Question 1:

1. Define Node.js as used in backend development.

Answer:

1. **JavaScript Runtime**: Node.js is an open-source, cross-platform JavaScript runtime environment.
2. **Server-Side Execution**: It allows developers to execute JavaScript code on the server-side, outside of a web browser.
3. **Backend Development**: Primarily used for building scalable network applications like APIs, web servers, and real-time services.

Question 2:

2. What is the role of npm in a Node.js project?

Answer:

1. **Dependency Management:** Installs, updates, and manages external libraries and packages (dependencies) required by the project.
2. **Project Script Runner:** Executes predefined project-specific scripts (e.g., `start`, `test`, `build`) defined in `package.json`.
3. **Project Initialization:** Helps set up new Node.js projects and manages project metadata via the `package.json` file.

Question 3:

3. Explain what is meant by dependencies in software development.

Answer:

1. **Main Point:** A dependency is a component (code, library, service, or resource) that a software module or application requires to function correctly.
2. **Supporting Point:** These can be internal (other modules within the same project) or external (third-party libraries, APIs, operating system services, or specific hardware).
3. **Example:** A Python application might have a dependency on the `requests` library to make HTTP calls, or a web frontend might depend on a JavaScript framework like React.

Question 4:

4. Define a backend application and give one example.

Answer:

1. A backend application runs on a server, handling the core logic, data storage, and processing requests.
2. It operates behind the scenes, invisible to the end-user, and communicates with frontend applications or other services.

3. Example: The server-side API that processes user logins and retrieves data for a social media website.

Question 5:

5. What is the purpose of Express.js in Node.js??

Answer:

1. **Web Application Framework:** Express.js is a fast, unopinionated, minimalist web framework for Node.js.
2. **Simplifies Development:** It provides a robust set of features to build web applications and APIs more easily and efficiently.
3. **Core Functionality:** Offers tools for routing, middleware management, template engine integration, and handling HTTP requests and responses.

Question 6:

6. State the function of Nodemon during development.

Answer:

1. **Automatic Restart:** Nodemon monitors your Node.js application files and automatically restarts the server when changes are detected.
2. **Workflow Efficiency:** This eliminates the need for manual server restarts, significantly accelerating the development feedback loop.
3. **Live Development:** It enables a "live reload" experience, allowing developers to see code changes reflected almost instantly without interruption.

Question 7:

7. What are routes in a Node.js backend system?

Answer:

1. **Definition:** Routes define specific endpoints (URLs and HTTP methods) that a server listens for.
2. **Function:** Each route maps an incoming request to a corresponding handler function that processes the request and sends a response.
3. **Application:** They structure the API, allowing different actions for paths like `/users` (GET for list, POST for create) or `/products/:id` (GET for specific product).

Question 8:

8. List any two categories of Node.js client libraries.

Answer:

No answer generated. Please try again.

Question 9:

9. Give one use of an HTTP client library in Node.js..

Answer:

1. **Main point:** To programmatically make outgoing HTTP/HTTPS requests from a Node.js application.
2. **Supporting point:** This enables interaction with external web services, APIs, or other servers.
3. **Example or application:** Fetching data from a third-party REST API, such as retrieving weather forecasts or user information.

Question 10:

10. Describe the function of Postman in backend testing.

Answer:

1. **API Request Execution**: Postman allows users to send various HTTP requests (GET, POST, PUT, DELETE) to backend API endpoints.
2. **Response Validation**: It enables inspection and validation of API responses, including status codes, headers, and JSON/XML body data.
3. **Test Automation**: Users can create and automate test suites for APIs, chaining requests, and adding assertions to validate backend logic and data integrity.

Question 11:

11. Differentiate between ES5 and ES6 in terms of variable declaration and function syntax.

Answer:

Here's the differentiation:

1. **Variable Declaration:**
   * **ES5:** Primarily uses `var`, which is function-scoped and hoisted.
   * **ES6:** Introduced `let` and `const`, which are block-scoped and not hoisted to the top of their block. `let` allows re-assignment, while `const` requires initialization and prevents re-assignment.
   * *Example:* `var x = 1;` (ES5) vs. `let y = 2; const z = 3;` (ES6)

2. **Function Syntax:**
   * **ES5:** Uses the `function` keyword for declarations and expressions.
   * **ES6:** Introduced concise arrow functions (`=>`). Arrow functions have a lexical `this` binding (inherit `this` from the parent scope), unlike traditional functions where `this` is dynamic.
   * *

Question 12:
12. Explain the difference between a class and an object in JavaScript.

Answer:
1. **Class:** A blueprint or template for creating objects, defining their common properties and methods. In JavaScript, classes are syntactic sugar over prototype-based inheritance.
2. **Object:** A concrete instance created from a class. It's a specific entity that embodies the structure and behavior defined by its class, holding its own unique data.
3. **Instantiation:** You create an object (instance) from a class using the `new` keyword (e.g., `let myCar = new Car();`).

Question 13:
13. Describe how a method differs from a property in object-oriented programming.

Answer:
1. **Properties** represent data or attributes of an object (what it *is* or *has*).
Methods represent actions or behaviors an object can perform (what it *does*).
2. Properties typically store and retrieve values (like a variable), while methods contain executable code to perform an operation.
3. **Example**: For a `User` object, `UserName` would be a property, and `Login()` would be a method.

Question 14:
14. State two advantages of using arrow functions in ES6.

Answer:
No answer generated. Please try again.

Question 15:
15. Explain how template literals improve string handling in ES6.

Answer:
1. **Enhanced Readability & Interpolation:** Template literals (using backticks `` ` ``) allow direct embedding of variables and expressions using `${expression}` within strings.
2. **Multi-line Support:** They natively support multi-line strings without requiring `\n` escape sequences or string concatenation across lines.
3. **Example:** `const user = "Bob"; const message = `Hello, ${user}!
   Welcome to our site.`;`

Question 16:

16. Define an API and state its importance in backend development.

Answer:

1. **Definition:** An API (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate and exchange data.
2. **Importance in Backend:** APIs are crucial in backend development as they expose the backend's functionalities (data, business logic) to be consumed by frontend applications, mobile apps, or other services.
3. **Application:** They enable modularity, reusability, and seamless integration, allowing a mobile app to fetch user data from a database via a REST API endpoint, or a web service to interact with a payment gateway.

Question 17:

17. What is meant by the term modular code in JavaScript development?

Answer:

1. **Definition:** Code organized into independent, self-contained units (modules), each encapsulating specific functionality.
2. **Benefits:** Promotes reusability, maintainability, and separation of concerns by preventing global scope pollution and managing dependencies clearly.
3. **Implementation:** Achieved primarily using ES Modules (`import`/`export`), CommonJS (`require`/`module.exports`), or older patterns like IIFEs.

Question 18:

18. State the purpose of using environment variables in backend systems.

Answer:

1. **External Configuration:** Decouple configuration settings (e.g., database URLs, API keys, port numbers) from the application code.
2. **Security:** Securely store sensitive credentials, preventing them from being hardcoded or committed to version control.
3. **Environment-Specific Settings:** Enable easy adaptation of application behavior across different deployment environments (development, staging, production) without code changes.

Question 19:

19. Give one difference between relational and non-relational databases.

Answer:

1. **Schema Enforcement:** Relational databases enforce a predefined, rigid schema (tables, columns, data types) that data must conform to.

2. **Schema Flexibility:** Non-relational (NoSQL) databases, conversely, are schema-less or have flexible schemas, allowing for dynamic and evolving data structures.
3. **Application:** SQL databases (e.g., PostgreSQL) are relational; Document databases (e.g., MongoDB) are non-relational.

Question 20:

20. With examples, describe how Postman is used to test API routes in a Node.js backend.

Answer:

1. **Simulating Client Requests:** Postman sends various HTTP requests (GET, POST, PUT, DELETE) to your Node.js API endpoints, mimicking how a frontend or another service would interact with it.
2. **Configuring Request Details:** It allows precise configuration of the request, including the HTTP method, the full URL (e.g., `http://localhost:3000/api/users`), custom headers (e.g., `Authorization`, `Content-Type`), and request bodies (e.g., JSON payloads for POST/PUT).
3. **Validating Responses:** Postman displays the API's response (status code, body, headers, response time), enabling you to verify if the Node.js backend returns the expected data, correct status codes (e.g., 200 OK, 201 Created, 404 Not Found), and handles errors appropriately.

* **Example (GET):** To test a `GET /api/users` route, set the method to `GET`, URL to `http://localhost:3000/api/users`, and click Send. Verify the returned JSON array of user objects.
* **Example (POST):** To test a `POST /api/products` route, set the method to `POST`, URL to `http://localhost:3000/api/products`, add `Content-Type: application/json` header, and include a JSON body like `{"name": "New Product", "price": 99.99}`. Verify the 201 Created status and the newly created product data in the response.

Question 21:

21. Describe three common categories of Node.js libraries and state one example for

Answer:

Here are three common categories of Node.js libraries:

1. **Web Frameworks**: Used for building web applications and APIs.
   * Example: Express.js
2. **Utility Libraries**: Provide general-purpose functions for common programming tasks.
   * Example: Lodash
3. **Database Drivers/ORMs**: Enable interaction with various databases.
   * Example: Mongoose

Question 22:

22. With an example, explain how the

Answer:

The question "22. With an example, explain how the" is incomplete.

1. **Incomplete Question:** The question is missing the subject or concept to be explained (e.g., "how the *system works*", "how the *algorithm functions*").
2. **General Explanation:** To explain a technical concept with an example, one typically defines the concept, outlines its mechanism or purpose, and then illustrates it with a specific, practical scenario.
3. **Example Structure (Generic):** If the question were "explain how the *encryption process* works," an example would involve showing a cleartext message being transformed into ciphertext using a specific key.