End Term Exam Answers: JavaScript, Node.js, Express, Authentication

1. JavaScript Fundamentals

- Difference between let, const, and var:

  let and const are block-scoped, while var is function-scoped.

  let allows reassignment, const does not. var can be redeclared, while let and const cannot be redeclared within the same scope.

- Scopes in JavaScript:

  JavaScript has function scope (for var) and block scope (for let and const).

  Variables defined with let or const are available only within the block they are declared in.

- Asynchronous Programming:

  Promises are used to handle asynchronous operations. async/await provides a more readable syntax for handling promises.

- Arrow Functions:

  Arrow functions provide a shorter syntax and do not have their own 'this' value, instead, they inherit 'this' from the surrounding context.

- Module Import/Export:

  Modules allow code to be split into smaller files. Use 'export' to expose code from a module and 'import' to bring it into other files.

- Debugging in JavaScript:

Debugging involves identifying and fixing issues in the code. It can be done through console.log() statements, breakpoints in the developer tools, or using debuggers.

2. Fundamentals of Node.js

- Node.js:

Node.js is a runtime environment for running JavaScript on the server side. Unlike traditional server environments, Node.js is non-blocking and asynchronous.

- npm:

npm is the Node Package Manager, used to install and manage packages in Node.js projects.

- fs module:

The fs module in Node.js provides functions to work with the file system, including reading and writing files.

- Streams and Buffers:

Streams are objects that allow reading or writing data continuously. Buffers are used to handle binary data, like reading files or streams.

3. Working with Express

- RESTful API principles:

RESTful APIs use HTTP methods (GET, POST, PUT, DELETE) for CRUD operations. Each URL represents a resource, and the HTTP methods perform actions on these resources.

- Middleware in Express:

Middleware functions are used to process requests before they reach the route handler. They can be used for logging, authentication, or validation.

- Error Handling:

Errors are handled using try/catch blocks in route handlers. Express also has built-in error-handling middleware.

- Body Parsing and Content-Type Handling:

Body parsers like 'express.json()' and 'express.urlencoded()' are used to parse request bodies for JSON and form data.

4. Authentication with Express

- Authentication vs Authorization:

Authentication verifies identity (e.g., logging in), while authorization determines whether a user can access a resource (e.g., role-based access).

- JWT (JSON Web Tokens):

JWT is a compact and self-contained way of transmitting data between parties. It is often used for authentication and authorization.

- next() function:

The next() function is used to pass control to the next middleware function in Express.

- Role-Based Access Control (RBAC):

RBAC allows access control based on user roles. Middleware can check a user's role and grant or deny access to routes based on the user's role.

- bcrypt for password hashing:

bcrypt is used to hash passwords before storing them in the database to ensure security.

5. Full Stack Integration and Best Practices

- MongoDB integration:

MongoDB is used for storing data in a NoSQL format. Mongoose provides an easy way to work with MongoDB in Node.js.

- CORS (Cross-Origin Resource Sharing):

CORS is a policy that allows or restricts resources to be requested from different origins. It is important for security and to allow frontend-backend communication across domains.

- Axios:

Axios is a promise-based HTTP client used for making API requests from the frontend.

- Deployment strategies:

Full-stack applications can be deployed using platforms like Heroku, AWS, or DigitalOcean. Best practices include setting up environment variables, ensuring proper error handling, and optimizing performance.