERN stack development.

RESTful APIs are a set of guidelines for designing web APIs based on HTTP protocols. They:

Use HTTP verbs: Utilize HTTP verbs (GET, POST, PUT, DELETE) to perform CRUD operations (create, read, update, delete) on resources.

Define resources: Represent data entities as resources (e.g., users, products) with unique URLs.

Follow statelessness: Each request should be independent and not rely on previous interactions.

Enable communication: Facilitate data exchange between the front-end (React.js) and back-end (Express.js) through API calls.

Q85. What is the purpose of a package manager like npm or yarn in MERN development?

Package managers like npm and yarn are crucial for managing project dependencies in MERN development. They:

Install packages: Download and install necessary libraries and modules for the project.

Manage dependencies: Track the versions of installed packages and their dependencies.

Resolve conflicts: Handle dependency conflicts and ensure compatibility between packages.

Share packages: Allow publishing and sharing custom packages with other developers.

Q86.How would you debug a MERN application?

Debugging a MERN application involves understanding where the issue arises and using appropriate tools:

Browser console: Use browser developer tools to inspect logs, errors, and network requests.

Node.js debugger: Utilize the Node.js debugger to step through code execution and inspect variables.

Logging: Add logging statements to track application flow and identify problematic areas.

Test-driven development (TDD): Write tests to identify and isolate bugs early in development.

Q87. What are some common security vulnerabilities to consider when developing a MERN application?

MERN applications are susceptible to various security vulnerabilities:

Cross-Site Scripting (XSS): Malicious scripts injected into the application to steal user data.

SQL injection: Malicious SQL queries injected into the database to access or modify sensitive data.

Authentication and authorization vulnerabilities: Weak authentication mechanisms or improper authorization can lead to unauthorized access.

Data leaks: Sensitive data exposed through insecure configurations or practices.

Q88. How do you deploy a MERN application?

Deploying a MERN application typically involves the following steps:

Choose a deployment platform: Select a hosting provider or cloud service (e.g., Heroku, AWS, Netlify).

Configure the server: Set up the server environment (Node.js, Express.js) and configure database access.

Build the application: Compile the front-end (React.js) code for production and bundle it.

Deploy the front-end: Upload the compiled front-end code to the server.

Start the server: Run the back-end server (Node.js) to handle requests.

Q89. What are some popular tools and libraries used for MERN development?

MERN development utilizes numerous tools and libraries:

React Router: Enables routing and navigation within a React application.

Axios: A popular library for making HTTP requests.

Mongoose: An ODM (Object Document Mapper) that simplifies MongoDB interaction.

Jest: A popular JavaScript testing framework.

Babel: Transpiles modern JavaScript code to be compatible with older browsers.

Webpack: A module bundler for packaging front-end code for production.

Q90. How do you test a MERN application?

Testing a MERN application involves testing different layers:

Unit testing: Test individual components and functions in isolation.

Integration testing: Test how different parts of the application work together.

End-to-end (E2E) testing: Simulate user interactions and validate the application's overall functionality.

API testing: Test the functionality of APIs, including data validation and response handling.

Q91. What is the difference between `npm install` and `npm update`?

`npm install` and `npm update` are used to manage project dependencies, but they have different purposes:

`npm install`: Installs packages for the first time or if they are missing from the project.

`npm update`: Updates existing packages to their latest versions.

`npm install`: Follows package versions specified in the `package.json` file.

`npm update`: Installs the latest versions of all packages, even if not specified in `package.json`.

Q92. What are some common methods for handling errors in a MERN application?

Error handling is crucial for reliable applications:

Try...catch blocks: Use try...catch blocks to handle potential errors during code execution.

Error middleware: Implement error middleware in Express.js to catch unhandled errors and provide appropriate responses.

Error handling in React: Use error boundaries in React to gracefully handle errors within components and prevent app crashes.

Logging: Log errors to track issues and troubleshoot problems.

Q93. How do you handle authentication in a MERN application?

Authentication is essential for securing user access:

Choose an authentication method: Select a method like JWT (JSON Web Tokens), session-based authentication, or OAuth.

Implement user registration and login: Allow users to create accounts and log in securely.

Store user data: Securely store user credentials and other information in the database.

Verify user identity: Validate user credentials and issue authentication tokens or session data.

Protect routes: Restrict access to specific routes based on user authentication status.

Q94. What do you mean by Asynchronous API?

An asynchronous API (Application Programming Interface) allows certain operations to be performed asynchronously. In the context of web development, this often involves making requests to servers or executing code without blocking the main thread. Asynchronous APIs use mechanisms like callbacks, promises, or async/await to handle asynchronous operations.

Q95. What is Callback Hell and What is the main cause of it?

Callback Hell, also known as “Pyramid of Doom,” refers to the situation where multiple nested callbacks make the code difficult to read and maintain. It occurs when handling asynchronous operations using callbacks within callbacks within callbacks, creating a deep and indented structure. The main cause is the nature of JavaScript’s callback-based approach for handling asynchronous tasks, which can lead to deeply nested and hard-to-follow code.

Q96. Explain the MVC architecture.

MVC architecture is a standard architectural way to design apps that distribute one functionality across several components. There are three main components of this:

- **Model:** It is the part of the application that is responsible for data and business rules.
- **View:** This is the part that focuses on how the user interacts with the application through its interface.
- **Controller:** A mediator between the Model and View that processes user input and performs changes in the Model and the View as needed.

Q97. Does MongoDB support foreign key constraints?

MongoDB does not natively support foreign key constraints like traditional relational databases. However, it allows for data modelling with embedded documents or manual reference checks to maintain relationships between collections.

Q98. How does Node prevent blocking code?

Node.js prevents blocking code by using an event-based approach and non-blocking I/O. Tasks are executed asynchronously with the use of callbacks, events and promises so that the main thread is not stalled while waiting for IO operations such as files or databases to respond.

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

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) throw err;

console.log(data);

});

console.log('File read initiated'); // This line runs before file reading completes
```

Q99. How does Node.js handle child threads?

Node.js is inherently single-threaded although it does manage child threads via the use of the child_process module. In this way, Node.js can create other processes in order to complete a task in parallel with the main thread instead of making an operation wait. Child processes can communicate with the parent process via messaging.

Code Example:

```
const { fork } = require('child_process');

const child = fork('child.js');

child.on('message', (message) => {

console.log('Message from child:', message);

});

child.send({ hello: 'world' });
```

Q100. If Node.js is single-threaded, then how does it handle concurrency?

Concurrency in Node.js can be achieved using its event loop and asynchronous I/o. So while classical JavaScript is single-threaded, Node.js sends all in/ out requests (such as HTTP or file requests) to a kernel thread, which can process all of these queries at the same time. When the requests finish, instead of blocking the main thread to fetch data, Node uses callbacks or promises.

Q101. What is aggregation in MongoDB?

Aggregation in MongoDB is the processing of large volumes of documents in a certain way that documents are presented in an ordered manner passing through stages of filtering, grouping, sorting and other computations. The aggregation framework is exactly like a GROUP BY clause in SQL except that there are no limitations.

Q102. How do synchronous and asynchronous code execution differ in JavaScript?

- **Synchronous:** This is a step-by-step code executed whereby each step cannot progress until the previous one is fully completed. This may cause blocking in situations where a particular line of code takes an extended period.
- **Asynchronous:** In this, no waiting occurs for an incomplete task. Further action is taken to the next task. After that, once the asynchronous action is done a callback, promise, or async/await is employed to take care of the output.

Q103. What are the differences between GET and POST methods in HTTP?

Feature	**GET**	**POST**
Data Transmission	Appends data to the URL (query string)	Sends data in the request body

Security	Less secure (data visible in URL)	More secure (data not visible in URL)
Data Size	Limited by URL length	No size limitations for data
Use Case	Retrieving data	Submitting data (e.g., forms)
Idempotence	Idempotent (multiple requests = same result)	Non-idempotent (multiple requests can have different effects)

Q104. What is sharding in MongoDB?

A shard in MongoDB is a logical ‘chunk’ of a dataset that has been split for distribution across various servers. Sharding is horizontal scaling by dividing a large dataset into smaller fragments or shards so that they become more manageable. Each shard is situated on a different server therefore performance is enhanced and more data and traffic can be accommodated in the database.

Q105. What is a stream and what types of streams are available in Node.js?

In Node.js, a stream is an abstract interface that encompasses classes such as, Writable stream, Readable stream, Transform stream, Duplex stream, etc, that deal with the input and output of data through chunks rather than in a single scenario which may take longer to process data. Data streams are useful to prevent excessive memory consumption during the reading and writing of a file.

Types of streams include:

- **Readable streams:** Used to read data, e.g., fs.createReadStream().
- **Writable streams:** Used to write data, e.g., fs.createWriteStream().
- **Duplex streams:** Both readable and writable, e.g., TCP sockets.

Q106. How do promises differ from callbacks in JavaScript?

- **Callbacks:** These are ordinary functions which take other functions as their arguments, and are executed once the required operation is accomplished.
- **Promises:** These are the objects which are used to complete or fail an asynchronous operation. Promises are preferred over callbacks since they can be easily chained together leading to a reduction in complexity and an increase in code comprehension.

Q107. What is blocking code?

Blocking code refers to code that stops the execution of further instructions until the current task is completed. In a synchronous process, blocking code can halt the entire flow, making the system less responsive. Node.js avoids blocking code by using non-blocking, asynchronous methods.

Q108. Can you elaborate on the MongoDB aggregation pipeline?

The MongoDB aggregation pipeline is a framework for data aggregation, which processes data in a series of stages. Each stage transforms the documents, and the output of one stage is passed as input to the next. Common stages include:

- **$match:** Filters documents to pass only those that match the criteria.
- **$group:** Groups documents by a specified key and performs aggregate functions.
- **$project:** Reshapes each document by including, excluding, or adding fields.
- **$sort:** Orders the documents based on specified fields.
- **$limit:** Limits the number of documents to pass to the next stage.

Q109. Compare Node.js and traditional web servers like Apache.

Feature	**Node.js**	**Apache**
Language	JavaScript	C, Perl, PHP, etc.
Concurrency	Non-blocking, event-driven	Thread-based
Performance	High for I/O-bound tasks	High for CPU-bound tasks
Flexibility	Highly flexible, full-stack development	More rigid, primarily serves static content
Use Case	Real-time applications, APIs	Static content, traditional websites

Q110. Could you tell us what Mongoose is?

Mongoose is a library dealing with object data modelling (ODM) in node.js based systems and was specifically made for the use of the MongoDB database. It provides a solution/schema to your application data and includes a built-in validator, query builder and middleware. With the use of Mongoose, we can also work with the structure of MongoDB by creating schemas and models for MongoDB collections.

Q111. Compare the pros and cons of using MongoDB vs. Firebase for a project.

Feature	**MongoDB**	**Firebase**
Data Model	Document-based (JSON-like)	Realtime NoSQL database
Scalability	High scalability (sharding)	Scalable with auto-syncing capabilities
Offline	Not built-in, requires	Built-in offline data persistence

Support	additional setup	
Flexibility	Highly flexible, schema-less design	Less flexible, designed for real-time apps
Querying	Powerful querying and aggregation	Limited querying capabilities
Use Case	Complex queries, large datasets	Realtime features, mobile apps
Learning Curve	Requires more setup and knowledge	Easier to start with, especially for beginners

Q112. What are the key differences between front-end and back-end development?

Aspect	**Front-end Development**	**Back-end Development**
Focus	User interface and user experience	Server-side logic, database management
Languages	HTML, CSS, JavaScript, React, Angular	Node.js, Python, Java, Ruby, PHP
Frameworks	React, Angular, Vue.js	Express.js, Django, Spring
Tools	Webpack, Sass, Bootstrap	Databases (SQL, NoSQL), Server management
Interaction	Directly interacts with users	Handles data storage, business

		logic
Performance	Focused on load times, responsiveness	Focused on server speed, database queries

Q113. What is a document in MongoDB?

A document in MongoDB is a record within a MongoDB collection, comparable to the table row in the relational database. It is a type of structure that is made of fields and their corresponding values; the fields are more like the columns in a table.

Example:

```
{
"_id": ObjectId("507f1f77bcf86cd799439011"),
"name": "John Doe",
"age": 30,
"email": "john.doe@example.com"
}
```

Q114. What is the Mongo shell?

The Mongo command line interface that allows a user to interact with the Mongo internals and build, manage, and retrieve information from MongoDB databases is known as the Mongo shell. It helps you manage your database, run queries, do admin work and check commands for MongoDB. The Mongo shell is a software environment commonly used by developers and database administrators to run database tasks from a command window.

Example:

```
$ mongo > use myDatabase > db.users.find({ name: “John Doe” })
```

Q115. Explain the term “indexing” in MongoDB.

The creation of indexes in MongoDB is a technique used to optimise the search processes over a collection. It is a specialised structure which contains some parts of the data, and it is easy to navigate. MongoDB uses indexes on its fields/texts for finding the necessary documents without scanning through the entire collection, thus enhancing the query performance.

Common Index Types:

- **Single field index:** Indexes a single field.
- **Compound index:** Indexes multiple fields within a document.
- **Text index:** Supports text search queries on string content.
- **Geospatial index:** Supports location-based queries.

Q116. How do you delete a document in MongoDB?

If you want to delete one or a group of documents from the collection in MongoDB, you can apply methods deleteOne() or deleteMany() to a particular collection. deleteOne() deletes one document satisfying the requirements, on the contrary deleteMany() gets rid of all satisfying requirements documents.

Example:

```
// Delete one document

db.users.deleteOne({ name: "John Doe" });

// Delete multiple documents

db.users.deleteMany({ age: { $lt: 18 } });
```

Q117. What are the differences between a web server and an application server?

Feature	Web Server	Application Server

Primary Function	Serves static content, such as HTML, CSS, and images	Executes business logic, serves dynamic content
Processing Capability	Handles HTTP requests and responses	Manages application logic, database interactions
Examples	Apache, Nginx	Tomcat, WebSphere, JBoss
Languages Supported	HTML, CSS, JavaScript	Java, .NET, PHP, Python, etc.
Interaction	Directly interacts with the client's browser	Interacts with databases, APIs, and web servers
Performance	Optimised for serving static files	Optimised for complex transactions and logic processing

Q118. What is a replica set in MongoDB?

The primary function of a replica set in MongoDB is to ensure redundancy and high uptime through a set of MongoDB servers that hold identical data. There is a primary node and secondary nodes in each replica set. The primary node is responsible for every write operation on the replicas as the secondary nodes only replicate data from the primary server. If the primary node fails, an automatic election process selects a new primary, ensuring minimal downtime.

Key Features:

- **Primary Node:** Handles write operations and replicate data to secondaries.
- **Secondary Nodes:** Replicate data from the primary and can serve read operations.
- **Arbiter:** Participates in elections but does not store data.

Q119. How can I authenticate users in Express?

User authentication in Express can be implemented using middleware to handle user sessions, tokens, or credentials. Common methods include:
Session-based authentication**:** Storing user sessions on the server using libraries like express-session.

- **Token-based authentication:** Using JSON Web Tokens (JWT) to verify user identity. The token is sent with each request and verified by the server.
- **OAuth:** Implementing third-party authentication via providers like Google or Facebook using passport.js.

Q120. Which template engines does Express support?

Express supports various template engines that allow you to generate HTML pages dynamically. Some popular template engines include:

- **Pug (formerly Jade):** A high-performance template engine with a concise syntax.
- **EJS (Embedded JavaScript):** A simple template engine that lets you embed JavaScript in your HTML.
- **Handlebars:** An extension of the Mustache template engine with more powerful features.

Q121. What function arguments are available to Express.js route handlers?

The route handler of Express.js has three main arguments.

- **req (request):** This is the request object which contains all the information of the request that was made such as headers, URLs etc.
- **res (response):** This refers to the response content that will be sent to the client (e.g. status, JSON data, HTML content).
- **next (next middleware):** This is a function that is called to pass control over to the next middleware or the route handler on the stack. This is used when multiple functions have to be called when handling one

request or when multiple functions handle one request, especially in the case of errors.

Q122. How do I render plain HTML in Express?

To render plain HTML in Express, you can use the res.sendFile() method to send an HTML file from your server to the client, or you can use res.send() to send HTML as a string.

Example:

```
// Serve HTML as a string

app.get('/about', (req, res) => {

res.send('<h1>About Page</h1><p>This is a simple Express app.</p>');

});
```

Q123. How to avoid callback hell in Node.js?

Callback hell occurs when multiple nested callbacks make code difficult to read and maintain. You can avoid callback hell in Node.js by using:

- **Promises:** Allows chaining of asynchronous operations.
- **Async/Await:** Provides a more synchronous-looking flow for asynchronous code.
- **Modularization:** Break down complex code into smaller, reusable functions or modules.

Example:

```
// Using Promises

function asyncOperation() {

return new Promise((resolve, reject) => {

setTimeout(() => resolve('Done'), 1000);

});}
```

```
asyncOperation().then(result => {
console.log(result);
}).catch(error => {
console.error(error);
});
// Using Async/Await
async function doAsyncWork() {
try {
const result = await asyncOperation();
console.log(result);
} catch (error) {
console.error(error);
}
}
return re.test(email);
};
const handleSubmit = (e) => {
e.preventDefault();
if (!validateEmail(email)) {
setError('Invalid email');
} else {
setError('');
// Submit form

}

};

return (

<form onSubmit={handleSubmit}>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)}/>

{error && <p>{error}</p>}

<button type="submit">Submit</button>

</form>

);

}
```

Q124. What are the differences between HTTP and HTTPS?

Feature	HTTP	HTTPS
Security	Unencrypted, vulnerable to attacks	Encrypted using SSL/TLS
Port	Uses port 80 by default	Uses port 443 by default
Data Integrity	Data can be tampered with during transit	Data is secure from tampering
Certificate	No certificates required	Requires an SSL/TLS certificate

Use Case	Non-sensitive data transfer	Sensitive data transfer (e.g., payments)

Q125. What is REPL in Node.Js?

Ans: REPL stands for Reading Eval Print Loop. It also evaluates datasets and displays the results. It is essential to understand that REPL replicates the environment of a Linux shell. REPL is related to the successful completion of the tasks listed below.

- **Print:** Print links with printing the results derived from the command.
- **Read** The read command links with reading the inputs provided by the user. It stores the structure inside the function.
- **Loop:** Loop would continue to twist the command until the user uses Ctrl+C.

Q126. Define BSON in MongoDB.

Ans: BSON is a binary serialization format that MongoDB uses to store documents and make remote procedure calls. BSON expands the JSON format to provide more data types, ordered fields, and fast encoding and decoding across languages.

Q127 .What is the purpose of ObjectId in MongoDB?

ObjectId is the default unique identifier for MongoDB documents, containing information like timestamp and machine ID.

Q128. Explain replication in MongoDB.

Replication ensures high availability and redundancy by copying data across multiple servers.

Q129. What is the Aggregation Framework in MongoDB?

A feature for advanced data processing, like grouping, filtering, and sorting.

Q130. What is req and res in Express.js?

req is the request object, and res is the response object in an HTTP request.

Q131. What is next() in Express.js middleware?

It passes control to the next middleware in the stack.

Q132. What are the key features of Node.js?

Non-blocking I/O, event-driven architecture, and single-threaded execution.

Q133. What is the Event Loop in Node.js?

A mechanism to handle asynchronous operations in a single-threaded environment.

Q134. What is the difference between require and import in Node.js?

require is CommonJS, and import is ES6 syntax.

Q135. What is a callback in Node.js?

A function passed as an argument to another function to handle asynchronous tasks.

Q140. If Node.js is single-threaded, then how does it handle concurrency?

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

Q141. What is the purpose of the module .Exports?

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

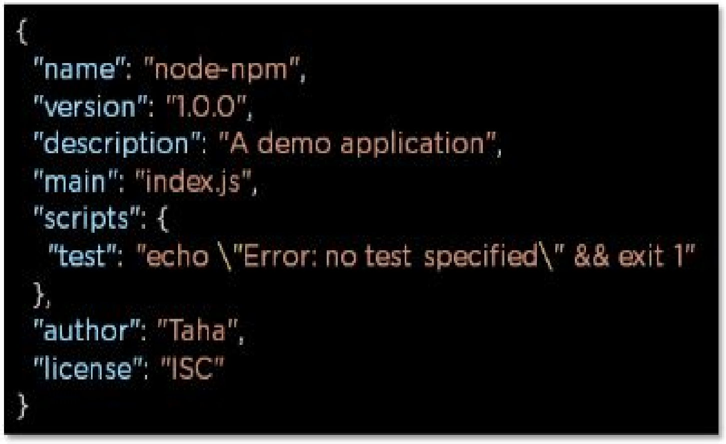
Q142. What is the command used to import external libraries?

The “require” command is used for importing external libraries. For example - “var http=require (“HTTP”).” This will load the HTTP library and the single exported object through the HTTP variable.

Q143. What is the package.json file?

The package.json file is the heart of a Node.js system. This file holds the metadata for a particular project. The package.json file is found in the root directory of any Node application or module.This is what a package.json file looks like immediately after creating a Node.js project using the command: npm init

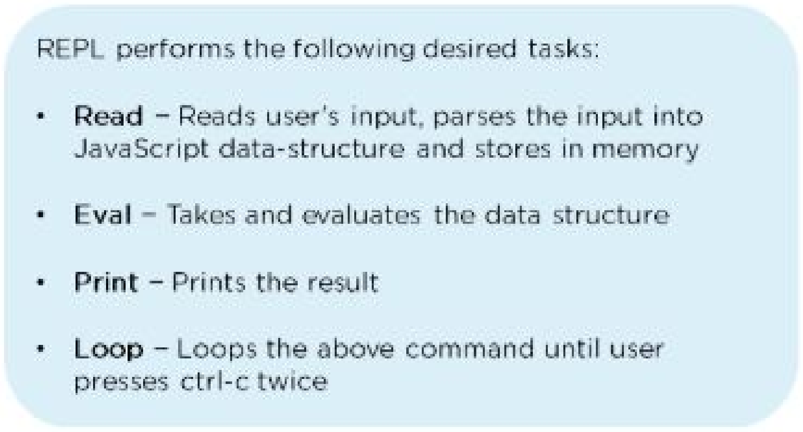
You can edit the parameters when you create a Node.js project.



```
{
  "name": "node-npm",
  "version": "1.0.0",
  "description": "A demo application",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Taha",
  "license": "ISC"
}
```

Q144. What is REPL in Node.js?

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



REPL performs the following desired tasks:

- **Read** – Reads user's input, parses the input into JavaScript data-structure and stores in memory
- **Eval** – Takes and evaluates the data structure
- **Print** – Prints the result
- **Loop** – Loops the above command until user presses ctrl-c twice

Q145. How can we use async await in node.js?

To use async/await in Node.js, you'll need to use functions that return promises. You can then use the async keyword to mark a function as asynchronous and the await keyword to wait for a promise to resolve before continuing with the rest of the code.

Q146. What is a passport in Node.js?

Passport is a popular authentication middleware for Node.js. It provides a simple and modular way to implement authentication in Node.js applications. Passport supports many authentication mechanisms, including username/password, social logins like Facebook and Google, and JSON Web Tokens (JWTs).

Q147. How to manage sessions in Node.js?

Session management can be done in node.js by using the express-session module. It helps in saving the data in the key-value form. In this module, the session data is not saved in the cookie itself, just the session ID.

Q148. Explain the packages used for file uploading in Node.js?

The package used for file uploading in Node.js is Multer. The file can be uploaded to the server using this module. There are other modules in the market but Multer is very popular when it comes to file uploading. Multer is a node.js middleware that is used for handling multipart/form-data, which is a mostly used library for uploading files.

Q149. What is an Index in MongoDB, and How to Create One?

An index in MongoDB is a data structure that improves the speed of data retrieval operations on a collection. You can create an index using the createIndex method. For example, to create an index on the name field:

db.collection.createIndex({ name: 1 })

Q150. Describe the Aggregation Framework in MongoDB

The Aggregation Framework in MongoDB is a powerful tool for performing data processing and transformation on documents within a collection. It works by passing documents through a multi-stage pipeline, where each stage performs a specific operation on the data, such as filtering, grouping, sorting, reshaping and computing aggregations.
This framework is particularly useful for creating complex data transformations and analytics directly within the database.

Q151. How to Perform Aggregation Operations Using MongoDB?

Aggregation operations in MongoDB are performed using the aggregate method. This method takes an array of pipeline stages, each stage representing a step in the data processing pipeline. For example, to calculate the total sales for each product, you might use the following aggregation pipeline:

```
db.sales.aggregate([ { $match: { status: "completed" } },  // Filter completed sales
 { $group:{ _id: "$product", totalSales:{$sum: "$amount" }}},//Group by product and sum
the sales amount
  { $sort: { totalSales: -1 } }  // Sort by total sales in descending order
])
```

Q153. What is MongoDB Atlas, and How does it Differ From Self-Hosted MongoDB?

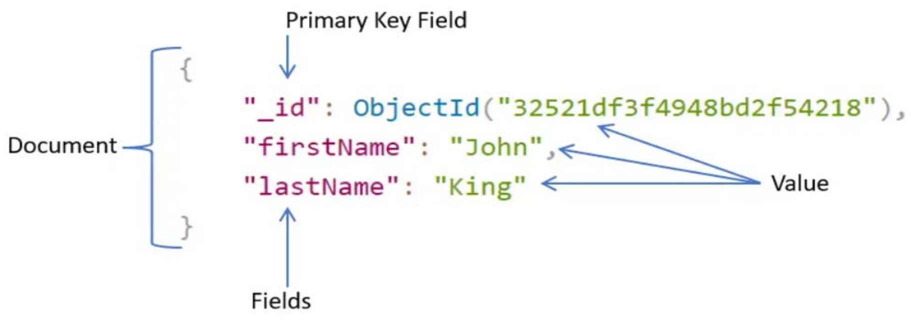
MongoDB Atlas is a fully managed cloud database service provided by MongoDB. It offers automated deployment, scaling, and management of MongoDB clusters across various cloud providers (AWS, Azure, Google Cloud). Key differences from self-hosted MongoDB include:

- **Managed Service**: Atlas handles infrastructure management, backups, monitoring, and upgrades.
- **Scalability**: Easily scale clusters up or down based on demand.
- **Security**: Built-in security features such as encryption, access controls, and compliance certifications.
- **Global Distribution**: Deploy clusters across multiple regions for low-latency access and high availability.
- **Integrations**: Seamless integration with other cloud services and MongoDB tools.

Mongo DB Theory

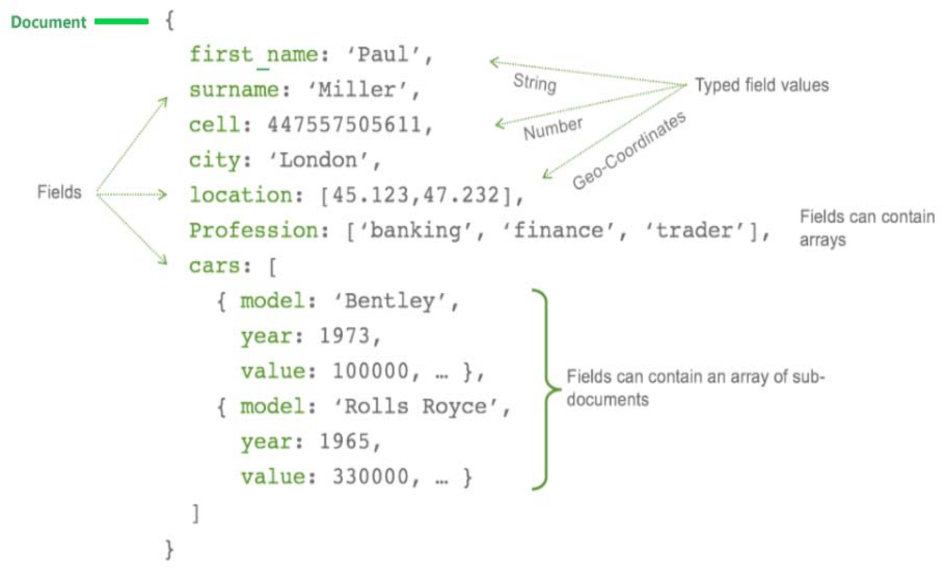
#1. Introduction to MongoDB

- ✓ MongoDB is an open source NoSQL(Not only SQL), document-oriented database designed with both scalability and developer agility in mind.
- ✓ Instead of storing your data in tables and rows as you would with a relational database, in MongoDB you store JSON-like documents with dynamic schemas (schema-free, schema less).
- ✓ MongoDB offers support for many programming languages, such as C, C++, C#, Go, Java, Kotlin, Node.js, PHP, Python, Ruby, Rust, Scala, and Swift.
- ✓ Instead of using tables and rows as in relational databases, as a NoSQL database, the MongoDB architecture is made up of collections and documents. Documents are made up of Key-value pairs -- MongoDB's basic unit of data. **Some Object Oriented DBMS are (MongoDB, DynamoDB and CosmosDB)**



```
{
    "_id": ObjectId("32521df3f4948bd2f54218"),
    "firstName": "John",
    "lastName": "King"
}
```

#2. MongoDB Document Structure

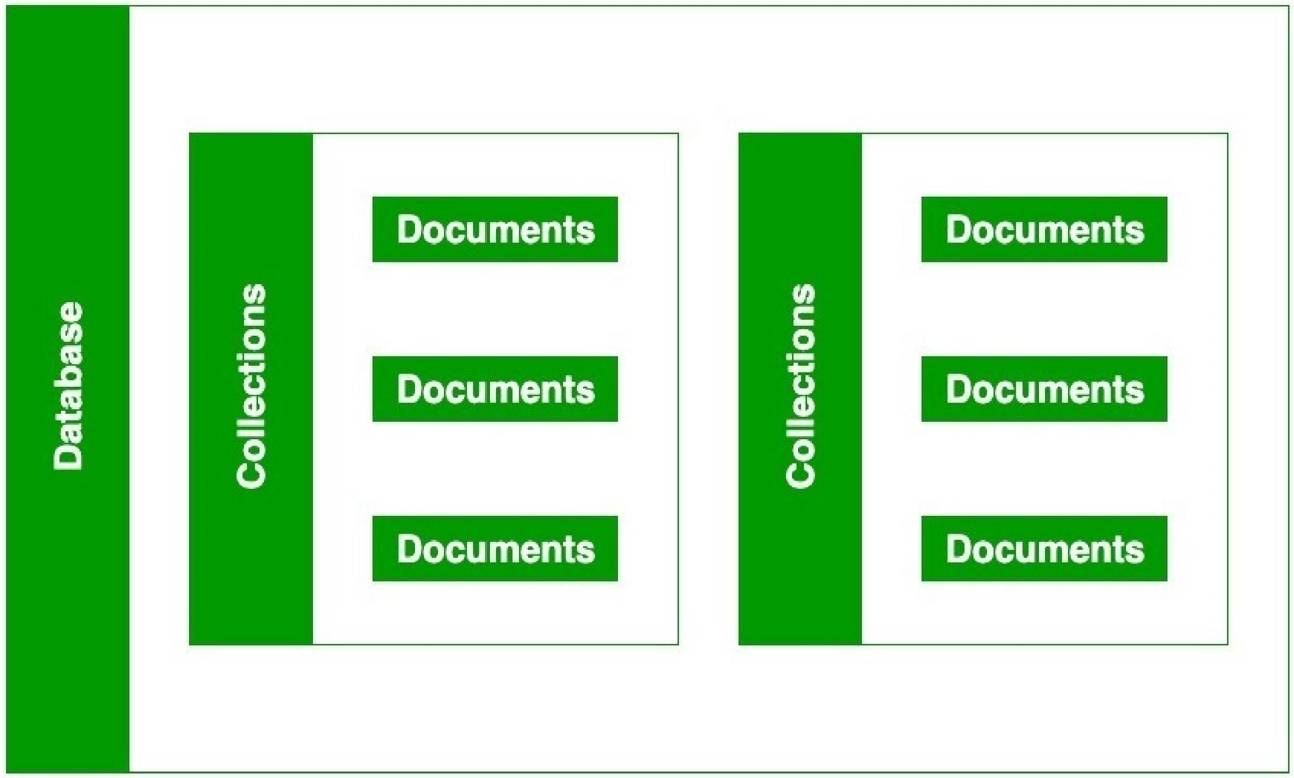

Document
Fields
String
Number
Geo-Coordinates
Typed field values
Fields can contain arrays
Fields can contain an array of sub-documents

```
{
    first_name: 'Paul',
    surname: 'Miller',
    cell: 447557505611,
    city: 'London',
    location: [45.123,47.232],
    Profession: ['banking', 'finance', 'trader'],
    cars: [
      { model: 'Bentley',
        year: 1973,
        value: 100000, … },
      { model: 'Rolls Royce',
        year: 1965,
        value: 330000, … }
    ]
}
```

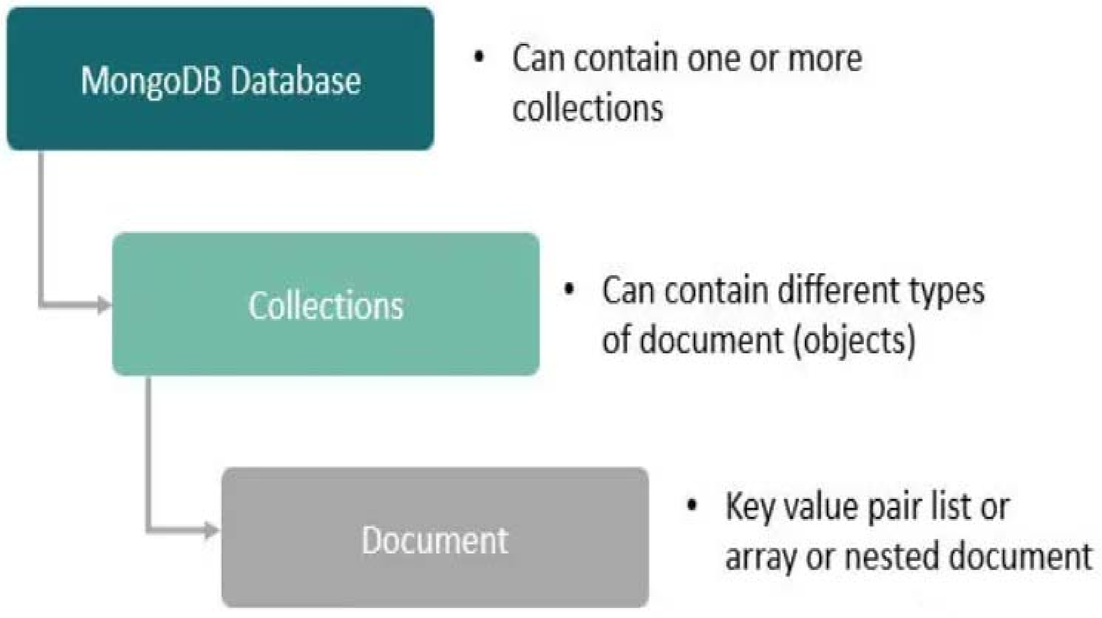
#3. Database-Collections-Documents

- **Database** is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.
- **Collection** is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.
- **A document** is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

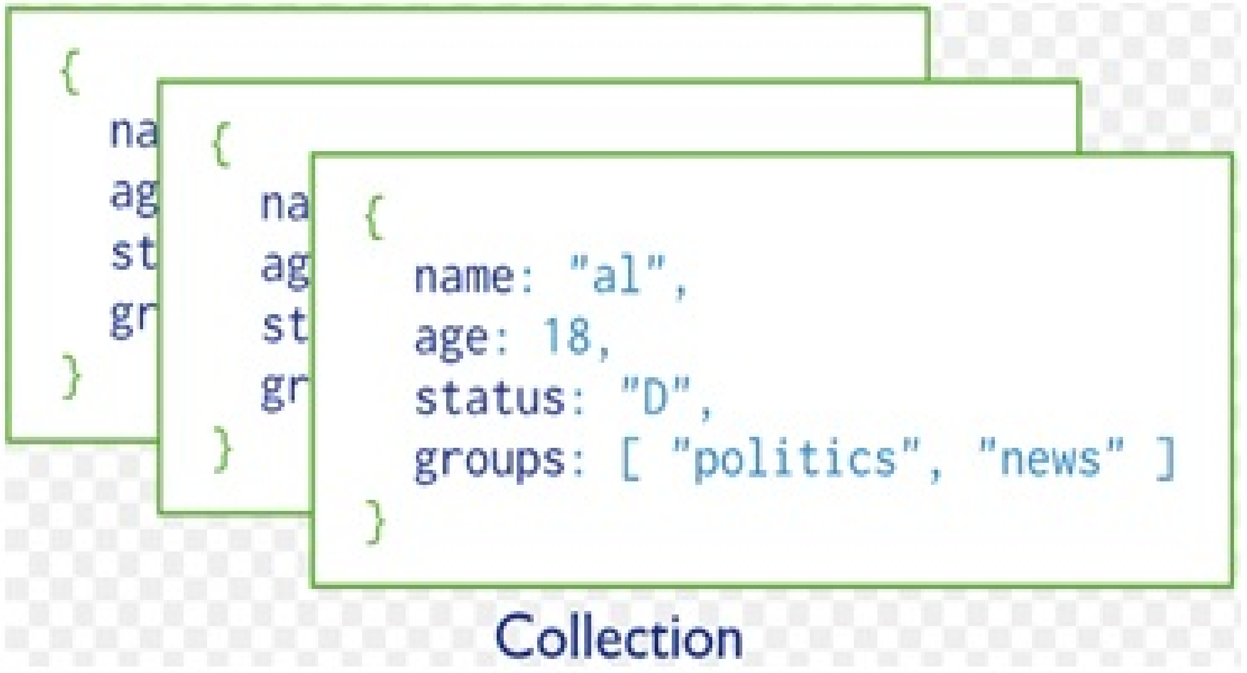
#4. Database-Colletions-Documents



#5. Concepts of collections and documents in MongoDB.


MongoDB Database
• Can contain one or more collections
Collections
• Can contain different types of document (objects)
Document
• Key value pair list or array or nested document

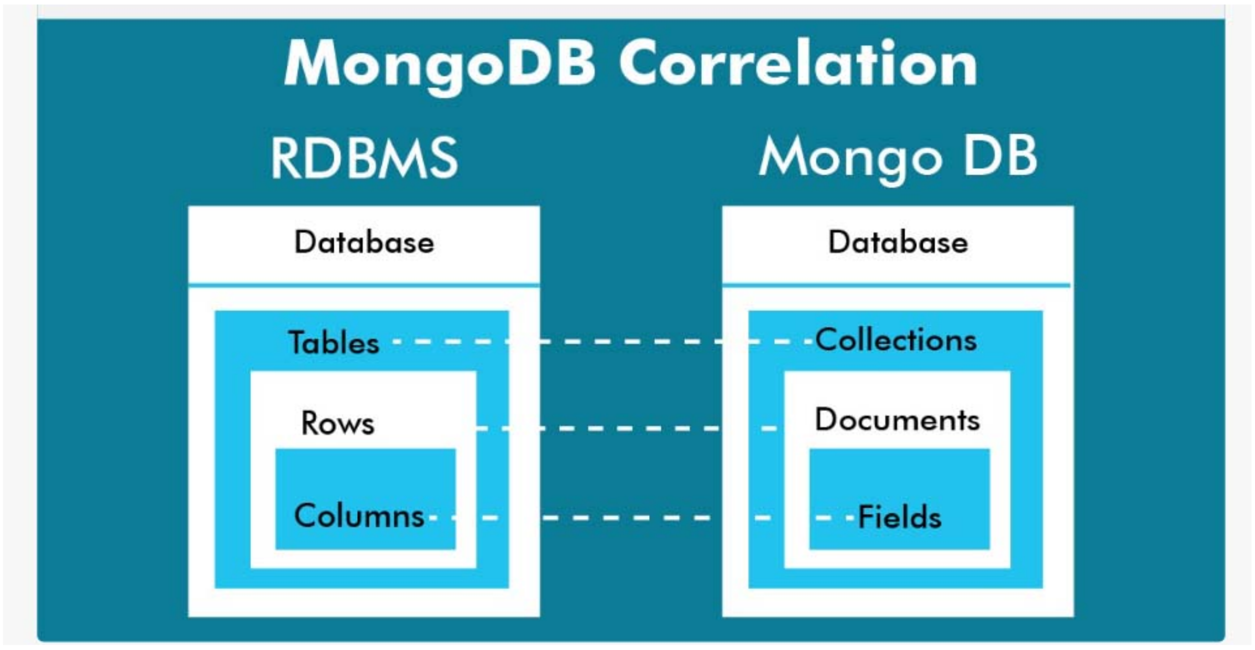
#6. A COLLECTION IN MONGODB having 3 DOCUMENTS


```
{
   name: "al",
   age: 18,
   status: "D",
   groups: [ "politics", "news" ]
}
```

#7. Relationship of RDBMS Terminology with MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)

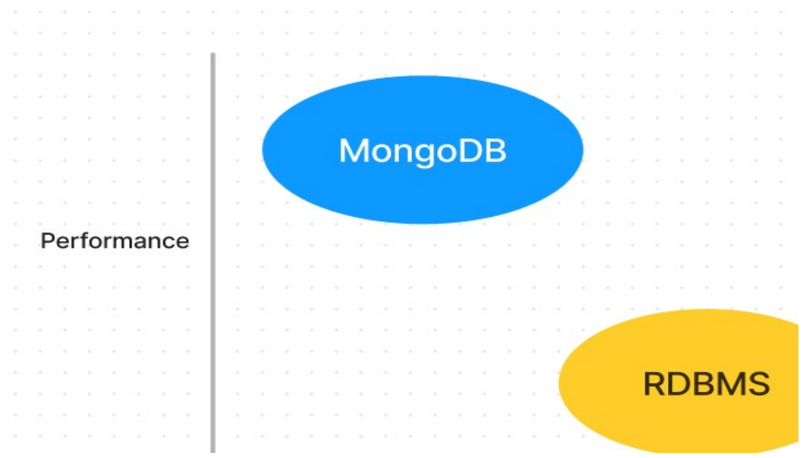
If you compare it with an RDBMS, documents correspond to the record (row) which have a unique identifier ID (primary key), fields/values are columns that have an associated key to them, and collections are the equivalent of a table.



#8. RDBMS and MongoDB Difference

RDBMS	MongoDB
It is a relational database.	It is a non-relational and document-oriented database.
Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
It is vertically scalable i.e increasing RAM.	It is horizontally scalable i.e we can add more servers.
It has a predefined schema.	It has a dynamic schema.
It is quite vulnerable to SQL injection.	It is not affected by SQL injection.
It is row-based.	It is document-based.
It is slower in comparison with MongoDB.	It is almost 100 times faster than RDBMS.
Supports complex joins.	No support for complex joins.
It does not provide JavaScript client for querying.	It provides a JavaScript client for querying.
It supports SQL query language only.	It supports JSON query language (JAQL).

#9. The trade-off between RDBMS and MongoDB:



#10. Features of MongoDB –

- **Schema-less Database:** It is the great feature provided by the MongoDB. A Schema-less database means one collection can hold different types of documents in it.
- **Replication:** MongoDB provides high availability and redundancy with the help of replication, it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.
- **Aggregation:** It allows to perform operations on the grouped data and get a single result or computed result. It is similar to the SQL GROUPBY clause
- **High Performance:** The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.
- **Document-Oriented**: MongoDB stores data in flexible, JSON-like documents called BSON (Binary JSON), which allows for easy storage and retrieval of complex data structures.
- **Scalability**: MongoDB is designed to scale horizontally across multiple servers, making it suitable for handling large volumes of data and high-traffic applications.
- **Rich Query Language**: MongoDB supports a powerful query language that allows for complex queries, aggregations, and data manipulation operations.
- **Horizontal Scaling**: MongoDB supports horizontal scaling through sharding, allowing data to be distributed across multiple servers to handle increased load and storage requirements.

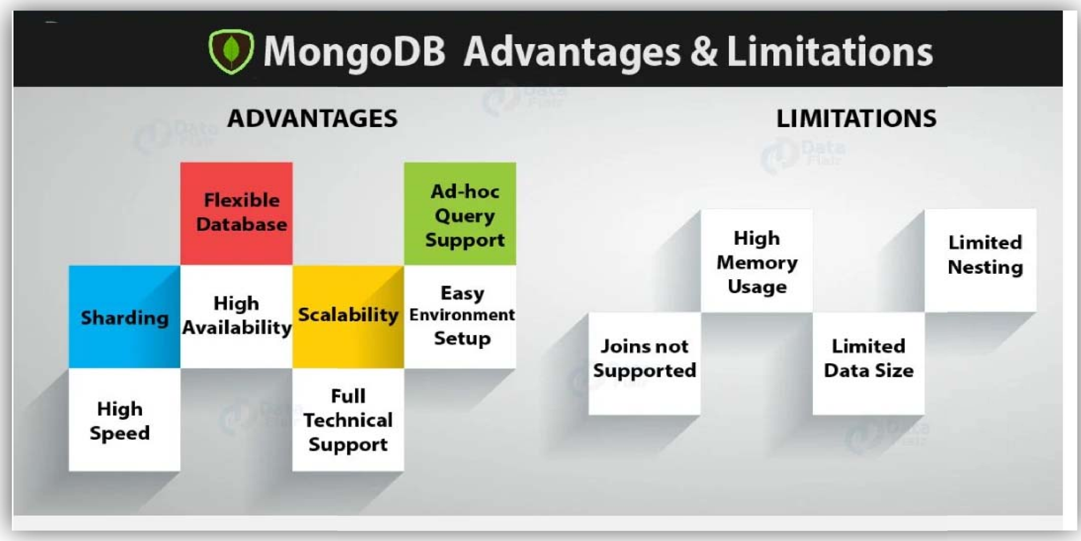
#11. Advantages of MongoDB

- MongoDB stores data as JSON based document that does not enforce the schema. It allows us to store hierarchical data in a document. This makes it easy to store and retrieve data in an efficient manner.
- It is easy to scale up or down as per the requirement since it is a document based database. MongoDB also allows us to split data across multiple servers.
- MongoDB provides rich features like indexing, aggregation, file store, etc.
- MongoDB performs fast with huge data.
- MongoDB provides drivers to store and fetch data from different applications developed in different technologies such as C#, Java, Python, Node.js, etc.
- MongoDB provides tools to manage MongoDB databases.

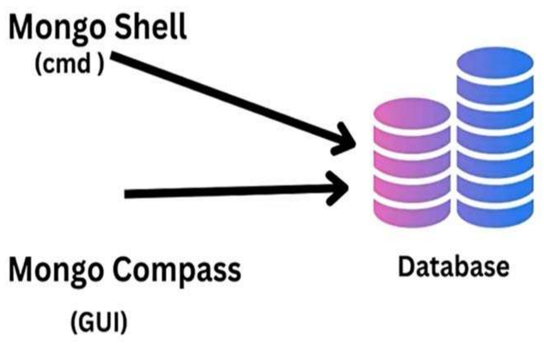
#12. Disadvantages of MongoDB

- **1) Joins not Supported**
  MongoDB **doesn't support joins** like a relational database. Yet one can use joins functionality by adding by coding it manually. But it may slow execution and affect performance.
- **2) High Memory Usage**
  MongoDB stores key names for each value pair. Also, due to no functionality of joins, **there is data redundancy**. This results in increasing **unnecessary usage of memory.**
- **3) Duplicates**
  Another one of the major limitations of MongoDB is the **duplication of data**. The limitation makes it difficult to handle data sets as the relations are not defined well.
- **4) Limited Data Size**
  You can have a document size, not more than **16MB.**
- **5) Limited Nesting**
  You cannot perform the nesting of documents for more than **100 levels.**

MongoDB Advantages & Limitations Diagram:



#13. MongoDB Tool



#14. MongoDB Compass-MongoDB Shell

MONGODB COMPASS

MongoDB Compass is a GUI based tools to interact with local or remote MongoDB server and databases. Use Compass to visually explore your data, run ad hoc queries, perform CRUD operations, and view and optimize your query performance. It can be installed on Linux, Mac, or Windows.
When we explore exploring our data in the visual environment, we can use Compass GUI to optimize performance, manage indexes, and implement document-validation.

MONGODB SHELL

MongoDB Shell is the quickest way to connect, configure, query, and work with your MongoDB database. It acts as a command-line client of the MongoDB server. Shell is a command-line interface provided by MongoDB to interact with MongoDB instances. It serves as a powerful tool for performing various database operations, such as querying, updating, and managing databases, collections, and documents.

#15. MongoDB and Mongo Shell version

- To check the mongoDB version:
  mongod   --version
- To check the mongo Shell version:
  mongosh --version

MogoDB Commands (JQL)

# MongoDB - Create Database

The use Command

#Syntax
Basic syntax of **use DATABASE** statement is as follows-

- **use DATABASE_NAME**

#Example
If you want to use a database with name **<mydb>**, then **use DATABASE** statement would be as follows –

- **use mydb**

To check your currently selected database, use the command **db**

- **db**

Mydb

Note: Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

- **>db.movie.insert({"name":"Cybrom tutorial", "rating": 10})**

- **>show dbs**

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

# MongoDB - Drop Database

- **Syntax**

Basic syntax of **dropDatabase()** command is as follows

- **db.dropDatabase()**

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

# MongoDB - Create Collection

- **The createCollection() Method**
- **db.createCollection(name, options)**
- Basic syntax of **createCollection()** method without options is as follows –
- >use test switched to db test

   - **>db.createCollection("mycollection")**

You can check the created collection by using the command **show collections**.

   - **>show collections**

Note :- **In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.**

   - **>db.students.insert({"name" : “Sanjay Sharma"}),**

# MongoDB - Drop Collection

Basic syntax of **drop()** command is as follows –

   - **db.COLLECTION_NAME.drop()**

Example

First, check the available collections into your database mydb.

   - **>use mydb**

Now drop the collection with the name **mycollection**.

   - **>db.mycollection.drop()**

# MongoDB - Insert Document

The insert() method

The basic syntax of **insert()** command is as follows –

   - **>db.COLLECTION_NAME.insert(document)**

The insertOne() method

The basic syntax of insertOne() command is as follows –

- **>db.COLLECTION_NAME.insertOne(document)**

The insertMany() method
You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

- **> db.empDetails.insertMany([{},{},{}])**

# MongoDB - Query Document

The basic syntax of **find()** method is as follows –

- **>db.COLLECTION_NAME.find()**

find() method will display all the documents in a non-structured way.

To display the results in a formatted way, you can use pretty() method.

- **>db.COLLECTION_NAME.find().pretty()**

Apart from the find() method, there is findOne() method, that returns only one document.

- **>db.COLLECTIONNAME.findOne()**

# AND & OR in MongoDB

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND –

- **>db.mycol.find({ $and: [{<key1>:<value1>}, { <key2>:<value2>}] })**
- **> db.mycol.find({$and:[{“city":“Bhopal"},{“grade”: “B"}]}).pretty()**

OR in MongoDB:

- **>db.mycol.find({ $or: [{key1: value1}, {key2:value2}] }).pretty()**
- **>db.mycol.find({$or:[{“city":“Bhopal"},{“grade": “A"}]}).pretty()**
- **Using AND and OR Together**
- **>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"}, {"title": "MongoDB Overview"}]}).pretty()**

# NOR & NOT in MongoDB

NOR :- This operator is used to perform logical NOR operation on the array of one or more expressions and select or retrieve only those documents that do not match all the given expression in the array

- **Syntax:**
    - **{ $nor: [{ Expression1 }, { Expression2 }, ... { ExpressionN }] }**
- **Exp:**
    - **db.contributor.find({$nor: [{salary: 3000}, {branch: "ECE"}]})**

NOT:- The $not operator can be used with regular expressions. It selects or retrieves only those documents that do not match the given operator expression.

- **Syntax:**
    - **{ field: { $not: { operator-expression } } }**
- **Exp:**
    - **db.student.find({"Total_marks" : {$not: {$lt : 400}}})**

# MongoDB Update Document

To update an existing document we can use the updateOne() or updateMany() methods.

- **updateOne()**

The updateOne() method will update the first document that is found matching the provided query.

- Example
    - **db.posts.updateOne({ title: "Post Title 1" }, { $set: { likes: 2 } })**

updateMany()

The updateMany() method will update all documents that match the provided query.

- **Syntax:**
    - **db.Collection_name.updateMany({Selection_Criteria},{$set:{Update_Data}})**
- **Exp:--**
    - **db.empDetails.updateMany({Age:{ $gt: "25" }}, { $set: { Age: '00'}})**
- Example
- Update likes on all documents by 1. For this we will use the $inc (increment) operator:
    - **db.posts.updateMany({}, { $inc: { likes: 1 } })**

# MongoDB - Delete Document

We can delete documents by using the methods deleteOne() or deleteMany().

- **deleteOne()**

The deleteOne() method will delete the first document that matches the query provided.

- **Example**
  - ➢ **db.posts.deleteOne({ title: "Post Title 5" })**
- **deleteMany()**

The deleteMany() method will delete all documents that match the query provided.

- **Example**
  - ➢ **db.posts.deleteMany({ category: "Technology" })**

# MongoDB – Projection

- Syntax

The basic syntax of **find()** method with projection is as follows –

  - ➢ **>db.COLLECTION_NAME.find({},{KEY:1})**
  - ➢ **>db.mycol.find({},{"title":1,_id:0})**

Please note **_id** field is always displayed while executing **find()** method, if you don't want this field, then you need to set it as 0.

# MongoDB limit() Method

In MongoDB, limit() method is used to limit the fields of document that you want to show. Sometimes, you have a lot of fields in collection of your database and have to retrieve only 1 or 2. In such case, limit() method is used.
The MongoDB limit() method is used with find() method.

- **Syntax:**
  - ➢ **db.COLLECTION_NAME.find().limit(NUMBER)**
- **Exp:**
  - ➢ **db.students.find().limit(1)**

MongoDB skip() method
In MongoDB, skip() method is used to skip the document. It is used with find() and limit() methods.

- **Syntax:**
  - ➢ **db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)**

Execute the following query to retrieve only one document and skip 2 documents.

- **Example**

- **db.cybrom.find().limit(1).skip(2)**

# Create Nested Document using $set

- Insert some document like this
  - **{ _id: ObjectId('662f9e99b9dbd126b646b799'), name: 'raman', age: 20 },**
  - **{ _id: ObjectId('662f9e99b9dbd126b646b79a'), name: 'sanju', age: 12 },**
  - **db.students.updateOne({"name":"sonu"}, {$set:{idCards:{panCard:false, adharCard:true}}});**
  - **db.students.updateMany({}, {$set:{hobbies:["reading", "running"]}});**

# MongoDB sort() method

In MongoDB, sort() method is used to sort the documents in the collection. This method accepts a document containing list of fields along with their sorting order.

- The sorting order is specified as 1 or -1.
- 1 is used for ascending order sorting.
- -1 is used for descending order sorting.
- **Syntax:**
  - **db.COLLECTION_NAME.find().sort({KEY:1})**

Execute the following query to display the documents in descending order.

  - **db.javatpoint.find().sort({"Course":-1})**

# MongoDB Query Operators

Comparison

- **$eq**: Values are equal
- **$ne**: Values are not equal
- **$gt**: Value is greater than another value
- **$gte:** Value is greater than or equal to another value
- **$lt**: Value is less than another value
- **$lte**: Value is less than or equal to another value
- **$in**: Value is matched within an array

- **$nin**: Value is not matched within an array

# Comparison Operators

- **$eq**: Values are equal
  - **db.students.find({city:{$eq:'indore'}})**
- **$ne**: Values are not equal
  - **db.students.find({city:{$ne:"indore"}})**
- **$gt**: Value is greater than another value
  - **db.students.find({fees:{$gt:50000}})**
- **$gte:** Value is greater than or equal to another value
  - **db.students.find({fees:{$gte:50000}})**
- **$lt**: Value is less than another value
  - **db.students.find({fees:{$lt:20000}})**
- **$lte**: Value is less than or equal to another value
  - **db.students.find({fees:{$lte:20000}})**
- **$in**: Value is matched within an array
  - **db.students.find({city:{$in:["satna", "Bhopal"]}});**
- **$nin**: Value is not matched within an array
  - **db.students.find({city:{$nin:["satna", "Bhopal"]}});**

# MongoDB $rename operator

The $rename operator is used to change the name of a field. The new name of the field must be different from the old name of the field.

- Syntax of the $rename operator:
  - **{ $rename : { <field 1> : <new name 1>, <field 2> : <new name 2>, ... } }**
- **Example 1: Renaming a single field**
  - **db.students.update({rollno:120}, {$rename:{city:"address"}});**
- **Example 2: Renaming multiple fields in the documents**
  - **db.employees.updateMany({}, {$rename: {"phone_no": "contact_no"}})**
- **Example 3: Renaming a field in nested documents**
  - **db.employee.update({"employee_name": "Temba"},**
  - **{$rename: {"salary.first_month": "salary.month"}})**

# MongoDB – $inc Operator

The $inc operator is used to increment the value of a given field by a specified amount. This operator accepts both positive and negative values for addition.

- Syntax:
  - ➢ **{ $inc: { field1: amount1, field2: amount2, ... } }**
  - ➢ **db.cats.updateMany({ }, { $inc: { age: 3 } })**
- **Decrement the value of the field using the $inc operator**
  - ➢ **db.cats.updateMany({ }, { $inc: { age: -2 } })**
- **Passing a Missing Nested Field to the inc operator in MongoDB**
  - ➢ **db.cats.updateMany({ },{ $inc: { "moreDetails.cost": 20000 } })**

# MongoDB $mul operator

The $mul is a field update operator that allows you to multiply the value of a field by a specified number.

- **syntax:**
  - ➢ **{ $mul: { <field1>: <number1>, <field2>: <number2>, ... } }**
- **The <field> that you want to update must contain a numeric value**
- **Exp:**
  - ➢ **db.products.updateOne({ _id: 5 }, { $mul: { price: 1.1 } })**
- **Exp:**
  - ➢ **db.Details.update({name: "apple"}, {$mul: {price: 5}})**

# Remove a Field($unset)

In MongoDB, you can use the $unset field update operator to completely remove a field from a document.

  - ➢ **db.dogs.updateMany({ }, { $unset: { type: "" } })**
- **Remove Multiple Fields**

You can specify multiple fields to remove by separating them with a comma.

  - ➢ **db.dogs.updateMany({ }, { $unset: { name: "", weight: "" } })**

Embedded Documents

And suppose we want to remove the awards field from the document.

- **We can do this:**
  - ➢ **db.pets.updateMany({ _id: 1 }, { $unset: { "details.awards": "" } })**

$pull : The $pull operator removes from an existing array all instances of a value or values that match a specified condition.
Remove array value:

- **db.stu1.updateOne({"rollno":126}, {$pull:{"hobbies":"swimming"}});**

# MongoDB Aggregation

MongoDB aggregation operations process the data records/documents and return computed results. It collects values from various documents, groups them, and then performs different types of operations on that grouped data like sum , average , minimum , maximum , etc to return a computed result. It is similar to the aggregate function of SQL .

Sample Data:-

```
[
{pno: 101, pname: 'mouse‘, qty: 5 },
{pno: 102,pname: 'printer‘,qty: 6},
……..]
```

- **db.products.aggregate([{$group:{_id:"$pname", noOfProduct:{$sum:1}}}])**
- **db.products.aggregate([{$group:{_id:"$pname", noOfProQty:{$sum:"$qty"}}}])**
- **db.products.aggregate([{$group:{_id:"$pname", ProductAvg:{$avg:"$qty"}}}])**
- **db.products.aggregate([{$group:{_id:"$pname", ProductMax:{$max:"$qty"}}}])**
- **db.products.aggregate([{$group:{_id:"$pname", ProductMin:{$min:"$qty"}}}])**
- **db.products.aggregate([{$group:{_id:"$pname", ProductFirst:{$first:"$qty"}}}])**
- **db.products.aggregate([{$group:{_id:"$pname", ProductLast:{$last:"$qty"}}}])**
- **db.products.aggregate([{$match:{"pname":"printer"}}, {$group:{"_id":{"city":"$city", "product":"$pname"}, "totalSellingInCity":{$sum:"$qty"}}}])**
