MongoDB (Database)
1. What is MongoDB, and how does it differ from a traditional relational database?
MongoDB is a NoSQL database that stores data in a flexible, JSON-like format called BSON. Unlike traditional RDBMS, MongoDB uses collections and documents instead of tables and rows, allowing for schema-less data.

2. Explain the concept of NoSQL and how MongoDB fits into this category.
NoSQL stands for "Not Only SQL" and refers to databases that don't use relational schemas. MongoDB fits this as it is a document-based database, storing data in schema-less, JSON-like documents.

3. What are the different types of data models supported by MongoDB?
MongoDB primarily supports document-based models. While it doesn't natively support graph or columnar models like other NoSQL databases, it can simulate those through document design.

4. How does MongoDB handle schema-less data, and how does this benefit applications?
MongoDB allows each document to have a different structure. This flexibility supports rapid development and iteration, especially in applications with evolving requirements.

5. What are indexes in MongoDB, and why are they important?
Indexes improve query performance by avoiding full collection scans. They allow MongoDB to search through a small subset of data efficiently.

6. Explain the aggregation framework in MongoDB.
The aggregation framework allows advanced data processing, such as filtering, grouping, sorting, and transforming documents—similar to SQL's GROUP BY, but more powerful.

7. What is the purpose of the ObjectId in MongoDB, and how is it different from other data types?
ObjectId is a 12-byte unique identifier automatically generated for each document. It's optimized for uniqueness and sorting in distributed systems.

8. What are the pros and cons of using MongoDB?
Pros: Flexible schema, scalable, JSON-native, fast for many workloads.
Cons: Potential consistency issues, learning curve, less mature ACID support compared to RDBMS.

9. How would you handle large amounts of data in MongoDB, ensuring optimal performance?
Use sharding for horizontal scaling

Apply indexes to optimize queries

Use replication for availability and failover

Express.js (Backend Framework)
1. What is Express.js, and why would you use it in a MERN stack application?
Express.js is a lightweight Node.js web framework used to build APIs and server-side logic. It's ideal for handling HTTP requests in MERN applications due to its simplicity and flexibility.

2. What is middleware in Express? Can you provide an example?
Middleware functions execute during the request/response cycle. They can modify the request/response or terminate the cycle.

```
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
});
```

3. How does routing work in Express, and how do you manage routes in larger applications?
Routes are managed using methods like app.get(), app.post(), etc. In larger apps, modularize routes using

Express Router.

```
const userRoutes = require('./routes/users');
app.use('/users', userRoutes);
```

4. What is the purpose of the next() function in Express middleware?
next() passes control to the next middleware function. It's essential for chaining middleware and error handling.

5. How do you handle error handling in Express applications?
Use an error-handling middleware with four parameters.

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

6. Explain the concept of RESTful APIs and how to implement them in Express.js.
RESTful APIs use HTTP methods (GET, POST, PUT, DELETE) to manage resources. In Express:

```
app.get('/users', ...);
app.post('/users', ...);
```

7. How do you protect an Express app from common security vulnerabilities?
Use Helmet to set security headers (protects against XSS)
Use csurf for CSRF protection
Sanitize inputs

8. What is CORS, and how do you handle it in Express?
CORS (Cross-Origin Resource Sharing) allows cross-domain API requests.

```
const cors = require('cors');
app.use(cors({ origin: 'http://example.com' }));
```

9. How would you structure the backend of a MERN application to ensure scalability?
Use modular folders: routes, controllers, models, services
Use environment variables
Implement logging, caching, and possibly microservices

10. How would you handle authentication and authorization in an Express app?
Use JWT for authentication, and check user roles/permissions for authorization.

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

app.use((req, res, next) => {
  const token = req.headers['authorization'];
  jwt.verify(token, 'secretkey', (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
});
```

React.js (Frontend Library)
1. What are the key differences between React and other JavaScript frameworks like Angular or Vue?
React is a library focused on UI, using JSX and a virtual DOM. Angular is a full-featured framework; Vue is similar to React but offers more out-of-the-box.

2. Explain the concept of a "virtual DOM" and its advantages.
The virtual DOM is a memory-efficient, in-memory representation of the real DOM. React uses it to minimize DOM updates, improving performance.

3. What are React hooks, and how do they differ from class components?
Hooks like useState and useEffect allow using state and side effects in functional components, replacing the need for class components.

4. What is the significance of useEffect and useState hooks?
useState:

```
const [count, setCount] = useState(0);
```

useEffect:

```
useEffect(() => {
  // side effect like fetching data
}, []);
```

5. How do you manage state in a React application?
useState for local state
Context API or Redux for global state

6. Explain the concept of "lifting state up" in React.
Move state to a common parent when child components need shared state, then pass it down via props.

7. What are props, and how do they differ from state in React?
Props are read-only and passed from parent to child. State is mutable and local to a component.

8. What is Redux, and when would you use it in a React application?
Redux is a state management library with a centralized store. It's useful for large apps where many components need access to shared state.

9. How does React handle component re-renders, and what are some strategies to optimize performance?
React re-renders when props/state change. Optimization strategies:

React.memo
useMemo
useCallback

10. What is JSX, and why is it used in React?
JSX allows writing HTML-like syntax in JavaScript, making UI code more readable and declarative.

Node.js (Runtime Environment)
1. What is Node.js, and why is it a good choice for building scalable web applications?
Node.js is a JavaScript runtime that uses an event-driven, non-blocking I/O model. It's ideal for handling many concurrent requests.

2. Explain the event-driven, non-blocking I/O model in Node.js.
Operations like file or DB access don't block the main thread. Node uses callbacks, Promises, or async/await to handle results asynchronously.

3. What is the role of the require() function in Node.js?
require() is used to import modules or files.

4. How does Node.js handle asynchronous code, and what are some common ways to deal with asynchronous operations?

Node uses:

Callbacks
Promises
async/await

5. What are some ways to manage packages and dependencies in Node.js applications?
Use:

npm
yarn
package.json to track dependencies

6. What is the purpose of the process object in Node.js?
The process object gives access to environment variables, system information, and command-line arguments.

7. What are streams in Node.js, and when would you use them?
Streams allow reading/writing data in chunks (e.g., file I/O, network communication), useful for handling large datasets.

8. How would you manage logging and error tracking in a Node.js application?
Use Winston, Morgan for logging
Use **