1. What is Mongoose and why do we prefer to use it in NODE instead of the default MongoDB provider?

   Mongoose is a Node.js library that provides a higher-level abstraction for working with MongoDB, a NoSQL database. It offers features like schema definition, data validation, query building, middleware, and object-document mapping (ODM). Using Mongoose in Node.js simplifies MongoDB interactions and enhances productivity.
   Here are a few reasons why developers prefer using Mongoose in Node.js instead of the default MongoDB provider:

   - **Schema Definition:** Mongoose allows you to define a schema for your data. With a schema, you can define the structure, data types, validation rules, and default values for your documents.

   - **Middleware and Hooks:** Mongoose provides middleware and hooks that allow you to define pre- and post-save, update, remove, and query hooks. These hooks enable you to perform operations such as data validation, pre-processing, or triggering actions before or after certain database operations.

   - **Data Validation:** Mongoose includes built-in validation support for your data. This helps in maintaining data integrity and preventing invalid data from being stored in the database.

   - **Middleware and Plugins:** Mongoose allows you to define custom middleware and plugins to extend its functionality.

2. What is Content Delivery Network (CDN) and why do we prefer it in production mode?
   A Content Delivery Network (CDN) is a distributed network of servers located in different geographical locations. Its primary purpose is to deliver web content, such as images, videos, JavaScript files, and CSS files, to end users with high performance and availability.

   When a user requests content from a website, the CDN determines the server closest to the user's location and delivers the content from that server. This helps in reducing the latency and network congestion that can occur when delivering content from a single origin server.

   Here are a few reasons why CDNs are preferred in production mode:

   - Improved Performance: CDNs are designed to deliver content with optimal speed and performance. By caching content on servers located closer to end users, CDNs can reduce the distance and network hops required to fetch content. This leads to faster load times and improved user experience.

   - Scalability and High Availability: CDNs are built to handle high volumes of traffic and distribute the load across multiple servers. They are designed with redundancy and failover mechanisms to ensure that content is always available even if some servers experience issues or become overloaded. This improves the overall reliability and availability of the website or application.

- Global Reach: CDNs have a widespread network of servers located in various regions and countries around the world. This enables content to be delivered quickly to users regardless of their geographic location. It helps in reducing latency and ensures a consistent experience for users across different regions.

- Bandwidth Optimization: CDNs can help optimize bandwidth usage by offloading the delivery of static content from the origin server. Since CDN servers are geographically distributed, they can handle a significant portion of the content requests, reducing the load on the origin server. This allows the origin server to focus on processing dynamic content or performing other essential tasks.

- Cost Efficiency: By offloading content delivery to a CDN, organizations can reduce the bandwidth and infrastructure costs associated with serving content directly from their origin server. CDNs typically offer pricing models based on usage, allowing businesses to scale their content delivery costs according to their needs.

3. What are development environments? List down at least three different environments
Development environments, also known as software development environments or IDEs, are software tools or platforms that provide a comprehensive set of features and tools to facilitate software development. They offer an integrated workspace for writing, debugging, testing, and deploying code.
1. Local Development Environment
2. Development and Staging Servers
3. Cloud Development Environments

4. Following Questions are regarding token-based authentication.
a) Assume you are using jQuery to write pseudo codes of two methods named login and signup to send Ajax requests to Express Endpoints to carry out both procedures. Use the console.log method for error reporting.

```
// Login method
function login(username, password) {
 $.ajax({
  url: '/login',
  method: 'POST',
  data: {username: username, password: password },
  success: function(response) {
   // Handle successful login
   console.log ('Logged in successfully');
   console.log ('Token:', response.token);
  },
  error: function(error) {
   // Handle login error
   console.log ('Login error:', error);
  }
```

```
  });
}

// Signup method
function signup(username, password) {
  $.ajax({
    url: '/signup',
    method: 'POST',
    data: { username: username, password: password },
    success: function(response) {
      // Handle successful signup
      console.log('Signed up successfully');
      console.log('Token:', response.token);
      // Save the token for future requests
      // Example: localStorage.setItem('token', response.token);
    },
    error: function(error) {
      // Handle signup error
      console.log('Signup error:', error);
    }
  });
}
```

b) In which format passwords should be stored in a database.
Passwords should be securely hashed and stored in the database instead of plain text. Hashing is a one-way process that transforms the password into an irreversible string of characters. This helps protect the passwords in case of a data breach.

c) Using jQuery where will you save the token for future requests?
When using jQuery, the token can be saved in the client-side storage such as localStorage, sessionStorage, or cookies. For example, you can save the token in the browser's localStorage for future requests:

```
// Save token in localStorage
localStorage.setItem('token', response.token);
```

d) Fill the following table for login and signup routes in Express.

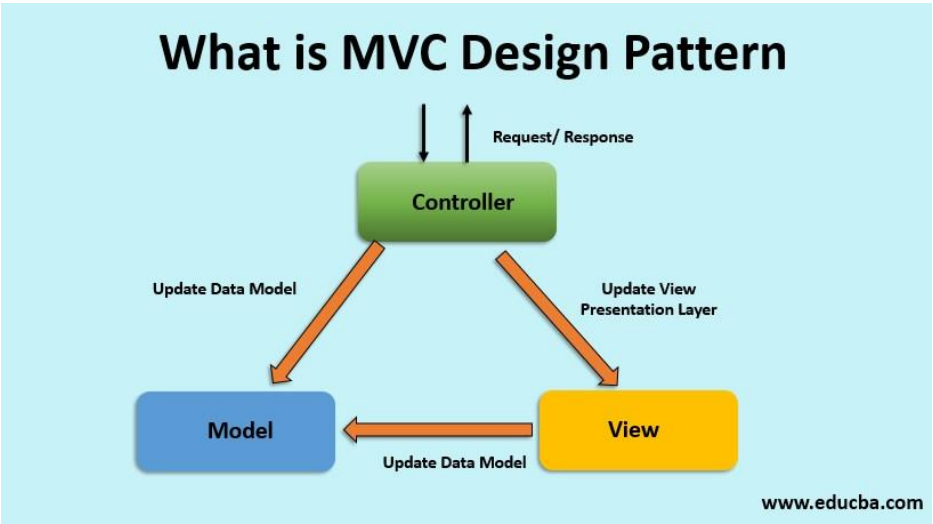
5. What is Model, View, and Controller architecture?
Here is a brief explanation of each component in the MVC architecture:

- **Model:** The Model represents the application's data and business logic. It encapsulates the data storage, retrieval, and manipulation operations. It defines the structure and behavior of the data and provides methods to interact with the data. The Model is responsible for maintaining data integrity and implementing the business rules of the application.

- **View**: The View is responsible for the presentation and rendering of the application's user interface. It displays the data to the users in a visually appealing and interactive manner. The View receives data from the Model and presents it to the users, often in the form of HTML, CSS, and JavaScript. It may also handle user interactions and events and send them to the Controller for further processing.

- **Controller:** The Controller acts as an intermediary between the Model and the View. It receives input and user actions from the View, processes them, and interacts with the Model to perform appropriate actions. The Controller contains the application's logic and is responsible for handling user requests, updating the Model, and selecting the appropriate View to display the results. It manages the flow of data between the Model and the View and ensures the separation of concerns between the two.

  Draw its architectural diagram.



  In EXPRESS which code segment plays the role of each component?

  In Express, routing and request handling can be implemented using route handlers and middleware functions. These code segments in Express can be considered as playing the role of the Controller component, while the View-related code can be implemented using template engines or frontend frameworks, and the Model-related code can be implemented using database libraries or ORM frameworks. The specific division of code segments into Model, View, and Controller components may vary based on the application's design and architectural preferences.

6. What is the difference between server-side rendering and client-side rendering? Also, explain tools and techniques which we learned during rendering methods.

   - Server-side Rendering (SSR):
     Server-side rendering involves generating the HTML content on the server and sending a fully rendered page to the client's browser. The server processes the request, retrieves data from databases or external APIs, renders the HTML templates, and sends the complete HTML response to the client. The client's browser receives and displays the rendered page without any additional processing.
     React with Express: You can use the Express framework on the server side to handle routing and render React components.

   - Client-side rendering involves sending the raw HTML, CSS, and JavaScript files to the client's browser. The browser then executes the JavaScript code, fetches data from APIs asynchronously, and dynamically renders the content on the client side.
     React: React itself can handle client-side rendering. By including the necessary JavaScript files in the HTML and configuring a root element, React can render components on the client side.

7. It is said that NODE's performance is better than other server-side languages. NODE is single-threaded architecture, and languages like PHP, JAVA, etc are multithreaded. This statement reams wrong as multi-threaded applications perform better than single-threaded applications.
   Do you see this statement? Give reasons for your arguments.

   It is not necessarily true that multi-threaded applications always perform better than single-threaded applications. The performance of an application depends on various factors, including the specific use case, the nature of the workload, and the efficiency of the language runtime and associated frameworks.

   For example, Node.js may be a great choice for building highly scalable, I/O-intensive applications, while multi-threaded languages like Java or PHP may be better suited for CPU-bound tasks or complex enterprise applications.

8. Suppose we want to build an e-commerce site where instead of selling products we want to give an option of a barter system. Products should be listed, edited, and deleted by users. Users should be able to register and log in. A user should be able to list all of the products posted by other users and select one for the barter system with one of his/her own's, Explain with the help of a diagram what is MVC architecture and identify models, controllers, and views. We only want to use server-side rendering.

9. Explain what a view engine in modern-day frameworks is and how we can add one in express Suppose you choose EJS for your express application. List down its limitations and strengths.

Limitations:
EJS doesn't have built-in support for layout templates or partials. You need to use additional libraries or custom solutions to achieve these functionalities.
EJS can sometimes lead to mixing logic and presentation code if not used carefully, as JavaScript code can be embedded directly in the templates.

Strengths:
EJS is easy to learn and use, especially for developers already familiar with HTML and JavaScript.
It provides flexibility in generating dynamic content by allowing JavaScript expressions and control structures within templates.
EJS has a large community and good documentation, making it well-supported and widely used.

10. Write down the differences between cookies and local storage.
Cookies:

- Cookies are small text files stored in the browser's memory or on disk.
- Cookies are sent to the server with each request, including subsequent requests for the same domain.
- Cookies have a limited storage capacity (typically a few kilobytes).
- Cookies can have an expiration date, after which they are automatically deleted.
- Cookies can be accessed and modified by both the client-side JavaScript code and the server.

Local Storage:

- Local storage is a storage mechanism provided by the browser.
- Local storage is persistent and remains stored on the client even after the browser is closed.
- Local storage has a larger storage capacity (typically several megabytes).
- Local storage is accessible only by client-side JavaScript code and not sent to the server with each request.
- Local storage doesn't have an expiration date, and data remains stored until explicitly cleared by the user or removed programmatically.

Give a scenario in which cookies are not preferred.
When dealing with sensitive data: Cookies are sent with each request to the server, including subsequent requests for the same domain. If you're dealing with sensitive data like passwords or personal information, storing them in cookies poses a security risk. It's better to use other methods like session storage or token-based authentication for such scenarios.

11. Write down the differences between client-side and server-side rendering. Also, explain for which type of web applications server-side rendering must be implemented.

12. What is Object-Relational Mapping (ORM) what benefits does it provide in terms of code production?

Object-Relational Mapping (ORM) is a technique that allows developers to interact with a relational database using object-oriented programming constructs. It provides a higher-level abstraction over database operations, making it easier to work with databases and reducing the amount of boilerplate code needed.

Benefits of using ORM in terms of code production include:

Increased productivity: ORM tools provide high-level APIs and abstractions that simplify database operations. Developers can work with objects and classes instead of writing raw SQL queries, saving time and effort.

Improved maintainability: ORM tools handle database operations and generate SQL queries automatically based on object mappings. This reduces the need for manual query management and makes code more readable and maintainable.

Database independence: With ORM, the codebase becomes less tightly coupled to a specific database system. Developers can switch between different database systems (e.g., MySQL, PostgreSQL) by making configuration changes rather than rewriting SQL queries.

Built-in data validation and type safety: ORM frameworks often provide features for data validation, ensuring that data stored in the database meets specified constraints. They also enforce type safety, reducing the chances of data-related errors.

Object-oriented approach: ORM allows developers to work with objects and relationships, mapping them to database tables and rows. This aligns well with the object-oriented nature of many programming languages, making it easier to reason about and manipulate data.

13. Suppose a user is logged in with session-based authentication from two different browsers. Explain whether two sessions will be maintained, or a single session will handle the authentication. Justify your answer.

When a user is logged in with session-based authentication from two different browsers, two separate sessions will be maintained. Each browser will have its own session with its own session ID. This allows the user to be logged in and authenticated independently in each browser. The server will manage and track these sessions separately to handle authentication requests from each browser.

14. What is JSON Web token-based authentication and how it is different from session-based authentication?

JSON Web Token (JWT) is a stateless authentication mechanism that is used to securely transmit information between parties as a JSON object. It is different from session-based authentication in the following ways:

Stateless: JWTs do not require the server to store session information. The token itself contains the necessary authentication data, including user information and expiration.
Scalable: Since JWTs are stateless, they can be easily scaled across multiple servers or services, making them suitable for distributed systems.
Cross-domain authentication: JWTs can be used for authentication across different domains or services, as the token itself carries the authentication information.
No server-side sessions: With JWTs, there is no need to maintain server-side sessions. This makes JWTs suitable for stateless APIs and microservices architectures.
Reduced server load: Since JWTs are self-contained, the server does not need to query a database or session store for each authentication request. This improves performance and reduces server load.

15. What is middleware in modern-day frameworks? How does it work? Make a middleware that should log the req object on the console.

Middleware in modern-day frameworks, including Express, is a function that sits between the server and the route handler and has access to the request and response objects. It allows for additional processing of the request or response before or after the route handler is executed.

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

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

// Other routes and middleware

app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

16. Explain how environment variables (eg, DB connection string, jut secret, etc.) are handled in express for different environments.
Environment variables in Express for different environments are typically handled using configuration files or by setting them directly in the deployment environment.

17. How Bootstrap Grid System helps to create Responsive Layouts?

The Bootstrap Grid System is a responsive layout system provided by the Bootstrap framework. It helps create responsive layouts by dividing the web page into a grid of rows and columns.
The benefits of using the Bootstrap Grid System include:

Responsive design: The grid system automatically adjusts the layout based on the screen size, making it easy to create responsive websites that adapt to different devices.
Easy column layout: The grid system provides a simple way to define the layout of columns in a row, allowing for flexible and intuitive designs.
Grid customization: The grid system can be customized to define the number of columns, breakpoints, and gutter sizes to match specific design requirements.
Consistency: The grid system helps maintain consistency across different pages and sections of a website, providing a unified layout structure.
Time-saving: By using the predefined grid classes, developers can save time and effort in creating responsive layouts from scratch.

18. What are the benefits of using external stylesheets over internal?
Separation of concerns: External stylesheets allow for a clear separation of HTML structure and presentation, making the code more maintainable and easier to understand.

Reusability: External stylesheets can be reused across multiple HTML files, resulting in cleaner code and reducing redundancy.
Caching and performance: External stylesheets can be cached by the browser, which improves page load time for subsequent visits as the CSS file is only downloaded once.

Browser compatibility: External stylesheets allow for better browser compatibility and consistent styling across different pages.
Collaboration: When working in a team, using external stylesheets enables parallel development and easier collaboration between designers and developers.

19. Explain with examples in how many ways you can embed your JS in your HTML?

```
<script>
  // Inline JavaScript code
</script>

<script src="script.js"></script>

<button onclick="myFunction()">Click me</button>
```

20. What is the difference between onload () and document.ready()?

onload(): This event is triggered when the entire webpage, including all external resources like images, stylesheets, and scripts, has finished loading. It is commonly used to perform actions that require the entire page to be loaded, such as initializing components or making API requests.

Example:

```
window.onload = function() {
  // Code to be executed when the page has finished loading
};
```

document.ready(): This event is triggered when the DOM (Document Object Model) has finished loading, but external resources like images may still be loading. It is often used to execute JavaScript code that manipulates or interacts with the DOM elements. Example using jQuery:

```
$(document).ready(function() {
  // Code to be executed when the DOM is ready
});
```

21. How does an asynchronous function improves webpage load time? Explain with an example.

Asynchronous functions improve webpage load time by allowing the non-blocking execution of tasks. When a task is asynchronous, it doesn't block the execution of subsequent code, which means the browser can continue loading and rendering the page while the task is being processed.

```
console.log('Start');

setTimeout(function() {
  console.log('Async task');
}, 2000);

console.log('End');
```

22. Promise() takes two functions as arguments explain both and details.

The Promise constructor takes two functions as arguments: resolve and reject. These functions are used to control the state and outcome of the Promise.

resolve: This function is called when the asynchronous operation succeeds or completes. It changes the Promise's state to "fulfilled" and passes a value (result) to the Promise's .then() method. For example:

```
const promise = new Promise((resolve, reject) => {
  // Asynchronous operation
  // ...
  resolve('Success');
});

promise.then((result) => {
  console.log(result); // Output: Success
});
```

reject: This function is called when the asynchronous operation fails or encounters an error. It changes the Promise's state to "rejected" and passes an error object to the Promise's .catch() method. For example:

```
const promise = new Promise((resolve, reject) => {
  // Asynchronous operation
  // ...
  reject(new Error('Something went wrong'));
});

promise.catch((error) => {
  console.log(error.message); // Output: Something went wrong
});
```

23. Why is routing handled at both the frontend and the backend in MERN-based applications? Provide one example to justify your answer.

24. Which JavaScript request handling approach is used at the backend to deal with performing operations on MongoDB? Provide one example to justify your answer.

Mongoose provides a straightforward way to model and interact with a MongoDB database using JavaScript objects. It abstracts away the complexity of working directly with the MongoDB driver and simplifies database operations.

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


// Connect to MongoDB
mongoose.connect('mongodb://localhost/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
```

```
});

// Define a schema for a collection
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
});

// Create a model based on the schema
const User = mongoose.model('User', userSchema);

// Perform operations on MongoDB using the model
User.find({}, (err, users) => {
  if (err) {
    console.error(err);
  } else {
    console.log(users);
  }
});

// Other operations (create, update, delete, etc.) can be performed in a similar manner
```

25. Write two custom middleware's called "authenticate" and "authorize". The authenticate middleware passes control to the authorize middleware.

```
// authenticate middleware
const authenticate = (req, res, next) => {
  // Authentication logic
  const isAuthenticated = true; // Example authentication check
```

```
  if (isAuthenticated) {
    // User is authenticated
    // Proceed to the next middleware
    next();
  } else {
    // User is not authenticated
    res.status(401).send('Unauthorized');
  }
};

// authorize middleware
const authorize = (req, res, next) => {
  // Authorization logic
  const isAdmin = true; // Example authorization check

  if (isAdmin) {
    // User is authorized
    // Proceed to the next middleware
    next();
  } else {
    // User is not authorized
    res.status(403).send('Forbidden');
  }
};

// Example usage of the middleware
app.get('/api/protected', authenticate, authorize, (req, res) => {
  // Route handler for protected API endpoint
  res.send('Protected data');
```

});

26. Line 1 in exam.js returns an error because it is a blocking code. Implement exam.js using callback, promise, and async and await calls to remove the error/s. The code returns cgpa as an object with value cgpa is

```
exam.js >...
const cgpa = getCGPA(SP19-BCS-000);
function getCGPA(id){
setTimeout(()= > {
console.log("Fetching CGPA...");
}, 1000)
return cgpa;
}
```

27. Assume the use of MongoDB, create collection "animals" to hold document with the following key value pairs- name of type String, legs of type Number, and availability of type Boolean. Write complete code including the use of any package, if required.

28. HTTP protocol helps clients interact with the server. Describe any two HTTP request methods used to transfer data from client to server.

Two HTTP request methods used to transfer data from client to server are:

a) POST: The POST method is commonly used to send data from the client to the server in the body of the request. It is typically used when submitting forms or sending data that modifies the server's state. In a POST request, the data is sent in the request body, which can be in various formats such as JSON, form data, or XML. The server can then process the data and perform the necessary actions.

b) PUT: The PUT method is used to update or replace an existing resource on the server. It sends the entire representation of the resource to be updated in the request body. PUT requests are idempotent, meaning that multiple identical requests should have the same effect as a single request. This method is commonly used in RESTful APIs for updating resources.

Also, discuss any two methods used to perform operations on MongoDB at the backend.

```
db.collection('users').insertOne({
  name: 'John Doe',
  age: 25,
  email: 'johndoe@example.com'
});
```

```
db.collection('users').updateOne(
```

```
  { name: 'John Doe' },
  { $set: { age: 26 } }
);
```

29. Discuss the change in at least three files/folders when a third-party package is installed in a MERN stack application.?

When a third-party package is installed in a MERN stack application, typically three files/folders are affected:

a) package.json: The package.json file is updated with information about the installed package, including its name, version, dependencies, and other metadata. It keeps track of all the installed packages and their versions in the application.

b) package-lock.json: The package-lock.json file is automatically generated by npm (Node Package Manager) when installing packages. It includes detailed information about the exact versions of the installed packages, their dependencies, and the entire dependency tree. It ensures that the exact same versions of the packages are installed across different environments and provides deterministic builds.

c) node_modules folder: The node_modules folder is created or updated to include the actual code files of the installed package and its dependencies. It contains all the package files required by the application to run. The package and its dependencies are installed in this folder.

What is the relation between Package.json and Package_lock.json files?

The package.json file serves as a manifest for the project, listing all the packages required for the application. It allows developers to manage dependencies, specify version constraints, and ensure consistent installations across different environments. The package-lock.json file is used by npm to ensure reproducibility and deterministic builds by locking down the exact versions of the installed packages.

Why Express JS is useful to handle backend operations instead of core Node JS?

Express.js is a web application framework built on top of Node.js. It simplifies the process of handling backend operations by providing a set of abstractions and middleware functions. Express.js makes it easier to handle routing, request processing, response generation, middleware integration, and more. It provides a higher level of abstraction and a simpler API compared to core Node.js, allowing developers to build robust and scalable web applications more efficiently. Express.js abstracts away many low-level details and provides a structured approach to building web applications, making it a popular choice for backend development in the Node.js ecosystem.

30. Why Asynchronous Javascript is preferred over Synchronous JavaScript considering the performance of the webpage (Consider Page load Time). Explain with an example.

When JavaScript code is executed synchronously, it blocks the execution of other tasks until the current task is completed. This can lead to a delay in processing and executing subsequent code, resulting in a slower page load time and potentially unresponsive user interfaces.

On the other hand, asynchronous JavaScript allows non-blocking execution of code. It allows tasks to be scheduled and executed in the background, without blocking the main execution thread. Asynchronous operations, such as AJAX requests or fetching resources, can be initiated while other parts of the code continue to execute. Once the asynchronous operation completes, a callback or promise is triggered to handle the result.

```
// Synchronous Code
console.log("Step 1");
console.log("Step 2");
console.log("Step 3");

// Asynchronous Code
console.log("Step 1");
setTimeout(function() {
  console.log("Step 2");
}, 1000);
console.log("Step 3");
```

31. Explain $.ajax, $.get and $.post each with an example.

The $.ajax method is a versatile function that allows you to make AJAX requests using various HTTP methods (GET, POST, PUT, DELETE, etc.) and configure various options.

```
$.ajax({
  url: "/api/data",
  method: "GET",
  success: function(response) {
    console.log(response);
  },
  error: function(xhr, status, error) {
    console.error(error);
  }
});
```

The $.get method is a shorthand function for making AJAX GET requests. It simplifies the syntax by providing a shorter way to make GET requests.

```
$.get("/api/data", function(response) {
  console.log(response);
});
```

The $.post method is a shorthand function for making AJAX POST requests. It simplifies the syntax by providing a shorter way to make POST requests.

```
$.post("/api/data", { name: "John", age: 25 }, function(response) {
  console.log(response);
});
```

32. Explain the usage of $ function with at least three different examples.

The $ function in jQuery is a shorthand alias for the jQuery function. It is the main entry point to access and manipulate DOM elements, handle events, perform animations, make AJAX requests, and more. Here are three examples of how the $ function can be used:

a) Selecting elements by CSS selector:

```
// Select all paragraphs on the page
$("p").text("Hello, World!");
```

b) Creating new DOM elements:

```
// Create a new div element
var $div = $("<div>").text("This is a new div");
```

c) Handling events:

```
// Add a click event handler to a button
$("#myButton").click(function() {
  console.log("Button clicked!");
});
```

33. Suppose we have following ajax requests prepared in JQuery. Each request send two numbers to back end with different configurations. You are to make express and points to handle sach request Suppose express is already setup and server is created with a variable app

A.

```
$.ajax({
method: "GET",
url: "api/calculator/add",
data: {a: 5, b: 6},
success: function (res) {
console.log(res);
//11 should be printed
},
});
```

B.
```
$.ajax({
method: "GET",
url: "api/calculator/add/"+5+"/"+6,
success: function (res) {
console.log(res);
},
});
```

C.
```
$.ajax({
method: "POST",
url: "api/calculator/add",
data: {a: "Whoala", b: 6 },
success: function (res) {
console.log(res);
//response should be an error message
},
});
```

34. Make an HTML form with id="myForm" to submit email and password to express. But instead of form submission stop its submission and make an ajax call only if the user has given email and password both. Starter Code is given below

```
$(function(){
S("#myForm").on("submit", function (e){
e.preventDefault();
//write your code here at the back of this sheet
})
})
```

```
$(function() {
```

```
$("#myForm").on("submit", function(e) {
  e.preventDefault();

  // Retrieve email and password values from the form
  var email = $("#email").val();
  var password = $("#password").val();

  // Check if both email and password are provided
  if (email && password) {
    // AJAX call to the Express endpoint
    $.ajax({
      url: "/checkUser",
      method: "POST",
      data: {
        email: email,
        password: password
      },
      success: function(response) {
        // Handle the response from the server
        console.log(response);
        // Further logic for successful response
      },
      error: function(xhr, status, error) {
        // Handle errors
        console.error(error);
      }
    });
  } else {
    // Display an error message if either email or password is missing
```

```
    console.log("Please provide both email and password");
  }
 });
});
```

35. Write down the properties of JSON Web Token.

- Compact: JWTs are designed to be compact and can be transmitted easily as URL parameters, in HTTP headers, or within an HTTP request body.
- Self-contained: JWTs contain all the necessary information within the token itself, eliminating the need for server-side storage or database lookups. The token carries information about the user, claims, and metadata.
- Secure: JWTs can be digitally signed using a secret key or a public/private key pair. This signature ensures the integrity of the token and prevents tampering.
- Stateless: JWTs are stateless, meaning the server does not need to keep track of the user's session or store token-related data on the server-side. This allows for easier scalability and distributed environments.
- Extensible: JWTs support the inclusion of custom claims, allowing developers to add additional information to the token payload as needed.

36. What is a middleware in express framework? What is its role and why we use them?

Middleware in the Express framework refers to a function or set of functions that sit in between the client request and the server response. They play a crucial role in handling requests, performing operations, modifying request/response objects, and enabling additional functionality within the Express application.

Middleware is used in Express for various purposes, including:

- Authentication middleware: Verify user identity, check for valid access tokens, and restrict access to protected routes.
- Logging middleware: Log request details, errors, or other relevant information for debugging or monitoring purposes.
- Error handling middleware: Handle and format errors, and send appropriate error responses to the client.
- Route-specific middleware: Perform specific operations or checks for certain routes only.

37. What is a mongoose model? Explain with example how it helps in reducing the coding efforts time.

In the context of MongoDB and the Mongoose library, a Mongoose model represents a structured schema and a collection in the database. It provides an interface for interacting with the MongoDB collection, allowing developers to define and enforce data schemas, perform CRUD operations, and define additional methods or statics for data manipulation.

A Mongoose model helps reduce coding efforts and time by providing the following benefits:

- Schema definition: With Mongoose models, developers can define the structure and data types of documents in the collection using a schema. This eliminates the need for manual schema management and ensures data consistency.
- Validation: Mongoose allows the definition of validation rules for the schema fields, ensuring data integrity and reducing the need for manual validation checks in application code.
- CRUD operations: Mongoose models provide a high-level API for performing create, read, update, and delete (CRUD) operations on the database. This abstracts away the complexity of writing MongoDB queries and provides a more intuitive interface for data manipulation.
- Middleware and hooks: Mongoose models allow the definition of pre-save hooks, post-save hooks, and other middleware functions. This enables developers to add custom logic, perform additional operations, or modify data before or after certain actions, reducing code duplication and enhancing code maintainability.
- Querying capabilities: Mongoose models provide a powerful query API that simplifies querying the database using a fluent and chainable syntax. This reduces the effort required to write complex queries and enables developers to retrieve and manipulate data more efficiently.

38. How jQuery selectors ease manipulating HTML DOM elements. Justify the statement with an example.

jQuery is a popular JavaScript library that simplifies HTML DOM manipulation and provides a wide range of utility functions. One of the key features of jQuery is its powerful selector syntax, which makes it easy to select and manipulate HTML DOM elements.

```
<ul>
  <li class="item">Item 1</li>
  <li class="item">Item 2</li>
  <li class="item">Item 3</li>
</ul>

$(document).ready(function() {
  $('.item').addClass('highlight').each(function() {
    console.log($(this).text());
  });
});
```

In this example, the jQuery selector $('.item') selects all elements with the class "item" (the <li> elements). The addClass() method adds the "highlight" class to each selected element, and the each() method loops through the selected elements, logging their text content to the console.

39. How MongoDB is different from MySQL. What are the advantages of using MongoDB over relation Databases? List at least 5.

MongoDB and MySQL are both popular database systems, but they differ in their data models and architectural approaches. Here are five advantages of using MongoDB over relational databases like MySQL:

- Flexible Schema: MongoDB is a NoSQL document database that uses a flexible schema called BSON (Binary JSON). It allows for dynamic and schema-less data structures, meaning each document in a collection can have a different structure. This flexibility is beneficial when dealing with evolving data models or when the data doesn't fit neatly into a fixed schema, offering more agility in development compared to the rigid schema of MySQL.

- Scalability and Performance: MongoDB is designed to scale horizontally by distributing data across multiple servers or clusters. It offers automatic sharding, which allows for easy horizontal scaling by dividing data across multiple machines. This scalability enables MongoDB to handle large amounts of data and high write loads efficiently. Additionally, MongoDB's document-oriented nature and denormalized data model can provide improved read performance for certain use cases, as it allows for retrieval of complete documents in a single database query.

- JSON-like Query Language: MongoDB uses a rich query language that resembles JSON called the MongoDB Query Language (MQL). MQL supports a wide range of queries, including filtering, sorting, aggregations, and geospatial queries. This flexibility and expressiveness make it easier to work with complex data structures and perform advanced queries. In contrast, SQL queries in MySQL can be more rigid and complex for certain data models.

- High Availability and Fault Tolerance: MongoDB provides built-in support for replication and automatic failover. It allows you to create replica sets, which are multiple copies of data spread across different servers. If the primary node fails, one of the replicas automatically becomes the new primary, ensuring high availability and minimal downtime. This feature is particularly useful for applications that require continuous operation and fault tolerance.

- Schema Evolution and Agile Development: MongoDB's flexible schema makes it easier to handle schema evolution over time. With MongoDB, adding or modifying fields in a document does not require altering the entire database schema. This flexibility supports agile development practices, allowing developers to iterate and adapt their data models more easily. In contrast, relational databases like MySQL typically require more effort to modify the schema, especially when dealing with large datasets or complex relationships.